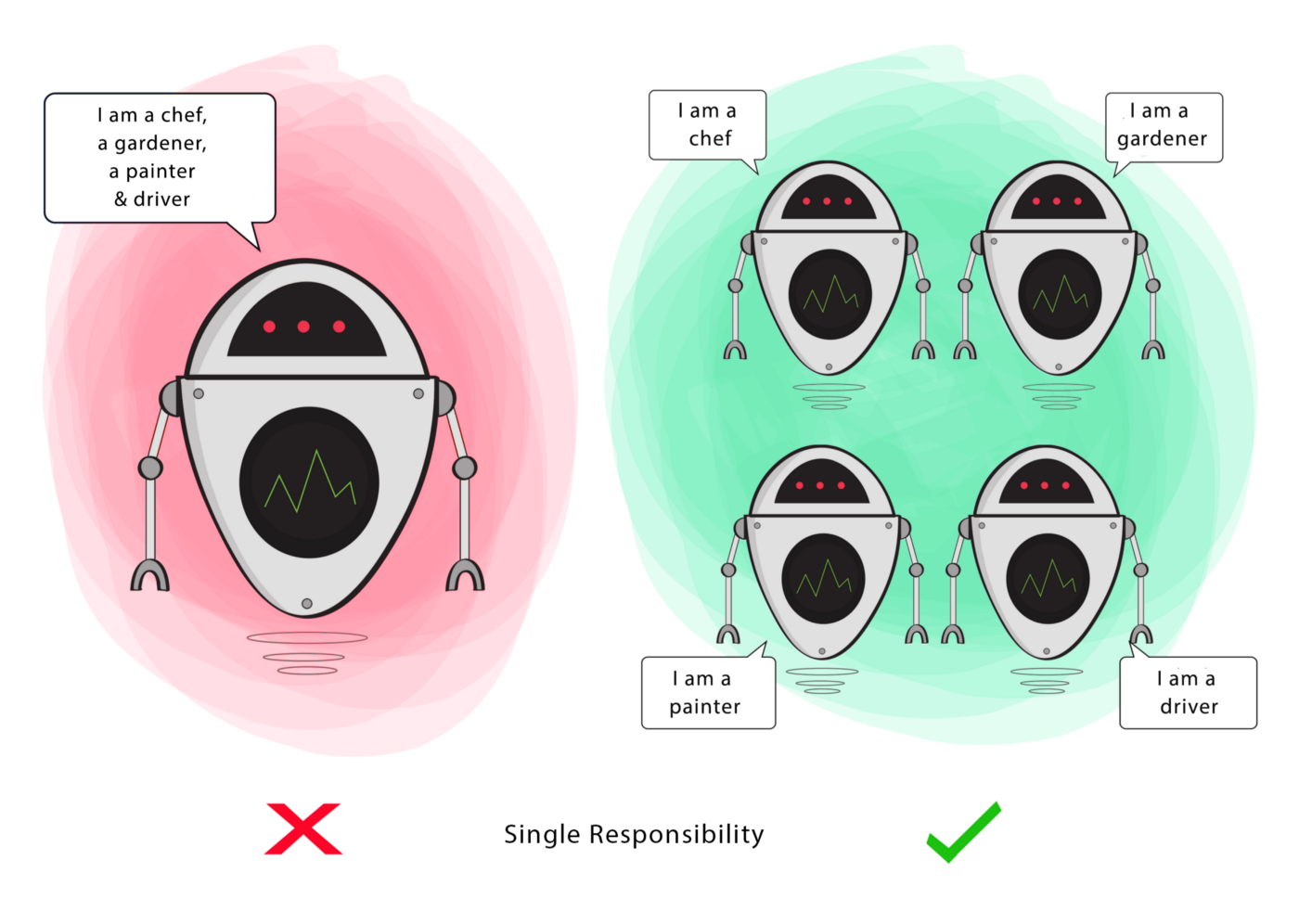
**S.O.L.I.D**

**S – Single-responsability principal**

*Princípio da Responsabilidade Única.*

Este princípio diz que: uma Classe, um Componente, uma Entidade ou até mesmo uma Função deve ter uma única responsabilidade.



