FUNDAMENTOS DE DESIGN DE SISTEMAS

AULA 4

Prof. Winston Fung

CONVERSA INICIAL

No processo de desenvolvimento de um sistema ou aplicativo, existe uma estrutura fundamental que orienta as decisões sobre tecnologias, desempenho, escalabilidade, interoperabilidade, compatibilidade e desempenho – essa é a Arquitetura de Software. Ela fornece o esquema essencial e os princípios de design que não apenas definem a organização do sistema, mas também ditam como ele vai evoluir ao longo do tempo. Compreender a Arquitetura de Software é, portanto, uma peça-chave para o desenvolvimento eficaz e a longevidade dos sistemas de software. Vamos conhecer essas definições e algumas arquiteturas de sistemas de informações.

TEMA 1 – ARQUITETURA DE SOFTWARE

No cenário em constante evolução da tecnologia, o desenvolvimento de sistemas e aplicativos tornou-se uma tarefa complexa e desafiadora. Com sistemas tornando-se cada vez mais complexos, surge a necessidade de se adotar uma estrutura robusta que garanta a definição adequada de tecnologias, desempenho, escalabilidade, interoperabilidade, compatibilidade e desempenho – a Arquitetura de Software.

Essa arquitetura busca orientar a concepção de um sistema de software estruturado, levando em consideração as necessidades do cliente e as tecnologias associadas. A ideia é que o sucesso do desenvolvimento de um sistema dependa de uma documentação bem definida de todos os seus aspectos, facilitando a comunicação entre a equipe de desenvolvimento e garantindo que a aplicação atenda aos padrões necessários para funcionar de forma eficaz.

1.1 O QUE É UM PADRÃO NA ARQUITETURA DE SOFTWARE?

Dentro da Arquitetura de Software, um padrão arquitetural é o elemento responsável por orientar uma visão alinhada aos negócios. Esse padrão serve como uma bússola, direcionando as decisões relacionadas ao projeto de software, definindo suas utilidades, linguagens, tecnologias e o relacionamento entre os subsistemas na construção de soluções para a aplicação.

1.2 O SURGIMENTO DA ARQUITETURA DE SOFTWARE

A discussão sobre estruturas sistêmicas começou no final dos anos 1960, quando cientistas começaram a debater formas de estruturar um sistema antes do seu desenvolvimento. Contudo, a Arquitetura de Software só ganhou um panorama mais formal e reconhecido na década de 1990, com a publicação do livro *Software architecture: perspectives on an emerging discipline*, por Mary Shaw e David Garlan, da Carnegie Mellon University. Nele, Shaw e Garlan apresentam várias perspectivas sobre a arquitetura de software, destacando sua importância na coordenação de grandes equipes de desenvolvimento e no gerenciamento de complexidades sistêmicas.

Além disso, a Arquitetura de Software tem desempenhado um papel cada vez mais crucial na definição de padrões e melhores práticas na engenharia de software. A norma ISO/IEC/IEEE 42010:2011, por exemplo, foi estabelecida com o objetivo de definir padrões para a criação, análise e descrição de arquiteturas de software. Essa norma representa um marco significativo na evolução da Arquitetura de Software, demonstrando seu reconhecimento e valor dentro da comunidade de engenharia de software.

A evolução do campo da Arquitetura de Software também foi moldada pela rápida evolução da tecnologia. Por exemplo, a transição para a computação em nuvem no século XXI tem tido grandes implicações para a Arquitetura de Software, exigindo novas abordagens e padrões para o design e a gestão de sistemas de software distribuídos.

Enquanto a Arquitetura de Software pode ter começado como uma maneira de gerenciar a complexidade e coordenar equipes de desenvolvimento, ela se tornou uma disciplina, com um corpo de conhecimento, padrões e práticas que continuam a evoluir para atender às demandas de um cenário tecnológico em constante mudança.

1.3 O PROFISSIONAL EM ARQUITETURA DE SOFTWARE

O papel do profissional em Arquitetura de Software vai além de meramente programar. Esse profissional deve oferecer soluções para corporações por meio do desenvolvimento de sistemas de tecnologia da informação que melhorem o desempenho das empresas.

Para isso, é necessário compreender os negócios da corporação, ter uma postura colaborativa e ter conhecimento em diversas tecnologias, serviços, normas, legislações, cloud computing e muito mais.

