Resumo sobre o vídeo: <https://www.youtube.com/watch?v=6SfrO3D4dHM&feature=youtu.be>

**O que é o S.O.L.I.D?**

O S.O.L.I.D é um acrônimo que representa cinco princípios da programação orientada a objetos e design de código teorizados pelo nosso querido **Uncle Bob** (Robert C. Martin) por volta do ano 2000. O autor Michael Feathers foi responsável pela criação do acrônimo:

1. **[S]**ingle Responsibility Principle (Princípio da Responsabilidade Única)

Uma classe, componente, entidade ou até função deve ter uma única responsabilidade. Para cada responsabilidade deve ter-se uma entidade específica, única e melhor para realizar aquilo. Isso possibilita reaproveitamento de código, refatorar, realizar testes, gerar menos bugs (e se acontecer de ter bugs, será mais fácil contornar e resolver o bug). Desta maneira o programador vai de escopo fechado em escopo fechado prestando atenção mais focada e de maneira mais organizada.

Um exemplo prático para entender onde começa e termina a responsabilidade de uma entidade é lavar uma meia vermelha numa maquinada de roupas claras – o que pode causar a mancha nas demais roupas, caso sejam misturadas à meia vermelha. Isso denomina um software ACOPLADO. Uma dica de melhor programação, neste caso, é colocar no nome da função exatamente tudo que ela pode fazer.

1. **[O]**pen/Closed Principle (Princípio do Aberto/Fechado)

Classes, entidades ou funções devem estar abertas para EXTENSÃO de novas funcionalidades, mas fechadas para MODIFICAÇÃO. Assim temos a seguinte situação: quando implementada uma função com seus métodos e, depois deum tempo, aparecem novos requisitos. Isso causa uma alteração no código do método, exemplo, if dentro de if para especificar cada situação nova dos novos requisitos. Isso deve ser mudado, separando cada coisa para um método ou classe diferente.

Resumindo: não modifique as entidades ou funções base, crie uma implementação separada para validar e processar esse nível de abstração correto.

1. **[L]**iskov Substitution Principle (Princípio da Substituição de Liskov)

Se tivermos uma classe e criarmos uma subclasse utilizando herança, o objeto resultante da herança DEVE conseguir substituir o objeto principal sem quebrar o programa. Respeitar esse princípio força o programa a ter as abstrações no nível certo e ser mais consistente. Um método criado no nível mais alto não deve gerar exceção quando executado em níveis mais baixos.

1. **[I]**nterface Segregation Principle (Princípio da Segregação de Interfaces)

Uma classe não deve ser forçada a depender de métodos/interface que ela no usa. Existem classes que não possuem os atributos ou requisitos que utilizam aqueles métodos, desta forma, evita-se a implementação oca de código.

1. **[D]**ependency Inversion Principle (Princípio da Inversão de Dependências)

Um módulo não deve depender de detalhes de implementação de outro módulo diretamente. Deve haver uma abstração, uma interface ali neste meio, e o método que se encaixar nessa interface intermediária conseguirá ser executada.