Resenha dos Capítulos 6 e 7 do livro Engenharia de Software Moderna

Aluno: João Vitor Romero Sales
Disciplina: Projeto de Software
Professor: João Paulo Aramuni

Os capítulos 6 e 7 do livro "Engenharia de Software Moderna" de Marco Tulio Valente abordam dois pilares essenciais para a construção de softwares robustos, flexíveis, de fácil manutenibilidade e entendimento: padrões de projeto e arquitetura de software.

O capítulo 6 apresenta os padrões de projeto como soluções testadas e validadas para problemas recorrentes em projetos de software. Inspirado nos trabalhos do arquiteto Christopher Alexander, o conceito se baseia na ideia de que problemas comuns no desenvolvimento de software podem ser resolvidos com soluções pré-definidas e reutilizáveis, otimizando o processo e promovendo a qualidade do código. O autor apresenta dez padrões de projeto em detalhes, com exemplos de código e cenários de aplicação.

1. Fábrica (Factory e Método Fábrica): A Fábrica e o Método Fábrica visam flexibilizar a criação de objetos, encapsulando o processo de instanciação e ocultando o tipo concreto dos objetos criados. Isso facilita a manutenção e a evolução do sistema, pois mudanças na implementação dos objetos não impactam o código cliente. Um exemplo seria a criação de objetos que representam diferentes tipos de conexão de rede (TCP, UDP), onde a fábrica centraliza a lógica de criação e o cliente apenas solicita um objeto "Conexão" genérico.

2. Singleton: O padrão Singleton garante que uma classe tenha apenas uma única instância durante a execução do programa. Ele é útil para representar recursos únicos do sistema, como logs, configurações globais e gerenciadores de conexão. A classe Singleton controla sua própria instanciação, disponibilizando um método estático para acessar a instância única. O autor alerta para o uso excessivo do Singleton como substituto para variáveis globais, o que pode aumentar o acoplamento entre classes e dificultar a criação e execução de testes.

3. Proxy: O padrão Proxy introduz um objeto intermediário (o proxy) entre um objeto base e seus clientes. O proxy controla o acesso ao objeto base, adicionando funcionalidades como cache, logging, controle de acesso e comunicação remota, sem que o objeto base precise ser modificado. Um exemplo seria um proxy para acesso a imagens em um sistema web, onde o proxy carrega as imagens sob demanda, otimizando o desempenho.

4. Adaptador: O padrão Adaptador converte a interface de uma classe para outra interface esperada pelos clientes. Ele permite a integração de classes com interfaces incompatíveis, sem a necessidade de modificar seu código fonte, por exemplo, um sistema que precisa utilizar uma biblioteca externa com uma interface diferente da esperada. O Adaptador faz a ponte entre as duas interfaces, traduzindo as chamadas dos métodos.

5. Fachada: O padrão Fachada oferece uma interface simplificada para um subsistema complexo, ocultando a complexidade interna e facilitando o uso do sistema pelos clientes, por exemplo, um sistema de e-commerce com diversos módulos, este padrão centraliza o acesso aos módulos, oferecendo uma interface unificada para o cliente.

6. Decorador: O padrão Decorador adiciona responsabilidades a um objeto dinamicamente, em tempo de execução, através de composição. Ele é uma alternativa flexível à herança, evitando a explosão de subclasses quando há múltiplas combinações de funcionalidades opcionais. Um exemplo seria um sistema de processamento de texto, onde funcionalidades como negrito, itálico e sublinhado podem ser adicionadas como decoradores a um objeto "Texto".

7. Strategy: O padrão Strategy permite a troca de algoritmos utilizados por uma classe em tempo de execução. Por exemplo, um sistema de ordenação de dados que precisa suportar diferentes algoritmos (bubblesort, quicksort, mergesort). O Strategy encapsula cada algoritmo em uma classe separada, permitindo que o cliente escolha o algoritmo desejado em tempo de execução.

8. Observador: O padrão Observador define uma dependência um-para-muitos entre objetos. Quando o estado de um objeto (o sujeito) muda, todos os seus dependentes (os observadores) são notificados e podem reagir à mudança. O Observador é frequentemente utilizado na implementação de interfaces gráficas reativas, onde a atualização da interface é disparada por mudanças nos dados.

9. Template Method: O padrão Template Method define o esqueleto de um algoritmo em uma classe abstrata, delegando a implementação de alguns passos para subclasses. Um exemplo seria um framework para desenvolvimento web que define um fluxo básico para processamento de requisições, mas permite que cada aplicação personalize a lógica de negócio de cada passo.