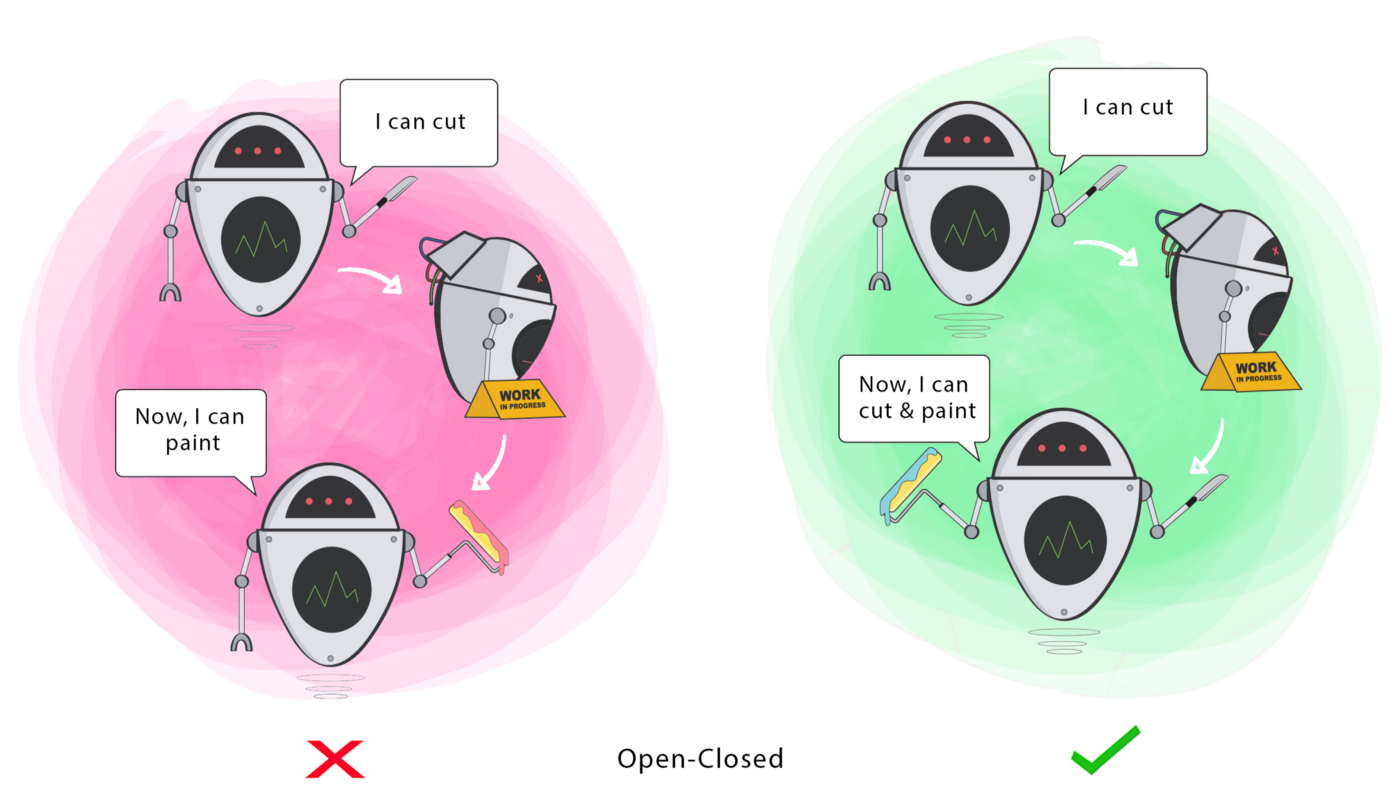
Porque usar esse princípio:

* Reaproveitamento de código;
* Refatoração de código;
* Testes automatizados;
* Gerar menos bugs;
* Resolução de bugs.

**O – Open-closed principal**

*Princípio de Aberto - Fechado.*

*Esse princípio diz que: Classes, Entidades ou Funções devem estar abertas para* ***Extensão****, mas fechadas para* ***Modificação****.*



Mudar o comportamento de uma Classe irá afetar todo o sistema utilizando aquela Classe. Se você quer realizar mais funções, a abordagem ideal é adicionar às funções que já existentes, NÃO as alterar.

