THIAGO ADRIANO

PÓS TECH

SOFTWARE ARCHITECTURE

ARQUITETURA DE SOFTWARE

AULA 01

SUMÁRIO

O QUE VEM POR AÍ?

O objetivo deste material será apresentar alguns pontos importantes sobre Arquitetura de Software, que você poderá utilizar no momento da concepção dos seus projetos.

HANDS ON

Nesta primeira videoaula, foi apresentada a importância do papel de um(a) arquiteto(a) para o planejamento de um projeto estruturado, manutenível e, principalmente, escalável. Por meio de exemplos lúdicos, como na analogia do pedreiro(a) com a pessoa desenvolvedora de software, foi demonstrado como devemos nos organizar sempre com uma documentação mínima, junto com uma estratégia, antes de construirmos os nossos sistemas.

Na outra videoaula, nós vimos um exemplo real de um projeto desenvolvido para a TV Bandeirantes. Assista para compreender ainda mais o conteúdo desta aula!

SAIBA MAIS

ARQUITETURA DE SOFTWARE

A arquitetura de software é a estrutura lógica e física de um sistema de software, incluindo os componentes, suas interações e suas restrições de design. Ela determina como um sistema deve ser construído e como os componentes devem se integrar uns aos outros.

O estudo da arquitetura de software é importante, pois ele afeta a maneira como um sistema é criado, desenvolvido, mantido, evoluído e como ele pode afetar a qualidade do sistema, na sua flexibilidade, escalabilidade, desempenho e segurança.

Vários fatores devem ser considerados na criação de uma arquitetura de software, como requisitos de negócio, restrições de tempo e orçamento, padrões de design e tecnologias utilizadas. Algumas das principais considerações dessa criação incluem a separação de preocupações, a modulação, a reutilização de componentes e a escolha de padrões de design apropriados.

Também devemos considerar as várias abordagens diferentes que podem ser utilizadas, como arquitetura baseada em camadas, arquitetura baseada em componentes e arquitetura baseada em serviços.

Cada abordagem tem suas vantagens e desvantagens, e a escolha da abordagem apropriada depende das necessidades do projeto e do contexto em que o sistema será utilizado.

Por todos esses motivos, a arquitetura de software é um tema amplo e complexo, e existem muitos livros e artigos que abordam o assunto em maior profundidade, mas um dos mais recomendados a todos que estão começando a estudar este assunto é o “Clean Architecture”, do autor Robert C Martin, que também é conhecido como "Uncle Bob".

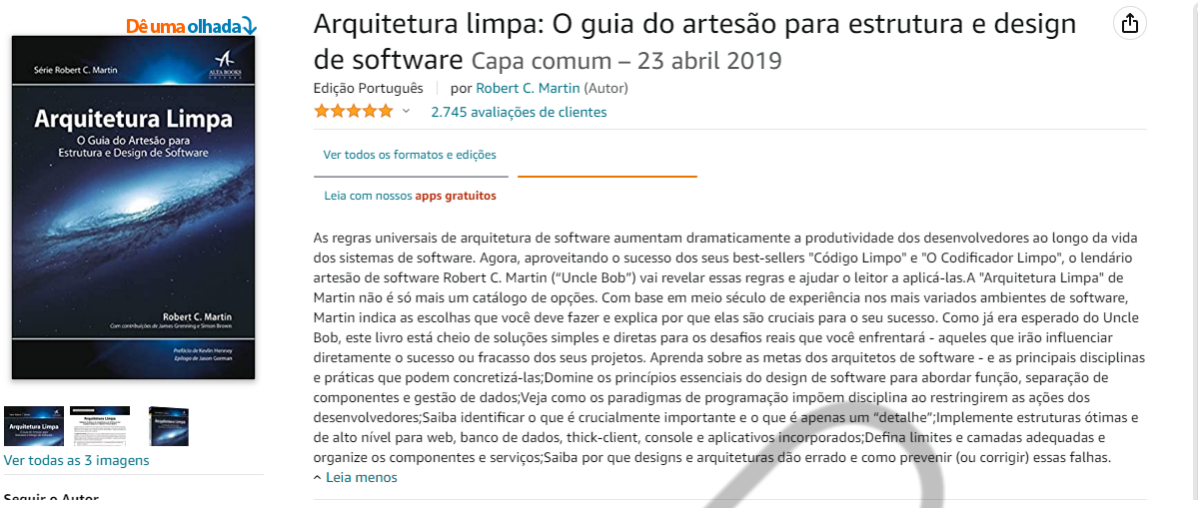

Figura 1 – Arquitetura limpa
Fonte: Amazon (2023)

Este livro é um guia para criar sistemas de software de alta qualidade. Ele apresenta um conjunto de práticas e princípios que visam ajudar os desenvolvedores a criar sistemas de software de maneira mais eficiente e sustentável.

