O capítulo 5, *Princípios de Projeto,* começa introduzindo a definição de **projeto de software** destacando a importância da decomposição de problemas e a criação de abstrações para lidar com a complexidade dos sistemas modernos. Além disso, o capítulo busca explorar os princípios fundamentais do projeto de software, incluindo conceitos como integridade conceitual, ocultamento de informação, coesão e acoplamento. Também são abordados princípios essenciais como os do SOLID e as métricas de avaliação da qualidade do software.

Em seguida o capítulo aborda o estudo das propriedades de projetos de software, sendo elas, integridade conceitual, ocultamento de informação, coesão e acoplamento.

- **Integridade Conceitual:** Integridade conceitual é uma propriedade de projeto proposta por Frederick Brooks, na qual busca defende que um sistema não pode ser um amontoado de funcionalidades, sem coerência e coesão entre elas. Dessa forma, o capítulo busca averiguar a importância da integridade conceitual pois a mesma facilita o uso e entendimento de um sistema por parte de seus usuários. No entanto, esse capítulo fala como a integridade conceitual possui uma discussão sobre se o princípio requer que uma autoridade central, porém o projeto ser liderado por uma pessoa apenas não faz parte da definição dessa propriedade.
- **Ocultamento de Informação:** Introduzido por David Parnas, sugere que módulos e classes devem esconder detalhes internos de implementação para promover flexibilidade e facilitar a manutenção. Essa propriedade traz algumas vantagens para um sistema, tais quais: Desenvolvimento em paralelo, Flexibilidade a mudanças e Facilidade de entendimento. Contudo, esses benefícios só podem ser atingidos caso as classes escondam decisões de projeto que são sujeitas a mudanças.
- **Coesão:** Refere-se ao grau em que os elementos dentro de uma classe estão relacionados entre si, sendo desejável que cada classe tenha uma única responsabilidade no sistema.
    - **Separação de interesses** possui um conceito semelhante ao de coesão, sendo outra propriedade desejável em projetos de software. Ela defende que uma classe deve implementar apenas um interesse (qualquer funcionalidade, requisito ou responsabilidade) da classe.
- **Acoplamento:** Se refere ao nível (força) de dependência entre diferentes classes, podendo possuir dois tipos:

    - **Acoplamento aceitável**: Existe quando uma classe A usa apenas métodos públicos da classe B.
    - **Acoplamento ruim:** Existe quando uma classe A para uma classe B quando mudanças em B podem facilmente impactar A.

Na seção seguinte, o autor apresenta os princípios de projeto, incluindo os princípios SOLID, um conjunto de diretrizes criado por Robert Martin. Esses princípios incluem:

- **Responsabilidade Única:** Esse princípio fala que toda classe deve ter apenas uma responsabilidade.
- **Segregação de Interfaces:** O princípio define que interfaces têm que ser pequenas, coesas e, mais importante ainda, específicas para cada tipo de cliente.
- **Inversão de Dependências:** Esse princípio recomenda que uma classe cliente deve estabelecer dependências prioritariamente com abstrações e não com implementações concretas.
- **Prefira Composição a Herança:** Esse princípio recomenda que se existir duas soluções de projeto, uma baseada em herança e outra em composição, a solução por meio de composição, normalmente, é a melhor
- **Princípio de Demeter:** Defende que a implementação de um método deve invocar apenas os seguintes outros métodos:
    - de sua própria classe (caso 1)
    - de objetos passados como parâmetros (caso 2)
    - de objetos criados pelo próprio método (caso 3)
    - de atributos da classe do método (caso 4)
- **Aberto/Fechado:** Acredita que uma classe deve estar fechada para modificações e aberta para extensões.
- **Substituição de Liskov:** Explicita regras para redefinição de métodos de classes base em classes filhas. Também determina as condições — semânticas e não sintáticas — que as subclasses devem atender para que um programa como o anterior funcione.

Por fim, a última seção do capítulo trata discutir métricas de código fonte para avaliar a qualidade de projetos de software. O objetivo é permitir a avaliação da qualidade de um projeto de forma mais objetiva, utilizando propriedades como tamanho, coesão, acoplamento e a complexidade do código. No entanto essa monitoração através de métricas não é algo tão comum nos dias de hoje pois algumas das propriedades citadas acima são consideradas subjetivas.

- **Tamanho:** A métrica de código fonte mais popular é linhas de código (LOC, *lines of code*), utilizada para medir o tamanho de uma função, classe, pacote ou de um sistema inteiro.

- **Coesão:** Uma das métricas mais conhecidas para se calcular coesão é chamada de LCOM (*Lack of Cohesion Between Methods*), responsável por medir "a falta de coesão" de uma classe.
- **Acoplamento:** CBO (*Coupling Between Objects*) é uma métrica para medir acoplamento estrutural entre duas classes.
- **Complexidade:** É uma métrica proposta para medir a complexidade do código de uma função ou método.

Alguns exemplos de casos para aplicar no mercado de acordo com o capítulo 5 do livro, *Engenharia de Software Moderna,* seriam as plataformas de streaming, como a Netflix e o Spotify, pois **t**odas as funcionalidades relacionadas à reprodução de vídeos e/ou músicas devem seguir regras consistentes, aplicando a integridade conceitual.