Aluno: Kayque Gabriel Ruiz RA: 22014348-2

RESUMO: COMO CRIAR E DESACOPLAR CAMADAS USANDO SOLID

Em resumo, o vídeo destaca a importância dos princípios SOLID na criação de software modular, desacoplado e de fácil manutenção, e como esses princípios podem ser aplicados para criar e desacoplar camadas em um sistema. Abaixo está um resumo dos pontos abordados:

1. Introdução ao SOLID: SOLID é uma abreviatura de cinco princípios de design de software, esses princípios são: Princípio de Responsabilidade Única, Princípio Aberto e Fechado, Princípio de Substituição de Liskov, Princípio de Isolamento de Interface e Princípio de Inversão de Dependência.

2. Princípio da Responsabilidade Única (SRP): Este princípio enfatiza que uma classe deve ter apenas um motivo para mudança, ou seja, deve ter apenas uma responsabilidade. Isso promove a coesão e torna o código mais fácil de manter.

3. Princípio Aberto/Fechado (OCP): O OCP recomenda que as classes sejam abertas para extensão e fechadas para modificação. Isto é conseguido através do uso de abstrações e interfaces, permitindo que novos comportamentos sejam adicionados sem modificar o código existente.

4. Princípio de Substituição de Liskov (LSP): Este princípio afirma que um objeto deve ser substituído por uma instância de sua subclasse sem alterar a integridade do programa. Isso garante que uma classe derivada possa ser usada no lugar de sua classe base sem introduzir erros.

5. Princípio de Segregação de Interface (ISP): O ISP propõe que as interfaces sejam específicas para os clientes que as utilizam e evitem interfaces genéricas e excessivamente grandes. Isso evita que as classes dependam de funcionalidades desnecessárias.

6. Princípio de Inversão de Dependência (DIP): O DIP sugere que as dependências devem estar em abstrações, não em implementações. Isso permite que as dependências de classe sejam abstraídas, tornando o código mais flexível e fácil de testar.

7. Desacoplamento de Camadas: Utilizando os princípios do SOLID, é possível desacoplar as camadas de um sistema, tornando cada camada independente e mais fácil de manter. Isso é alcançado através da criação de interfaces que definem contratos entre as camadas e da utilização de injeção de dependência para fornecer implementações concretas.