As habilidades desejadas de um Arquiteto de Software incluem:

- Conhecimento do domínio e tecnologias relevantes;
- Compreensão de questões técnicas para o desenvolvimento de sistemas;
- Conhecimento de técnicas de levantamento de requisitos e de métodos de modelagem e desenvolvimento de sistemas;
- Entendimento das estratégias de negócios das empresas;
- Conhecimento de processos, estratégias e produtos das empresas concorrentes;
- Capacidade de analisar, compreender, propor e criar formas inovadoras de negócios e soluções computacionais.

Ao dominar essas habilidades, o Arquiteto de Software torna-se um profissional indispensável no desenvolvimento de soluções tecnológicas que atendam às demandas do mercado e impulsionem o crescimento dos negócios.

TEMA 2 – ESTILO ARQUITETURAL

O estilo arquitetural é uma maneira de se pensar sobre a organização e o design de um sistema de software. Ele molda a maneira como vemos o sistema como um todo, permitindo a definição clara de como os componentes do sistema estão organizados e interagem entre si. O estilo arquitetural é essencial para descrever a estrutura macroscópica de um sistema de software e pode influenciar fatores como eficiência, flexibilidade, segurança e usabilidade do sistema.

O estilo arquitetural serve para caracterizar a arquitetura de software de um sistema, possibilitando a

- **Identificação de componentes:** o arquiteto identifica quais são os principais elementos que têm funcionalidades bem definidas, como um componente de cadastro de informações de usuários e um componente de autenticação de usuários em uma aplicação web;
- **Identificação de mecanismo de interação:** a comunicação entre objetos por meio de troca de mensagens constitui uma forma por meio da qual os componentes de software interagem entre si;
- **Identificação de propriedades:** o arquiteto pode analisar as propriedades oferecidas por cada estilo baseado na organização dos componentes e nos mecanismos de interação.

Por exemplo, considere a arquitetura de um aplicativo web. Podemos identificar vários componentes principais: um componente de interface do usuário, um componente de gerenciamento de sessão, um componente de banco de dados etc. Cada um desses componentes tem um papel bem definido – o componente de interface do usuário é responsável por interagir com o usuário, o componente de gerenciamento de sessão é responsável por controlar o acesso do usuário ao sistema, e o componente de banco de dados é responsável por armazenar e recuperar dados.

A maneira como esses componentes interagem é um aspecto crucial do estilo arquitetural. Em uma arquitetura de três camadas, por exemplo, o componente de interface do usuário interage com o componente de gerenciamento de sessão, que, por sua vez, interage com o componente de banco de dados. Essa estrutura permite uma separação clara de responsabilidades e facilita a manutenção e a evolução do sistema.

2.1 POR QUE O ESTILO ARQUITETURAL É IMPORTANTE?

À medida que os sistemas de software se tornam mais complexos, a necessidade de níveis mais altos de abstração se torna cada vez mais crucial. O estilo arquitetural fornece essa abstração, ajudando os envolvidos no projeto a entender a estrutura geral do sistema e a se comunicar efetivamente sobre ela.

Por exemplo, em um projeto de grande escala, pode ser útil utilizar um estilo arquitetural baseado em microsserviços. Isso permite que diferentes equipes trabalhem em diferentes serviços, cada um com sua própria base de código e estrutura de dados, o que pode aumentar a produtividade e facilitar a manutenção do sistema.

2.2 VANTAGENS DO ESTILO ARQUITETURAL

A adoção de um estilo arquitetural apropriado traz várias vantagens. Ele fornece suporte para atributos de qualidade, como desempenho, segurança e confiabilidade, ajudando a garantir que o sistema atenda a seus requisitos não funcionais.

Além disso, um estilo arquitetural bem definido facilita a diferenciação entre diferentes arquiteturas, tornando mais fácil para as partes interessadas entenderem as características únicas de cada sistema.

Além disso, um bom estilo arquitetural pode reduzir o esforço necessário para entender e manter o projeto, já que proporciona uma visão clara da organização do sistema. Ele também pode facilitar o reuso de arquitetura e conhecimento em novos projetos, acelerando o desenvolvimento e reduzindo o risco de erros.

Destacam-se as seguintes vantagens:

- Suporte a atributos de qualidade (ou requisitos não funcionais);
- Diferenciação entre arquiteturas;
- Menos esforço para entender o projeto;
- Reuso de arquitetura e conhecimento em novos projetos;
- Suporte ao planejamento e a gerência da manutenção e integridade da solução.

O estilo arquitetural fornece suporte para o planejamento e a gerência da manutenção do sistema, ajudando a garantir a integridade da solução ao longo do tempo. Por exemplo, se um sistema é projetado com um estilo arquitetural baseado em componentes, é mais fácil planejar e gerenciar atualizações e melhorias para cada componente individualmente, em vez de ter que atualizar todo o sistema de uma vez.

