PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Disciplina: Projeto de Software
Professor: João Paulo Carneiro Aramuni
Aluno: João Pedro Santana Marques

Capítulo 5: Princípios de Projeto

O primeiro capítulo aborda maneiras de como dividir problemas complicados em partes menores e mais fáceis de lidar, facilitando nosso papel como engenheiros de software. A ideia é criar sistemas que sejam mais fáceis de entender, manter e expandir.

O primeiro conceito discutido é a **Integridade Conceitual**, que defende a necessidade de coerência e coesão no projeto de software. Para alcançar essa integridade, é recomendável que um único arquiteto ou uma equipe enxuta seja responsável pelas principais decisões do projeto, garantindo um sistema somente com as funcionalidades necessárias.

Outro princípio abordado é o **Ocultamento de Informação**, que sugere que os módulos (classes) de um sistema devem expor apenas o necessário, escondendo detalhes de implementação que estão sujeitos a mudanças. Isso reduz o acoplamento entre componentes e facilita futuras modificações. Por exemplo, ao desenvolver uma biblioteca de funções matemáticas (`Math`), é preferível expor apenas as interfaces públicas, ocultando implementações específicas que podem ser alteradas sem impactar os usuários da biblioteca. No contexto de orientação a objetos, o uso de getters e setters também pode ser uma forma de ocultamento de informação. Esses métodos são utilizados para acessar dados privados de uma classe de maneira controlada, garantindo que a estrutura interna da classe permaneça oculta aos usuários externos. No entanto, é importante considerar se a liberação de um dado privado é realmente necessária, pois o uso indiscriminado de getters e setters pode violar o princípio de ocultamento de informação.

A **Coesão** é o grau em que as responsabilidades de um módulo são relacionadas entre si, isto é, toda classe deve implementar uma única funcionalidade ou serviço. Módulos altamente coesos são desejáveis, pois são mais fáceis de entender e manter. Além disso, a separação de interesses é uma propriedade desejável em projetos de software que se assemelha ao conceito de coesão. Essas são as recomendações equiparáveis: (1) uma classe deve ter uma única responsabilidade; (2) uma classe deve implementar um único interesse; (3) uma classe deve ser coesa.

Já o **Acoplamento** fala sobre o nível de dependência entre módulos; um baixo acoplamento é preferível para minimizar o impacto de mudanças em outras partes do sistema. Acoplamento aceitável é quando a classe A usa só os métodos públicos da classe B, e a interface dada por B é estável, ou seja, não fica mudando toda hora. Por outro lado, acoplamento ruim acontece quando a classe A depende demais da classe B de um jeito que qualquer mudança em B pode quebrar A. O problema do acoplamento ruim é que a dependência entre as classes não é mediada por uma interface estável. Ou seja, uma classe pode mudar algo sem nem perceber que vai afetar outra. Já no acoplamento aceitável, a classe que muda a interface sabe que isso vai impactar outras, porque a interface é justamente o contrato que ela oferece pro sistema.

O capítulo ainda apresenta ainda uma seção sobre **SOLID** e outros princípios de projeto diversos princípios de projeto. Princípios de projeto são regras mais específicas que ajudam na criação de um software, auxiliando no atendimento às propriedades de projeto vistas anteriormente. Cinco

dos princípios abordados são conhecidos como Princípios SOLID: Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle. O objetivo desses princípios não é apenas resolver problemas específicos, mas também garantir que o sistema seja flexível para manutenções e evoluções no futuro, considerando que requisitos e tecnologias mudam frequentemente.

Responsabilidade Única: Esse princípio é uma aplicação direta da ideia de coesão. Cada classe ou módulo deve ter uma única responsabilidade ou motivo para mudar.

Segregação de Interfaces: É um caso particular de Responsabilidade Única com foco em interfaces. Esse princípio define que interfaces devem ser coesas e específicas para um tipo de cliente.

Inversão de Dependências (Prefira Interfaces a Classes): Uma classe cliente não deve depender de implementações concretas (classes), mas sim de abstrações (interfaces).

Aberto/Fechado: Módulos devem ser abertos para extensão, mas fechados para modificação. O princípio tem como objetivo a construção de classes flexíveis e extensíveis

Substituição de Liskov: Subtipos (classes filhas) devem ser substituíveis por seus tipos base (classes mães) sem alterar o comportamento esperado do sistema.

Prefira Composição a Herança: A composição oferece maior flexibilidade ao permitir a combinação de comportamentos em tempo de execução.

Princípio de Demeter (Princípio do Menor Conhecimento): Um módulo deve ter conhecimento limitado sobre outros módulos, interagindo apenas com seus dependentes diretos.

Para finalizar, as **Métricas de Código Fonte** são ferramentas quantitativas utilizadas para avaliar propriedades específicas de um projeto de software, como tamanho, coesão, acoplamento e complexidade. Essas métricas são calculadas a partir da análise do código fonte já implementado, fornecendo valores numéricos que permitem uma avaliação mais objetiva da qualidade do projeto.

Aplicação no Mercado: Desenvolvimento de Microsserviços: Ao aplicar o princípio da Responsabilidade Única, uma empresa pode dividir um sistema em microsserviços, onde cada serviço tem uma função específica. Por exemplo, um comércio online pode ter um microserviço para gerenciar pedidos, outro para pagamentos e outro para catálogo de produtos, assim como no exemplo do Carnaval feito em sala.