Os principais pontos do livro incluem:

- Separação de preocupações: a ideia de que cada parte do sistema deve ter um propósito específico e não deve ser responsável por mais do que o necessário. Isso ajuda a manter o código organizado e fácil de manter.
- Responsabilidade única: cada módulo, classe ou função deve ter uma única responsabilidade e deve ser projetado de maneira a cumprir essa responsabilidade de forma eficiente.
- Abstração: a ideia de que as partes do sistema devem ser projetadas de maneira a serem independentes das implementações específicas. Isso permite que o sistema seja facilmente alterado ou ampliado no futuro.
- Depuração fácil: o sistema deve ser projetado de maneira a facilitar a depuração de problemas quando eles surgirem.
- Simplicidade: o sistema deve ser o mais simples possível, sem componentes desnecessários ou redundâncias. Isso ajuda a manter o sistema organizado e fácil de manter.

Saindo do livro, outros tópicos comuns que são discutidos em relação à arquitetura de software incluem a escolha de tecnologias, a criação de modelos de arquitetura, a avaliação e seleção de arquiteturas, a documentação de arquiteturas e o gerenciamento de mudanças em arquiteturas ao longo do tempo.

Esses pontos garantem que o projeto já tenha uma estrutura desde o seu dia 0, seja escalável e manutenível.

TIPOS DE ARQUITETURA DE SOFTWARE

Conforme citado anteriormente, a arquitetura de software é uma área importante da engenharia de software que se concentra na estruturação do software, incluindo a escolha de componentes, a definição de suas interações e a distribuição de responsabilidades. A seguir você tem alguns tópicos comuns quando falamos sobre a arquitetura de software:

- Design Patterns: padrões de design são soluções geralmente aceitas para problemas comuns na arquitetura de software. Alguns exemplos incluem o Padrão MVC, Padrão de Repositório, Padrão de Factory, entre outros.
- Microservices: é uma abordagem para o desenvolvimento de aplicativos que divide uma aplicação em pequenos serviços independentes que trabalham juntos para fornecer uma solução completa.
- Arquitetura de camadas: é uma abordagem para estruturar o software em camadas lógicas, cada uma responsável por uma funcionalidade específica.

DESIGN PATTERNS

Se você já trabalha a algum tempo na área de tecnologia, com certeza já deve ter ouvido falar sobre design patterns. Eles são soluções bem estabelecidas e comprovadas para problemas comuns na arquitetura de software que fornecem uma abordagem padronizada e comprovada para solucionar problemas específicos e ajudam a melhorar a reutilização de código e a manutenção do software.

Os designs patterns podem ser categorizados em três categorias principais: padrões de criação, padrões estruturais e padrões de comportamento.

Padrões de criação são design patterns que se concentram na criação de objetos e instâncias. Eles abordam questões comuns na criação de objetos, incluindo a ocultação da complexidade de criação, a garantia de que os objetos sejam criados corretamente e a centralização do processo de criação. Aqui estão alguns exemplos de padrões de criação:

- Padrão Factory: este padrão fornece uma interface para criar objetos em uma classe base sem especificar suas classes concretas. Isso permite a flexibilidade para adicionar novas classes sem afetar o código existente.
- Padrão Builder: este padrão separa a construção de um objeto complexo de sua representação, permitindo que diferentes representações sejam criadas com o mesmo processo de construção.
- Padrão Singleton: este padrão garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a ela.
- Padrão Prototype: este padrão especifica tipos de objetos a serem criados por uma operação de clonagem. Isso permite que novos objetos sejam criados rapidamente a partir de modelos existentes.

Padrões Estruturais

Padrões estruturais são padrões de design de software que se concentram na organização de classes e objetos para formar estruturas maiores e mais complexas. Esses padrões são úteis para simplificar e melhorar a comunicação entre diferentes partes de um sistema, bem como para reduzir a complexidade do código e aumentar sua reutilização.

Os padrões estruturais são divididos em três categorias principais: adaptação, agregação e composição. Cada categoria inclui vários padrões específicos:

Adaptação:

- Adapter: permite que objetos com interfaces incompatíveis trabalhem juntos, convertendo a interface de um objeto em outra interface que o cliente espera.
- Bridge: separa uma abstração de sua implementação, permitindo que ambas evoluam independentemente.
- Proxy: fornece um objeto representante que controla o acesso a outro objeto.

Agregação:

- Composite: compõe objetos em estruturas de árvore para representar hierarquias de partes-todo.
- Decorator: anexa responsabilidades adicionais a um objeto dinamicamente.
- Façade: fornece uma interface simplificada para um conjunto de interfaces complexas.