Dessa forma, o estilo arquitetural é uma ferramenta crucial para projetar, implementar e manter sistemas de software eficientes e eficazes. Ele permite uma compreensão clara da estrutura geral do sistema, facilita a comunicação entre as partes interessadas e ajuda a garantir que o sistema atenda às suas necessidades e requisitos.

TEMA 3 – DOCUMENTAÇÃO DA ARQUITETURA DE SOFTWARE

A documentação da arquitetura de software é um elemento crucial do desenvolvimento de sistemas, atuando como um registro duradouro das decisões e estruturas que moldam um sistema de software. Como o Software Engineering Institute da Carnegie Mellon University observa, a documentação da arquitetura de software "fala pelo arquiteto, hoje, amanhã e daqui a 20 anos" ("*Software architecture documentation speak for the architect, today, tomorrow and 20 Years from now*"). Essa documentação é o elo entre a concepção do sistema e sua implementação, manutenção e evolução. Dessa forma, a documentação de software é uma parte vital de qualquer projeto de desenvolvimento de software. Ela serve como um mapa, orientando os desenvolvedores e ajudando-os a entender a estrutura, as funções e os processos do software.

Para quem está iniciando os estudos, a documentação pode parecer um detalhe secundário em comparação ao próprio código. No entanto, é preciso enfatizar que a documentação de software é tão crucial quanto o código. É por meio dela que se descreve o que o software faz, como ele o faz e por que faz daquela forma, permitindo que outras pessoas entendam e trabalhem com o software de maneira mais eficaz.

Com frequência, ocorrem situações em que as arquiteturas de software são criadas sem uma documentação efetiva, resultando em uma comunicação inadequada. Isso significa que os desenvolvedores e outros envolvidos no sistema não têm acesso a uma representação adequada da arquitetura.

Na documentação, destaca-se a criação de documentos que:

- Definem as atividades que serão realizadas;
- São os primeiros artefatos a agregar qualidade;
- São os melhores artefatos nas primeiras fases do desenvolvimento;
- São elementos-chave para a posterior manutenção do sistema.

A documentação de software é tão importante que existe uma norma internacional dedicada a ela, a ISO/IEC/IEEE 42010:2011. Essa norma define um conjunto de diretrizes sobre como a documentação de software deve ser criada, o que deve incluir e como deve ser organizada, ajudando a equipe de desenvolvimento a criar documentação de alta qualidade que possa atender às necessidades de uma variedade de partes interessadas.

No entanto, apesar de sua importância, a documentação de software ainda é frequentemente negligenciada ou tratada como um incômodo. Essa é uma mentalidade que deve ser desafiada. Uma boa documentação de software não apenas facilita o trabalho dos desenvolvedores, mas também permite que o software seja mantido e evoluído ao longo do tempo de uma forma sustentável.

Além disso, ela pode ser uma ferramenta valiosa para a comunicação entre diferentes partes interessadas, desde gerentes de projeto e usuários até outros desenvolvedores e membros da equipe.

Embora a documentação de software possa parecer um tópico árido e teórico, é na verdade uma parte fundamental do desenvolvimento de software e uma habilidade que vale a pena aprender e dominar. Afinal, um código bem escrito pode ser uma obra de arte, mas sem uma documentação adequada, ele pode ser tão inacessível e inútil quanto uma obra de arte trancada em um cofre.

3.1 PRINCÍPIOS DE UMA BOA DOCUMENTAÇÃO DE SOFTWARE

A documentação eficaz da arquitetura de software é uma tarefa complexa e desafiadora. É preciso equilibrar o nível de detalhes, clareza, organização e atualização da documentação. Aqui estão alguns princípios fundamentais para a criação de documentação de arquitetura de software eficaz:

- **Documentar o ponto de vista do usuário**: a documentação deve ser escrita tendo em mente as pessoas que vão utilizá-la. Elas podem ser desenvolvedores, gerentes de projeto, analistas de qualidade ou partes interessadas do negócio. É importante usar uma linguagem clara, evitar jargões desnecessários e fornecer informações relevantes para cada tipo de usuário;
- **Evitar ambiguidades**: a documentação deve ser precisa e inequívoca. Evitar o uso de termos vagos ou confusos e garantir que os conceitos sejam claramente definidos e consistentemente usados em toda a documentação;
- **Usar um modelo ou template**: a documentação deve ser organizada de maneira padrão para facilitar a navegação e a compreensão. Um modelo bem projetado pode ajudar a garantir que todas as áreas importantes sejam cobertas e que a informação seja apresentada de forma clara e coerente;
- **Evitar repetições desnecessárias**: embora alguma redundância possa ser útil para fins de clareza, a repetição desnecessária pode tornar a documentação confusa e difícil de navegar. No entanto, em alguns casos, a redundância pode ser aceitável, por exemplo, quando é necessário entender o contexto ou a relação entre as informações;