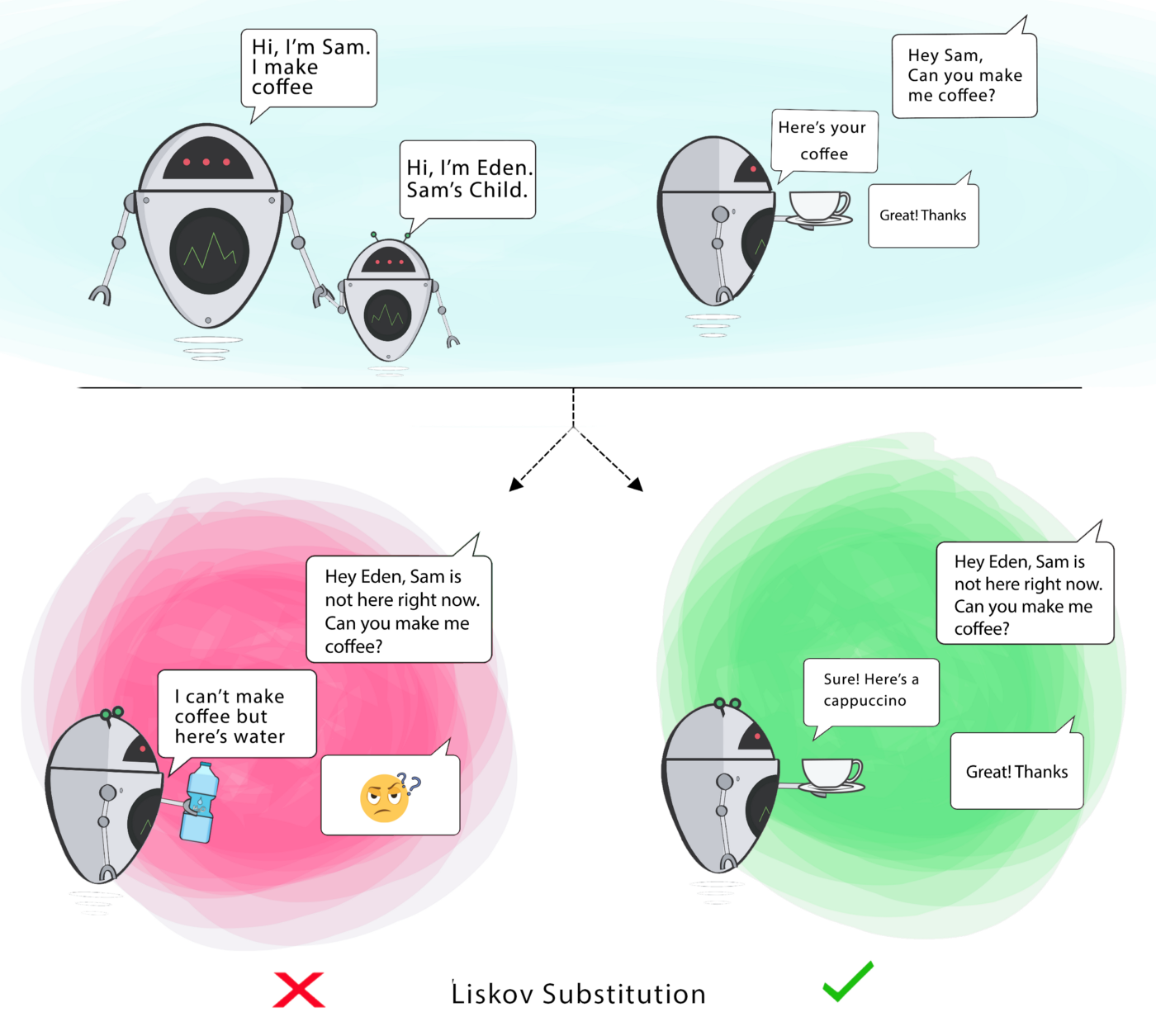
Objetivo: Esse princípio visa entender o comportamento da Classe sem mudar o comportamento existente daquela Classe. Isso evita os bugs ondes quer que a Classe seja usada.

