



From STUPID to SOLID



1 Princípios STUPID – Devem ser evitados

Os princípios STUPID resultam em código difícil de manter, testar e evoluir.

 Princípio	 Problema
Singleton	Padrão que garante que apenas uma instância de uma classe exista. O uso excessivo pode criar estado global, dificultando os testes e acoplando o código.
Tight Coupling (Acoplamento Forte)	Módulos altamente dependentes entre si dificultam manutenção e reutilização. Classes fortemente acopladas dependem muito umas das outras.
Untestability (Incapacidade de Testar)	Código mal projetado dificulta a criação de testes.
Premature Optimization (Otimização Prematura)	Otimizar cedo demais pode tornar o código complexo sem necessidade.
Indiscriptive Naming (Nomes pouco descritivos)	Nomes maus dificultam a leitura e manutenção do código.
Duplication (Duplicação de código)	Código repetido deve ser evitado com o princípio DRY (<i>Don't Repeat Yourself</i>).

2 Princípios SOLID (Boas práticas)

Os princípios SOLID promovem código mais **modular, reutilizável e fácil de manter**.

 Princípio	 Definição
Single Responsibility Principle (SRP)	Uma classe deve ter apenas uma responsabilidade .
Open/Closed Principle (OCP)	O código deve estar aberto para extensão , mas fechado para modificação . Ou seja, criar código flexível, que aceita novas funcionalidades sem precisar modificar o que já funciona.
Liskov Substitution	Substituir uma classe base por uma derivada não deve partir o sistema .

Principle (LSP)	
Interface Segregation Principle (ISP)	Uma classe não deve ser obrigada a implementar coisas que não usa. Ou seja, não obrigar uma classe a usar métodos desnecessários. Ex: Se um restaurante só vende pizza ele não deveria ser obrigado a implementar métodos como "fazer sushi".
Dependency Inversion Principle (DIP)	Depender de abstrações , e não de implementações concretas. Isto é, ao invés de classes dependerem diretamente umas das outras, elas devem depender de abstrações.