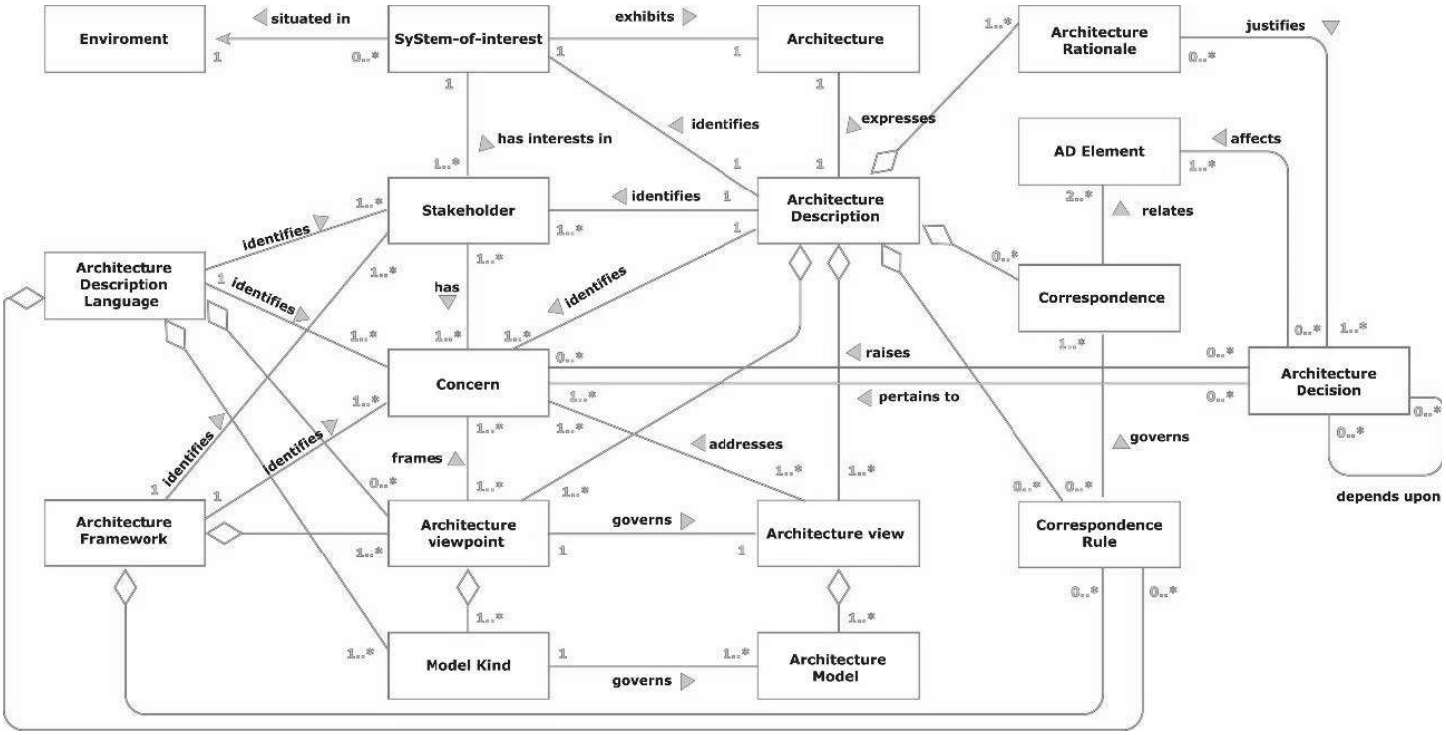
- **Documentar as razões para as decisões tomadas**: é importante registrar não apenas as decisões tomadas, mas também as razões por trás delas. Isso pode incluir discussões sobre alternativas consideradas e rejeitadas, bem como a lógica por trás da decisão final;
- **Manter a documentação atualizada**: a documentação deve refletir o sistema como ele é, não como era. Isso significa que a documentação deve ser atualizada sempre que o sistema for modificado;
- **Revisar a documentação criada**: a revisão da documentação por pares pode ajudar a garantir que ela seja clara, precisa e completa;
- **Tornar a documentação acessível a todos os participantes do projeto**: todos os que estão envolvidos no projeto devem ter acesso à documentação. Isso inclui não apenas a equipe de desenvolvimento, mas também os interessados, como os gerentes de projeto, cliente e os usuários finais.