**L – Liskov substitution principle**

*Princípio da substituição Liskov.*

Esse princípio diz que: Se “S” é um subtipo de “T”, então os objetos do tipo “T” no programa devem conseguir substituir objetos do tipo “S” sem alterar nenhuma das propriedades desejáveis daquele programa.

Não faça subclasses terem o comportamento não esperado, no sentido de que, um método fundamental existir numa altura da árvore de herança, e depois, esse mesmo método gerar uma exceção em uma altura mais abaixo.



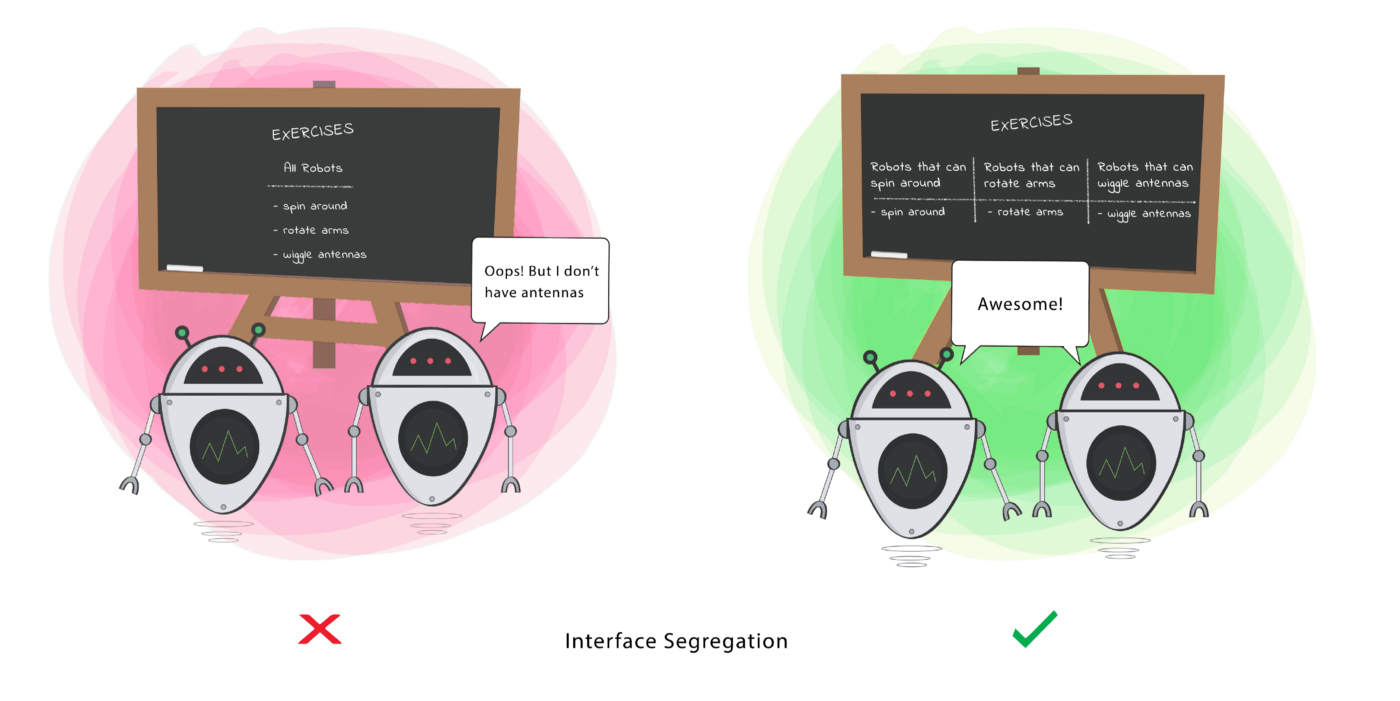
Objetivo: Esse princípio visa reforçar a consistência, assim a classe Pai ou a classe Filho, possam ser usadas da mesma maneira serem que haja erros.

**I – Interface segregation principle**

*Princípio da interface de segregação.*

Esse princípio diz que: Clientes (Classe) não devem ser forçados a depender de métodos que eles não usam.

