PUC Minas

Resenha

Projeto de Software

Gustavo Delfino Guimarães

1489062@sga.pucminas.br

Matrícula: 836079

Belo Horizonte, MG – Fevereiro de 2025

Engenharia de Software Moderna

Capítulo 5: Princípios de Projeto

O capítulo 5 do livro “Engenharia de Software Moderna” traz à tona boa parte do que se é estudado desde o primeiro período do curso de Engenharia de Software em matérias como Fundamentos da Engenharia de Software, Programação Modular, Arquitetura de Computadores e Engenharia de Requisitos, assim abordando os princípios para um bom desenvolvimento de software.

Ao início do capítulo é destacado o tema do projeto, que foi definido como “Solução para determinado problema”, que traduzindo para a Engenharia de Software seria atender os requisitos funcionais e não funcionais propostos para um sistema. Para solucionar o problema, o autor coloca em pauta de facilitar essa solução como quebrar o problema em problemas menores, combater a complexidade e abstração do problema. Ao ler o início do capítulo foi impossível não lembrar das aulas de programação modular do professor Danilo, onde fora explicado a partir de diagramas UML a forma de abstração de problemas da vida real.

A partir dessa breve introdução o autor colocou 4 temas em evidência:

- **Integridade conceitual:** Definido por Frederick Brooks, o autor coloca que um sistema não pode ser um amontoado de funcionalidades sem coesão e coerência. A partir disso um sistema com integridade conceitual é um sistema que tem funcionalidades que fazem sentido e atendem um propósito. Além disso, um sistema com integridade conceitual segue padronização não apenas nas funcionalidades mas também no seu código como padrões de nomeação de arquivos, funções e métodos.

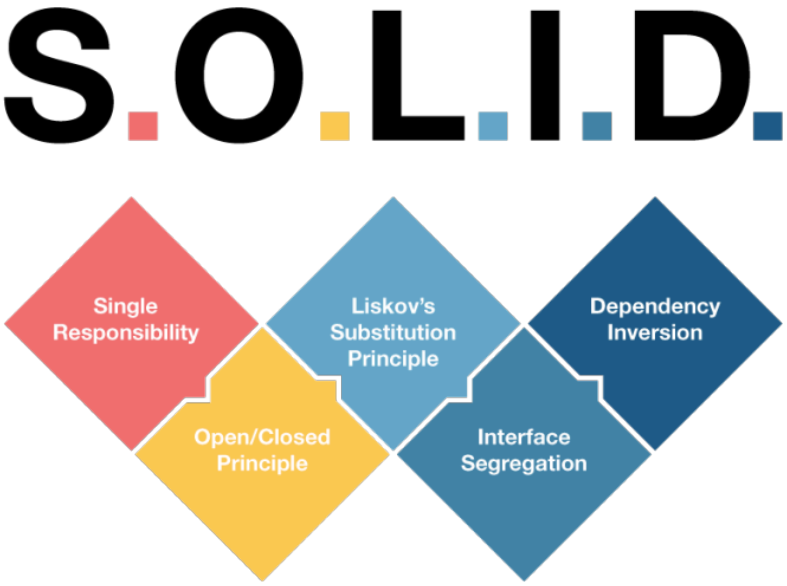
- **Ocultamento de informações:** Foi um tema abordado inicialmente por David Parner, onde é tratado da modularização do software fazendo com que tenha um ocultamento de informações importantes. Neste tópico é recomendado que as classes escondam detalhes de implementação suscetíveis a mudanças, expondo apenas o necessário por meio de interfaces estáveis. O autor faz um alerta importante sobre o uso de get e set nos códigos para assim não disponibilizar informações que não deveriam ser disponibilizadas.

- **Coesão:** Este tema entra no tema da modularidade, assim uma classe denominada coesa é uma classe que implementa uma única funcionalidade, sendo assim cada classe deve conter apenas uma única responsabilidade. A coesão entre classes facilita a implementação, entendimento e manutenção entre as classes.

- **Acoplamento:** O acoplamento é aceitável apenas quando uma classe A utiliza apenas métodos públicos de uma classe B, sem interferir nos seus métodos internos e no seu ocultamento de informações. Quando uma classe A compartilhar variável com a classe B, ou acessa informações que deveriam ser privadas podemos colocar que essas classes estão fortemente acopladas, logo o seu sistema não é coeso, não

tem um bom ocultamento de informações e nem uma integridade conceitual, colocando em ruína todos os outros pontos acima.

A partir dos conceitos elencados acima o livro elenca o princípio SOLID de projeto para ter uma melhor definição para projetar e desenvolver um software.



- **Princípio da Responsabilidade Única**: Cada classe deve ter uma única responsabilidade, colocando responsabilidade como uma única razão para ser modificada, promovendo alta coesão. O autor também coloca que uma das principais aplicações desse princípio é a separação entre classes de apresentação(frontend) e classes de regras de negócios(backend).

- **Princípio da Segregação de Interfaces**: Interfaces devem ser específicas para cada tipo de cliente, evitando que dependam de métodos que não utilizam.

- **Princípio de Inversão de Dependências**: Classes devem depender de abstrações (interfaces) e não de implementações concretas, tornando o sistema mais flexível a mudanças.

- **Prefira Composição a Herança**: A composição é recomendada sobre a herança, pois reduz o acoplamento e aumenta a flexibilidade do sistema.

- **Princípio de Demeter**: Também conhecido como "Lei do Menor Conhecimento", sugere que um objeto deve evitar um encapsulamento ao não depender tanto de outras classes. Sendo assim, a implementação de uma classe deve invocar apenas

métodos da sua própria classe, de objetos passados como parâmetro de objetos criados pelo próprio método e de atributos da classe do método.

- **Princípio Aberto/Fechado**: Classes devem estar abertas para extensão, mas fechadas para modificação, permitindo a adição de novas funcionalidades sem alterar o código existente. Esse princípio permite que as classes sejam flexíveis e capazes de se adaptarem a diversos cenários sem alterar seu código fonte.

- **Princípio de Substituição de Liskov**: Subclasses devem ser substituíveis por suas classes base sem alterar o comportamento esperado do sistema.

Por fim, o capítulo aborda a importância de métricas para avaliar a qualidade do projeto, como medidas de coesão, acoplamento e complexidade. Essas métricas fornecem uma base objetiva para identificar áreas que necessitam de melhorias e para monitorar a evolução da qualidade do software ao longo do tempo.

Em resumo, o capítulo 5 oferece uma visão abrangente e prática sobre os princípios de projeto de software, fornecendo fundamentos sólidos para desenvolvedores que buscam criar sistemas robustos, manuteníveis e alinhados com as melhores práticas da engenharia de software moderna.

Ao longo da leitura pude identificar vários temas abordados ao longo dos semestres passados, o princípio do SOLID foi bastante falado e utilizado durante a matéria de *programação modular,* lembro que colocamos em prática em um trabalho onde a ideia era sempre tentar abstrair ao máximo o projeto.