3.2 ISO/IEC/IEEE 42010:2011

A norma ISO/IEC/IEEE 42010:2011, também conhecida como "Sistemas e engenharia de software – Arquitetura de descrição", desempenha um papel crítico no campo da Arquitetura de Software. Ela estabelece um padrão para a descrição da arquitetura de sistemas e softwares, proporcionando um modelo conceitual que permite aos arquitetos de software descrever e comunicar efetivamente a arquitetura de um sistema.

Vale destacar que a norma não se limita a especificar um formato ou estrutura específica para uma descrição da arquitetura. Em vez disso, ela define um conjunto de conceitos e relações que formam a base para tais descrições. Isso dá aos arquitetos de software a liberdade de adaptar a norma às suas necessidades específicas, enquanto garante que a descrição da arquitetura seja consistente, compreensível e útil para todas as partes interessadas.

Figura 1 – Mapeamento sobre descrição de arquitetura de software com base na ISSO/IEC/IEEE 42010:2011



Fonte: ISO/IEC/IEEE 42010:2011.

A ISO/IEC/IEEE 42010:2011 baseia-se em várias visões para documentar a de um sistema. As "visões" são como diferentes lentes por meio das quais podemos examinar a arquitetura de um sistema, cada uma focada em um aspecto específico. Por exemplo, a visão lógica trata da funcionalidade do sistema, a visão de desenvolvimento foca na integração de componentes, a visão de processo trata da gestão da implementação e a visão física lida com funcionalidades para o usuário final.

Ao utilizar essas visões, é possível validar o design da arquitetura em relação aos casos de uso do sistema. Isso garante que a arquitetura seja capaz de suportar todos os requisitos do sistema, tanto funcionais quanto não funcionais.

Em um cenário de futuro, em que a arquitetura de software deve lidar com desafios crescentes, como computação em nuvem, microsserviços e inteligência artificial, a norma ISO/IEC/IEEE 42010:2011 continua sendo uma ferramenta valiosa. Ao garantir uma documentação clara, consistente e eficaz da arquitetura de software, ela ajuda a garantir que os sistemas sejam construídos de maneira eficiente e eficaz, prontos para enfrentar os desafios do futuro.

3.3 AS VIEWS DA ISO/IEC/IEEE 42010:2011

A norma ISO/IEC/IEEE 42010:2011 nos introduz a quatro tipos distintos de 'views': Lógica, Desenvolvimento, Processo e Física. Cada uma dessas 'views' enfoca um aspecto específico do sistema, permitindo aos arquitetos, desenvolvedores e outras partes interessadas entender diferentes facetas do sistema.

Figura 2 – Representação das visões de arquitetura de software segundo a ISO/IEC/IEEE 42010:2011

Crédito: Winston Sem Lun Fung/Freepik.

- **Visão Lógica**: essa visão concentra-se na funcionalidade do sistema. Ela descreve os principais elementos funcionais do sistema, suas responsabilidades, propriedades e interfaces;
- **Visão de Desenvolvimento**: essa visão aborda a estrutura do software, incluindo seus componentes e subcomponentes. Isso é particularmente útil para os desenvolvedores que estão construindo e integrando o software;
- **Visão do Processo**: essa visão trata do aspecto dinâmico do sistema, incluindo o processamento de tarefas, concorrência e sincronização;
- **Visão Física**: essa visão descreve a infraestrutura necessária para o sistema, incluindo hardware, software de sistema e redes.

Essas visões, juntas, fornecem uma visão completa e compreensiva do sistema e sua arquitetura. Cada uma delas é relevante para diferentes partes interessadas e serve a diferentes propósitos. A

ISO/IEC/IEEE 42010:2011 fornece um padrão útil e eficaz para a documentação da arquitetura de software.

TEMA 4 – OS DIFERENTES MODELOS ARQUITETURAIS

Os modelos arquiteturais de software são uma forma de organizar os elementos funcionais de um sistema de software. Eles ajudam a definir a estrutura geral de um sistema, especificando como as diferentes partes de um sistema vão interagir. Existem vários modelos arquiteturais que são comumente usados, e a escolha de um modelo depende das necessidades específicas do sistema em questão.

4.1 ARQUITETURA EM CAMADAS (LAYERED PATTERN)

A arquitetura em camadas organiza um sistema em um conjunto de camadas, cada uma das quais fornece serviços para a camada acima dela. Esse modelo de arquitetura é especialmente útil para o desenvolvimento de sistemas complexos, pois permite que diferentes partes do sistema sejam desenvolvidas e atualizadas de forma independente.

