

Resenha – Capítulo 7: Arquitetura e Refactoring

O capítulo 7 do livro **Engenharia de Software Moderno** apresenta um panorama detalhado sobre arquitetura de software e refactoring, destacando a importância de se estruturar um sistema de forma planejada e coerente para garantir sua escalabilidade, manutenibilidade e eficiência a longo prazo. A obra enfatiza que a arquitetura não é apenas um “layout” do sistema, mas sim uma visão estratégica que orienta a comunicação entre os diversos módulos, estabelecendo como cada parte do sistema se integra e evolui. Nesta resenha, apresento meu entendimento pessoal, utilizando exemplos práticos e minha experiência no desenvolvimento de um sistema de ecommerce.

A Importância da Arquitetura de Software

A arquitetura de software atua como um sinalizador das partes essenciais de um sistema, definindo de forma clara como os módulos interagem e se comunicam. Conforme a leitura, ter uma boa arquitetura é crucial mesmo em sistemas de baixo nível, pois ela ajuda a evitar soluções improvisadas, conhecidas popularmente como “go horse”. No meu caso, ao desenvolver um ecommerce, iniciei sem um planejamento arquitetural robusto, focando somente no aprendizado das tecnologias. Porém, essa abordagem me levou a enfrentar diversos problemas de manutenção e crescimento, forçando uma reestruturação completa do banco de dados, da estrutura de pastas e das páginas. Essa experiência reforça o argumento do capítulo: sem uma visão clara e planejada, o desenvolvimento tende a ser repleto de retrabalhos e desperdício de código.

A escolha da arquitetura adequada depende muito da complexidade e das necessidades do projeto. Em projetos menores, uma arquitetura simples pode ser suficiente, mas mesmo nesses casos, é fundamental evitar práticas que comprometam a qualidade do código a longo prazo. Uma estrutura bem definida, por exemplo, utilizando o padrão MVC (Model-View-Controller), pode facilitar a manutenção e permitir que cada componente seja alterado de forma isolada, reduzindo o acoplamento e facilitando a adição de novas funcionalidades.

MVC versus Microserviços: Escolhas para Diferentes Cenários

O capítulo também discute diferentes abordagens arquiteturais, dentre as quais se destacam o padrão MVC e a arquitetura de microserviços.

Modelo MVC

O padrão MVC divide a aplicação em três camadas principais:

- Model (Modelo): Responsável pela lógica de negócio e acesso aos dados.
- View (Visão): Responsável pela interface e apresentação dos dados ao usuário.
- Controller (Controlador): Atua como intermediário, gerenciando as interações do usuário e coordenando as respostas do sistema.

Na minha experiência, o MVC foi a abordagem mais utilizada e eficaz para projetos que não exigem uma escalabilidade extrema. Por exemplo, em um ecommerce de porte médio, onde a complexidade do sistema é moderada, o uso de MVC garante uma separação clara entre as responsabilidades. Essa separação facilita o refactoring e a manutenção do código, pois permite que cada camada seja modificada independentemente sem impactar as demais. Além disso, o MVC oferece um equilíbrio entre organização e simplicidade, permitindo que os desenvolvedores se concentrem tanto na lógica de negócio quanto na experiência do usuário.

Arquitetura de Microserviços

Por outro lado, a arquitetura de microserviços propõe a divisão da aplicação em serviços pequenos e independentes, cada um responsável por uma funcionalidade específica. Essa abordagem pode ser extremamente vantajosa em projetos de grande escala ou em sistemas que demandam alta disponibilidade e escalabilidade. No entanto, a implementação de microserviços pode ser considerada um overkill para projetos simples, como um ecommerce de pequeno porte.

Microserviços são ideais para cenários onde:

- Complexidade e Escalabilidade: A aplicação necessita lidar com uma grande quantidade de dados e alta demanda de processamento, dividindo as responsabilidades entre equipes especializadas.
- Desenvolvimento Paralelo: Diferentes equipes podem trabalhar de forma independente em serviços isolados, o que facilita a implementação de mudanças sem afetar todo o sistema.
- Isolamento de Falhas: Em sistemas críticos, a falha de um microserviço não compromete todo o funcionamento da aplicação, pois os serviços são isolados e podem ser reiniciados ou escalados individualmente.

Apesar dessas vantagens, a adoção de microserviços envolve uma complexidade adicional na comunicação entre os serviços e na gestão de dados, o que pode aumentar

o custo e o tempo de desenvolvimento se aplicado a projetos que não justifiquem tal investimento.

Refactoring: A Melhoria Contínua do Código

O capítulo também aborda o refactoring, que é o processo de reestruturação do código sem alterar sua funcionalidade externa. O objetivo principal do refactoring é melhorar a legibilidade, reduzir a complexidade e facilitar a manutenção futura do sistema. Na prática, o refactoring é indispensável para garantir que o código evolua de maneira sustentável, especialmente quando a arquitetura inicial não foi suficientemente planejada.

No meu caso, a ausência de um planejamento arquitetural robusto inicialmente me levou a uma situação onde, a cada avanço de três passos, era necessário voltar dois passos para corrigir problemas acumulados. Esse retrabalho poderia ter sido evitado com uma estruturação adequada desde o início. O refactoring, nesse contexto, serve como um alívio para as dores causadas por decisões técnicas apressadas e mal pensadas, permitindo que o sistema seja reestruturado e melhor organizado conforme ele cresce.

Aplicações Práticas no Mercado

Os conceitos abordados no capítulo podem ser aplicados em diversos contextos de mercado:

- Projetos Startups e MVPs: Para startups que desejam lançar um MVP (Minimum Viable Product) rapidamente, pode ser tentador pular a definição de uma arquitetura robusta. Contudo, mesmo nesses casos, um padrão simples como o MVC pode ser adotado para garantir que, conforme o produto evolua, a manutenção e a escalabilidade não se tornem um empecilho.
- Ecommerce e Sistemas de Gestão: Em projetos de ecommerce ou sistemas de gestão, onde a interface do usuário é importante, a separação de responsabilidades do MVC permite uma melhor organização do código, facilitando atualizações e a integração de novas funcionalidades sem comprometer a estrutura existente.
- Sistemas de Grande Escala: Para empresas que lidam com sistemas complexos, como bancos ou grandes plataformas de serviços online, a arquitetura de microserviços pode oferecer a escalabilidade e resiliência necessárias. Cada serviço

pode ser desenvolvido e escalado de forma independente, permitindo um gerenciamento mais eficiente dos recursos e a mitigação de falhas.

- Sistemas Legados: Em cenários onde há um sistema legado que precisa ser modernizado, o refactoring é essencial. Ele possibilita a gradual modernização da base de código, permitindo que novas funcionalidades sejam adicionadas sem comprometer a estabilidade do sistema atual.

Conclusão

O capítulo 7 do *Engenharia de Software Moderno* ressalta a importância de uma boa arquitetura como pilar para o sucesso de qualquer projeto de software. Minha experiência com o desenvolvimento de um e-commerce reforça a ideia de que, independentemente do porte do projeto, uma estrutura bem planejada pode evitar muitos problemas futuros. Enquanto o MVC se mostra uma opção eficaz para a maioria dos projetos, a arquitetura de microservices se reserva para casos de alta complexidade e escalabilidade. Assim, a escolha da arquitetura deve ser feita com base nas necessidades específicas de cada projeto, sempre visando a manutenibilidade e a evolução contínua do sistema.