10. Visitor: O padrão Visitor permite adicionar novas operações a uma hierarquia de classes, sem modificar o código dessas classes. Ele encapsula a lógica da operação em um objeto "Visitor", que visita cada objeto da hierarquia e executa a operação.

O autor enfatiza a importância do design for change, projetando com foco na flexibilidade para futuras mudanças, o que facilita a manutenção e a evolução do sistema. Também destaca que o uso de padrões deve ser criterioso, evitando a paternite, ou seja, a aplicação excessiva e desnecessária de padrões, o que pode tornar o sistema complexo e difícil de entender.

O capítulo 7 aborda a arquitetura de software, definindo-a como o "projeto em alto nível" que determina a estrutura fundamental do sistema, organizando suas unidades maiores e mais relevantes para os objetivos do sistema, como camadas, módulos e serviços. Essa estrutura define como os componentes do sistema interagem entre si, impactando diretamente na manutenibilidade, escalabilidade e desempenho do software. A arquitetura também é mostrada como o conjunto de decisões de projeto mais importantes e difíceis de reverter, com impacto a longo prazo no sistema. O autor discute diversos padrões arquiteturais e suas características.

1. Arquitetura em Camadas: A Arquitetura em Camadas organiza o sistema em módulos hierárquicos, com cada camada oferecendo serviços para a camada superior. A Arquitetura em Três Camadas, frequentemente utilizada em sistemas corporativos, divide o sistema em: Interface com o Usuário (apresentação), Lógica de Negócio (aplicação) e Banco de Dados (persistência). Essa separação promove modularidade, reuso e facilita a manutenção do software.

2. MVC (Model-View-Controller): O MVC separa a interface do usuário (View), a lógica de negócios (Model) e o fluxo de controle (Controller). Ele promove a especialização do trabalho, reuso de componentes, testabilidade e facilita a criação de múltiplas interfaces para o mesmo modelo de dados. O autor destaca a evolução do MVC, desde sua aplicação original em interfaces gráficas desktop até sua adaptação para sistemas web com o uso de frameworks como Ruby on Rails, Django e Spring.

3. Microsserviços: Microsserviços são um padrão arquitetural que divide o sistema em processos independentes e autônomos, comunicando-se entre si através de APIs leves. Cada microsserviço representa uma funcionalidade específica do negócio, com sua própria base de dados e ciclo de desenvolvimento. Essa abordagem promove o desenvolvimento e deploy ágeis, escalabilidade granular, uso de diferentes tecnologias e tolerância a falhas. O autor discute o contexto que levou ao surgimento dos microsserviços, como a necessidade de maior agilidade e flexibilidade no desenvolvimento de software, os desafios relacionados à sua implementação, como a complexidade da comunicação entre serviços, e quando não utilizar este padrão arquitetural, pois afinal não é uma bala de prata.

4. Filas de Mensagens e Publish/Subscribe: Esses padrões são utilizados em sistemas distribuídos para promover o desacoplamento no tempo e espaço. Em

Filas de Mensagens, a comunicação entre os componentes é assíncrona, mediada por uma fila onde os produtores inserem mensagens e os consumidores as processam. Já em Publish/Subscribe, os produtores publicam eventos para um tópico e os consumidores se inscrevem nos tópicos de seu interesse, recebendo notificações quando um evento relevante ocorre.

5. Pipes e Filtros: Pipes e Filtros é um padrão orientado a dados, onde os componentes (filtros) processam os dados recebidos na entrada e geram uma nova saída, comunicando entre si através de buffers (pipes). Essa abordagem é ideal para processamento flexível e paralelo, permitindo a combinação de diferentes filtros para realizar tarefas complexas.

6. Anti-padrões: "Big Ball of Mud": O autor finaliza o capítulo apresentando um anti-padrão arquitetural, o "Big Ball of Mud", caracterizado pela falta de estrutura definida, com alta dependência entre os módulos e código complexo e difícil de entender. Essa situação geralmente surge quando o sistema cresce sem um planejamento arquitetural adequado, e cheio de dívidas técnicas, tornando a manutenção e a evolução do software impossíveis.

Os capítulos 6 e 7 do livro Engenharia de Software Moderna demonstram a importância crucial de padrões de projeto e arquitetura de software para o desenvolvimento de sistemas de alta qualidade. Através de exemplos concretos e discussões detalhadas, o autor guia o leitor na compreensão de conceitos e na escolha das melhores soluções para diferentes cenários. O livro enfatiza a importância de uma análise criteriosa para a escolha das soluções mais adequadas, evitando a aplicação indiscriminada de padrões e a complexidade desnecessária, visando a construção de softwares robustos, flexíveis e de fácil manutenção.