Figura 3 – Exemplo de Arquitetura em camadas

Fonte: Fung, 2023.

Pense na arquitetura em camadas como um bolo de vários andares. Cada andar do bolo é como uma camada no seu software.

Cada camada tem uma responsabilidade específica, e cada uma se comunica com as camadas acima e abaixo dela.

Por exemplo, em um aplicativo de e-commerce: a camada inferior (como a base do bolo) pode ser a camada de dados, que lida com tudo relacionado ao armazenamento e recuperação de informações do banco de dados.

O próximo andar do bolo é a camada de negócios, que processa as regras do e-commerce, como calcular descontos ou verificar a disponibilidade de estoque. O andar superior do bolo é a camada de apresentação, que lida com a interação com o usuário – tudo o que o usuário vê e com o que interage no site.

4.2 ARQUITETURA CLIENTE-SERVIDOR (CLIENT-SERVER PATTERN)

A arquitetura cliente-servidor é um modelo em que um servidor fornece serviços, e um ou mais clientes consomem esses serviços. O servidor mantém e gerencia a maior parte dos recursos de dados e funções de negócios, enquanto o cliente é responsável por solicitar esses serviços e apresentar os resultados ao usuário.

Figura 4 – Exemplo da arquitetura cliente-servidor

Fonte: Fung, 2023.

A arquitetura cliente-servidor é como um restaurante. O cliente (você, pedindo comida) faz um pedido (solicita um serviço), e o servidor (o garçom) atende ao pedido e traz a comida (o serviço). No

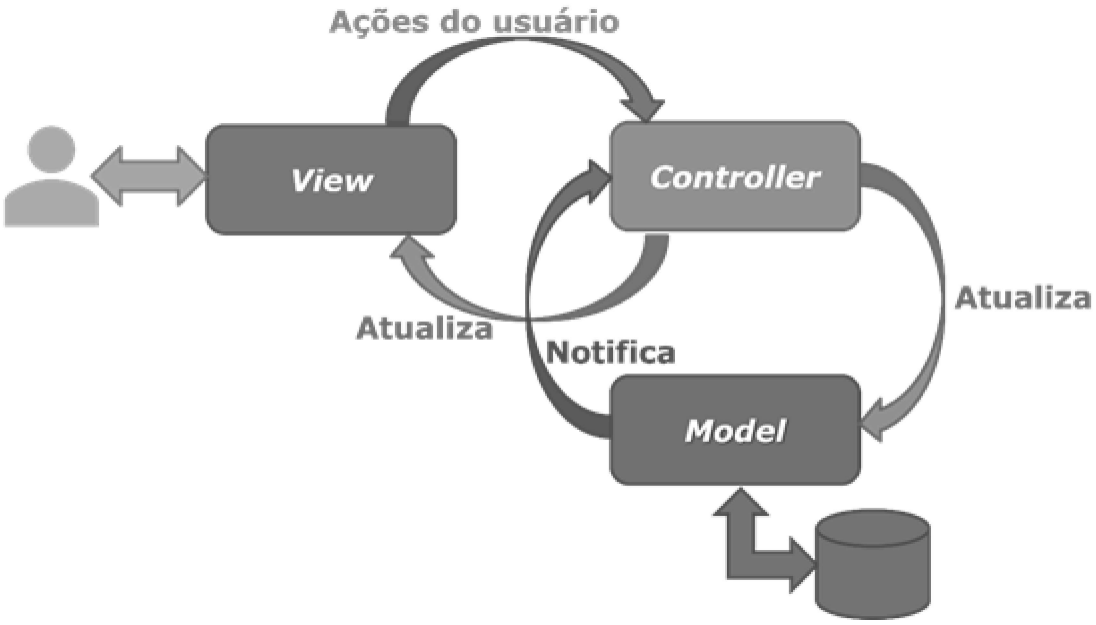
mundo dos softwares, o "cliente" é o seu computador ou aplicativo, e o "servidor" é o sistema remoto que está fornecendo os dados ou serviços que o cliente solicitou.

Por exemplo, quando você acessa seu e-mail por meio de um aplicativo, seu aplicativo (o cliente) solicita ao servidor de e-mail (o servidor) para baixar suas novas mensagens.

4.3 ARQUITETURA MODEL-VIEW-CONTROLLER PATTERN (MVC)

O padrão de arquitetura MVC divide uma aplicação em três partes interconectadas. Isso é feito para separar representações internas da informação do sistema das maneiras como a informação é apresentada e aceita do usuário.

Figura 5 – Exemplo da arquitetura MVC



Fonte: Fung, 2023.