Composição:

- Flyweight: minimiza o uso de memória compartilhando o estado entre objetos que são idênticos ou semelhantes.
- Private Class Data: protege a integridade de dados encapsulando-os em uma classe privada.
- Bridge: separa uma abstração de sua implementação, permitindo que ambas evoluam independentemente.

Cada um desses padrões oferece uma abordagem diferente para a organização e estruturação do código, e pode ser aplicado em diferentes contextos para resolver diferentes problemas de design de software.

Padrões de Comportamento

Padrões de comportamento são padrões de design de software que lidam com algoritmos, comunicação e responsabilidades entre objetos. Esses padrões ajudam a definir como os objetos interagem e se comunicam entre si, permitindo que o software seja mais flexível e modular.

Os padrões de comportamento são divididos em três categorias principais: comportamento de classe, comportamento de objeto e comportamento de interação. Cada categoria inclui vários padrões específicos:

Comportamento de Classe:

- Template Method: define o esqueleto de um algoritmo em uma superclasse e permite que as subclasses substituam etapas específicas do algoritmo sem alterar sua estrutura.
- Strategy: define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. O padrão permite que o algoritmo varie independentemente dos clientes que o usam.
- State: permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe.

Comportamento de Objeto:

- Observer: define uma dependência um-para-muitos entre objetos para que, quando um objeto mude de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.
- Chain of Responsibility: permite que vários objetos processem uma solicitação em cadeia. Cada objeto na cadeia tem a opção de processar a solicitação ou passá-la para o próximo objeto da cadeia.
- Command: encapsula uma solicitação como um objeto, permitindo que você parametrize clientes com diferentes solicitações, fila ou registre solicitações e suporte as operações “desfazer” e “refazer”

.

Comportamento de Interação:

- Interpreter: define uma representação gramatical para um idioma e fornece um interpretador para interpretar sentenças nessa linguagem.
- Mediator: define um objeto que encapsula como um conjunto de objetos interagem. O mediador promove o acoplamento fraco, evitando que os objetos se refiram uns aos outros explicitamente e permitindo que você varie sua interação independentemente.

- Visitor: permite que você adicione novas operações a uma estrutura de objeto existente sem modificar os objetos.

Para que possamos reforçar este assunto, o uso de Design Patterns ajuda a garantir que o software seja desenvolvido de forma consistente e organizada, melhorando sua qualidade e facilitando a manutenção e atualização futuras. Além disso, eles podem ser úteis para equipes de desenvolvimento que trabalham juntas, pois fornecem uma linguagem comum e uma abordagem compartilhada para soluções de design.

ARQUITETURA DE CAMADAS

A arquitetura de camadas é um padrão arquitetural comum para a construção de sistemas de software, em que o sistema é dividido em camadas ou níveis lógicos distintos, cada um com sua própria responsabilidade e funcionalidade específicas.

Geralmente, essas camadas são organizadas em uma pilha, onde as camadas mais baixas fornecem serviços para as camadas mais altas, e as camadas mais altas usam os serviços fornecidos pelas camadas mais baixas para implementar suas funcionalidades.

As camadas mais comuns em uma arquitetura de camadas incluem:

1. Camada de apresentação ou interface do usuário: é a camada responsável pela interação do usuário com o sistema, apresentando informações e recebendo comandos.
2. Camada de aplicação: é a camada que contém a lógica de negócios do sistema, implementando a funcionalidade específica do domínio do problema.
3. Camada de persistência de dados: é a camada que gerencia a persistência de dados do sistema, incluindo a leitura e gravação de dados em bancos de dados ou outros tipos de armazenamento.

Algumas implementações de arquitetura de camadas podem incluir camadas adicionais, como uma camada de serviços ou uma camada de infraestrutura.

O principal objetivo da arquitetura de camadas é separar as preocupações do sistema em níveis distintos de abstração, permitindo que cada camada seja implementada, testada e mantida de forma independente das outras. Isso facilita a escalabilidade, o teste e a manutenção do sistema, além de melhorar sua modularidade e flexibilidade.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula tivemos uma introdução à arquitetura de software, onde os nossos experts apresentaram alguns exemplos práticos da importância de uma boa arquitetura de software e design de software.

Venha participar da nossa comunidade no Discord! Lá você pode pedir ajuda com dúvidas, conversar sobre o conteúdo das aulas, fazer networking e muito mais. Estamos te esperando!

REFERÊNCIA

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3. ed. Massachusetts: Addison-Wesley Professional, 2012.

PALAVRAS-CHAVE

Arquitetura de software. Arquitetura Limpa. Arquitetura de camadas.