Resumo dos Capítulos 6 e 7 de "Engenharia de Software Moderna"

Capítulo 6: Padrões de Projeto

Introdução aos Padrões de Projeto

Padrões de projeto são soluções recorrentes para problemas comuns no design de software. Eles oferecem um vocabulário comum para desenvolvedores, auxiliando na criação de designs mais flexíveis e compreensíveis. A principal ideia é capturar práticas eficazes que podem ser aplicadas em diferentes contextos.

Classificação dos Padrões

1. **Padrões Criacionais**: Abstraem a forma como objetos são instanciados.
    - Exemplos: *Factory Method*, *Abstract Factory*, *Singleton*, *Builder*, *Prototype*.
2. **Padrões Estruturais**: Tratam da composição de classes e objetos.
    - Exemplos: *Adapter*, *Composite*, *Decorator*, *Facade*, *Proxy*, *Bridge*.
3. **Padrões Comportamentais**: Focam na comunicação e na atribuição de responsabilidades entre objetos.
    - Exemplos: *Observer*, *Strategy*, *Command*, *State*, *Iterator*, *Chain of Responsibility*.

Exemplos de Padrões

- **Padrão Observer**: Define uma dependência "um-para-muitos" entre objetos, onde mudanças em um objeto são automaticamente notificadas aos dependentes.
- **Padrão Strategy**: Permite a definição de diferentes algoritmos intercambiáveis que podem ser selecionados em tempo de execução.

Aplicação dos Padrões

Os padrões de projeto devem ser usados quando realmente resolvem um problema de design. Aplicá-los indiscriminadamente pode resultar em complexidade desnecessária.

Capítulo 7: Arquitetura

O Papel da Arquitetura no Software

A arquitetura de software define a estrutura geral de um sistema e orienta decisões de design e desenvolvimento. Ela visa garantir que o sistema seja escalável, manutenível e eficiente.

Estilos Arquiteturais

1. **Arquitetura Monolítica**: Todo o código e funcionalidades estão integrados em um único bloco, simples no início, mas com desafios de escalabilidade a longo prazo.
2. **Arquitetura em Camadas**: Separa o sistema em camadas com responsabilidades distintas, facilitando o desenvolvimento e manutenção.
3. **Arquitetura Cliente-Servidor**: Estrutura o sistema em dois componentes: cliente e servidor, ideal para aplicações distribuídas.
4. **Arquitetura Microservices**: Divide o sistema em pequenos serviços independentes, permitindo maior flexibilidade, mas aumentando a complexidade operacional.
5. **Arquitetura Event-Driven**: Baseia-se em eventos para comunicação entre componentes, sendo ideal para sistemas assíncronos e desacoplados.

Padrões Arquiteturais

- **MVC (Model-View-Controller)**: Separa apresentação, controle e modelagem dos dados, bastante utilizado em aplicações web.
- **CQRS (Command Query Responsibility Segregation)**: Separa operações de leitura e escrita para otimizar sistemas complexos.

Decisões Arquiteturais

Decisões de arquitetura devem considerar fatores como:

- **Desempenho**: Garantir que o sistema responda eficientemente.
- **Escalabilidade**: O sistema deve suportar crescimento sem perda de desempenho.
- **Manutenibilidade**: Facilitar alterações e melhorias no sistema.
- **Segurança**: Proteger o sistema contra ameaças internas e externas.

Considerações Finais

A escolha da arquitetura correta é essencial para o sucesso do projeto de software, devendo evoluir conforme o sistema cresce e novas demandas surgem.