Capítulo 5 – Resumo

Um sistema de software é criado com o objetivo de resolver um problema específico. Para isso, é essencial decompor esse problema de forma estruturada, dividindo-o em **requisitos funcionais e não funcionais**. Essa abordagem permite definir claramente as funcionalidades que o sistema deve oferecer, bem como os aspectos técnicos e restrições que precisam ser atendidos. Além disso, é fundamental planejar a **quebra do problema** e sua posterior implementação no software, garantindo que a solução seja eficaz e eficiente.

Uma das técnicas mais utilizadas para facilitar esse processo é a **abstração**, que consiste em simplificar a representação de entidades do sistema, destacando apenas os aspectos essenciais e ocultando detalhes desnecessários. Essa estratégia contribui para a criação de sistemas mais organizados, modulares e fáceis de manter.

Princípios de Projeto

Os princípios de projeto estabelecem diretrizes fundamentais para garantir que um sistema atenda a determinadas propriedades de qualidade. Dentre os principais, destacam-se:

- **Integridade Conceitual**: Um sistema deve apresentar **coesão e coerência** entre seus artefatos, garantindo que todas as partes do software estejam alinhadas e que o usuário possa utilizá-lo de forma intuitiva e integrada.
- **Ocultamento de Informação**: Esse princípio determina que cada usuário deve visualizar apenas as informações necessárias para sua função, evitando a exposição de dados irrelevantes ou permissões excessivas. Essa prática melhora a segurança, facilita o desenvolvimento paralelo, proporciona maior flexibilidade para mudanças e torna o sistema mais compreensível. Um exemplo comum dessa abordagem é o uso de **getters e setters** para controlar o acesso aos atributos de uma classe.
- **Coesão**: Um software deve ser altamente coeso, o que significa que seus componentes devem ter **responsabilidades bem definidas e relacionadas entre si**. A separação de interesses dentro do código deve ser clara, tornando a estrutura do software mais organizada e fácil de entender.
- **Acoplamento**: O acoplamento se refere ao grau de dependência entre diferentes partes do sistema. Classes com um **acoplamento excessivo** tornam o código difícil de modificar e testar. Por isso, é importante manter um **nível adequado de acoplamento**, promovendo a reutilização e facilitando a manutenção do software.
- **Código Extensível e Modular**: Um bom projeto deve evitar **códigos demasiadamente extensos e complexos**, pois isso prejudica a legibilidade e a manutenção. A modularidade facilita a reutilização de componentes, reduzindo a duplicação de código e tornando o sistema mais sustentável ao longo do tempo.

Relacionando os Princípios de Projeto com SOLID

Os conceitos abordados nos princípios de projeto possuem uma forte relação com os princípios do SOLID, estudados na disciplina de **Programação Modular** no segundo semestre. Cada princípio SOLID complementa um ou mais princípios de projeto, conforme descrito a seguir:

- **Responsabilidade Única (SRP - Single Responsibility Principle) → Coesão**
    - Cada classe deve ter **uma única responsabilidade bem definida**, garantindo que sua função dentro do sistema seja clara e coesa.
- **Segregação de Interface (ISP - Interface Segregation Principle) → Coesão**
    - Interfaces podem ser organizadas de diferentes formas para melhorar a estrutura do código. Interfaces menores e mais específicas evitam dependências desnecessárias, tornando o código mais modular.
- **Inversão de Dependências (DIP - Dependency Inversion Principle) → Acoplamento**
    - Classes devem depender de **abstrações** e não de implementações concretas, reduzindo o acoplamento e tornando o sistema mais flexível para futuras mudanças.
- **Prefira Composição a Herança → Acoplamento**
    - A composição favorece um acoplamento mais fraco entre classes, enquanto a herança pode criar dependências rígidas que dificultam a manutenção do sistema.
- **Lei de Demeter (Principle of Least Knowledge) → Ocultamento de Informação**
    - Um módulo ou classe deve ter **o mínimo possível de conhecimento sobre outras partes do sistema**, evitando dependências excessivas e tornando o código mais encapsulado e seguro.
- **Aberto/Fechado (OCP - Open/Closed Principle) → Extensibilidade**
    - O código deve estar **aberto para extensão, mas fechado para modificação**, permitindo adicionar novas funcionalidades sem alterar partes já implementadas.
- **Substituição de Liskov (LSP - Liskov Substitution Principle) → Extensibilidade**
    - Regras devem ser seguidas ao redefinir métodos de classes base em classes filhas, garantindo que a substituição de um objeto de uma classe base por um de sua subclasse não quebre a lógica do sistema.

Esses princípios, quando aplicados corretamente, resultam em sistemas mais organizados, reutilizáveis, seguros e fáceis de manter.