No mundo do software:

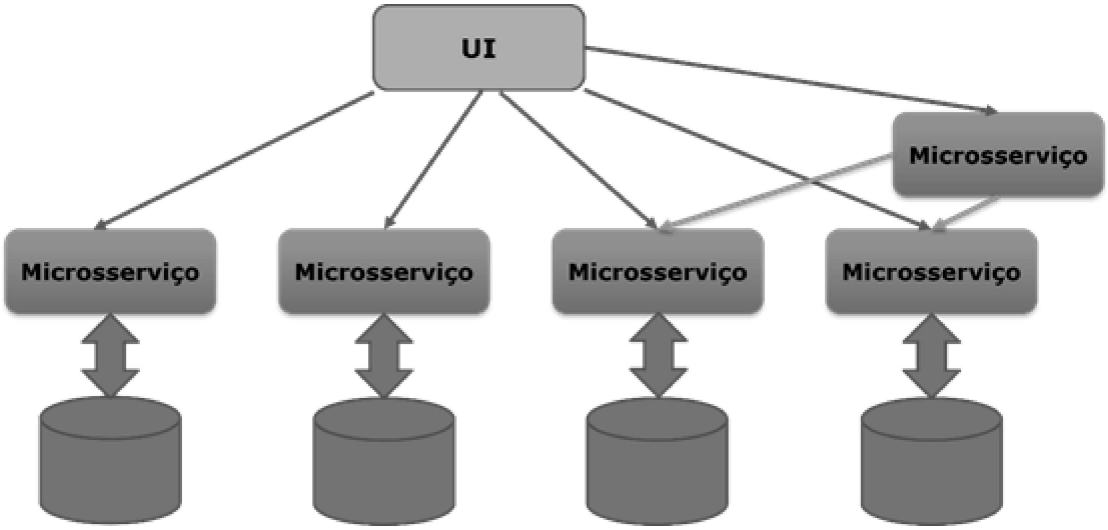
- O "Modelo" é onde os dados e as regras de negócios são armazenados;
- A "Visão" é como os dados são apresentados – é a interface do usuário;
- O "Controlador" é o que conecta a Visão e o Modelo. Ele atualiza a Visão quando o Modelo muda e atualiza o Modelo quando o usuário interage com a Visão.

4.4 ARQUITETURA DE MICROSSERVIÇOS (MICROSERVICES PATTERN)

A arquitetura de microsserviços é um estilo de arquitetura que estrutura uma aplicação como uma coleção de serviços que são:

- Altamente manuteníveis e testáveis;
- Livremente acoplados, ou seja, cada serviço pode ser atualizado independentemente dos outros;
- Independentes, o que permite que cada serviço seja implantado independentemente;
- Capazes de serem desenvolvidos e implantados por pequenas equipes.

Figura 2 – Exemplo de arquitetura de microsserviços



Fonte: Fung, 2023.

A arquitetura de microsserviços é como um conjunto de blocos de construção de Lego. Cada bloco (microsserviço) tem uma função específica e pode ser usado para construir várias estruturas diferentes (aplicativos). Se você quiser mudar ou atualizar algo, você pode simplesmente substituir ou mudar alguns blocos, sem ter que desmontar toda a estrutura.

Um exemplo de uso de microsserviços pode ser encontrado na Netflix. Eles utilizam essa arquitetura para suportar a escalabilidade e a continuidade de seus serviços de streaming. Cada parte do serviço da Netflix (recomendação de filmes, gerenciamento de contas de usuário, streaming de vídeos) é gerenciado por um microsserviço independente. Isso significa que, mesmo se uma parte do sistema falhar, as outras partes do sistema podem continuar funcionando normalmente.

TEMA 5 – O FUTURO DA ARQUITETURA DE SOFTWARE

Olhando para o futuro da arquitetura de software, podemos prever uma profunda influência de tecnologias emergentes, como computação em nuvem, inteligência artificial (IA) e microsserviços. Esses avanços estão remodelando a maneira como construímos e interagimos com os sistemas de software.

5.1 COMPUTAÇÃO EM NUVEM

A computação em nuvem é como uma biblioteca gigante que contém todos os tipos de livros (ou nesse caso, dados) que você pode acessar a qualquer momento, desde que tenha uma conexão com a Internet. Isso remove a necessidade de ter um armário de livros enorme (um servidor físico) em casa.

Assim como a biblioteca, a computação em nuvem permite que as empresas movam suas operações para a "nuvem" – servidores virtuais hospedados na Internet. Isso significa que os sistemas não estão mais restritos ao ambiente de uma empresa. Em vez disso, eles podem ser acessados de qualquer lugar, a qualquer momento.

Por exemplo, um software de gerenciamento de projetos baseado em nuvem permite que as equipes colaborem em projetos de qualquer lugar do mundo, em vez de estarem confinadas a um escritório físico.

5.2 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial é como um supercomputador que aprende com as experiências e é capaz de tomar decisões informadas com base nesse aprendizado. No mundo dos softwares, a IA pode ser usada para automatizar tarefas complexas, analisar grandes volumes de dados e melhorar a eficiência operacional.

Por exemplo, os chatbots de atendimento ao cliente, alimentados por IA, podem responder automaticamente às perguntas frequentes dos clientes, liberando os representantes humanos para lidar com consultas mais complexas. A IA também está sendo usada para prever tendências e comportamentos do consumidor, ajudando as empresas a tomarem decisões mais informadas.

5.3 MICROSSERVIÇOS

Os microsserviços são como uma equipe de especialistas, cada um focado em um aspecto específico de um projeto. Em vez de ter um único time tentando fazer tudo, temos vários times pequenos (microsserviços), cada um lidando com uma parte do projeto.

No mundo do software, isso significa que um aplicativo é dividido em pequenos serviços independentes que podem ser desenvolvidos, implantados e escalados individualmente. Isso facilita a manutenção e atualização do sistema, pois um problema em um serviço não afeta os outros.

Por exemplo, um aplicativo de comércio eletrônico pode ter um microsserviço para lidar com pedidos de clientes, outro para gerenciar o inventário e outro para processar pagamentos. Cada microsserviço pode ser desenvolvido e atualizado independentemente, melhorando a eficiência e a produtividade.

O futuro da arquitetura de software é promissor e cheio de possibilidades infinitas. À medida que continuamos a inovar e avançar, é vital que os futuros engenheiros de software entendam essas tendências emergentes e aprendam a adaptá-las às suas práticas de desenvolvimento. Essa é a chave para criar soluções de software robustas, escaláveis e eficientes que atendam às demandas em constante mudança do mundo digital.

FINALIZANDO

Nesta etapa, analisamos os diferentes aspectos da arquitetura de software, desde a compreensão dos estilos arquiteturais, a importância de uma documentação adequada e a exploração dos modelos arquiteturais predominantes até a consideração das tendências futuras. O estudo ampliou a nossa percepção de como a arquitetura de software pode ser efetivamente utilizada para impulsionar a qualidade e a eficiência do desenvolvimento de sistemas.

Visitamos a importância dos estilos arquiteturais na estruturação do sistema, a maneira como os componentes são identificados e interagem entre si e as propriedades que caracterizam cada estilo.

Esclarecemos também a importância da documentação da arquitetura de software, em que os princípios de uma boa documentação foram destacados, bem como a explanação da norma ISO/IEC/IEEE 42010:2011, que orienta a estruturação da documentação de arquitetura.

Aprofundamos nossa compreensão dos diferentes modelos arquiteturais, com ênfase nos padrões arquiteturais em camadas, cliente-servidor, MVC e microsserviços. Cada um desses modelos, conforme explicado, tem sua própria força e é aplicável a diferentes contextos de sistemas.

Olhamos também para o futuro da arquitetura de software, com destaque para o papel da computação em nuvem, inteligência artificial e microsserviços no desenvolvimento de sistemas modernos. Essas tecnologias emergentes estão proporcionando novas oportunidades para otimizar o desenvolvimento de software e enfrentar desafios complexos.

Assim, concluímos que o desenvolvimento de sistemas estruturados com uma arquitetura claramente documentada e disponibilizada para todos os participantes do projeto, aliado ao uso adequado de tecnologias modernas e emergentes, é essencial para minimizar os riscos associados ao desenvolvimento e à manutenção do sistema. Além disso, esse processo habilita o melhor aproveitamento das tecnologias, o compartilhamento efetivo de conhecimento entre a equipe e, finalmente, a produção de sistemas de software robustos e eficientes que atendam às necessidades do usuário e às demandas do mercado em constante mudança.

REFERÊNCIAS

FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. Porto Alegre: Bookman, 2007.

FREEMAN, E.; FREEMAN, E. **Use a cabeça! Padrões e projetos**. 2. ed. Rio de Janeiro: Alta Books, 2009.

GAMMA, E. et al. **Padrões de projetos**: soluções reutilizáveis de software orientados a objetos. Porto Alegre: Bookman, 2007.

NEIL, T. **Padrões de design para aplicativos móveis**. São Paulo: Novatec, 2012.