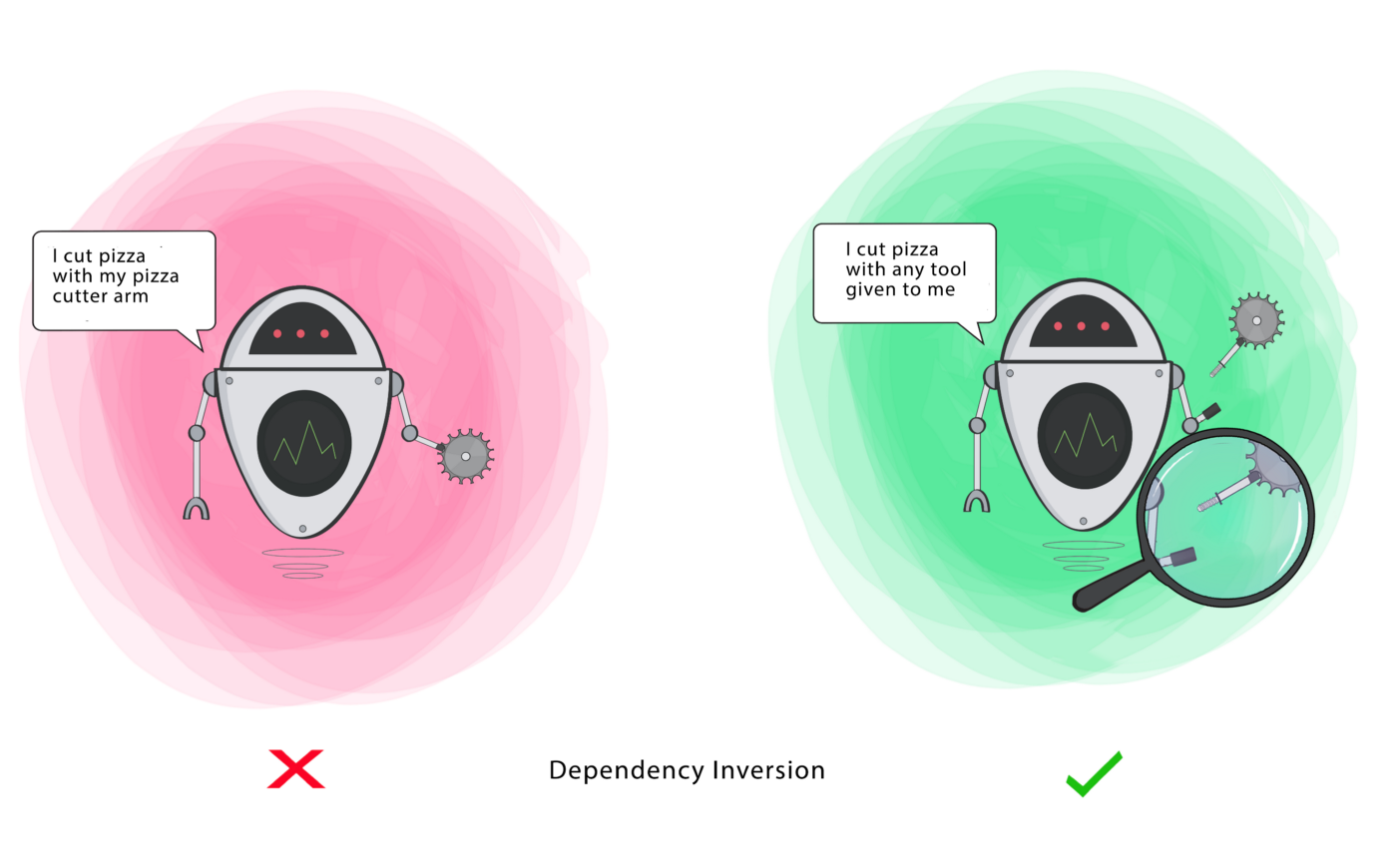
Por exemplo: uma Classe que é forçada a implementar uma interface, com métodos que ela não irá utilizar.



**D – Dependency inversion principle**

*Princípio da inversão de dependência.*

Esse princípio diz que: Um módulo não deve depender de detalhes de implementação de outro módulo diretamente, deverá existir uma interface ali no meio.



Objetivo: Esse princípio visa reduzir a dependência da high-level Class on the low-level Class introduzindo uma interface.

High-level Module(or Class): Class that executes an action with a tool.

Low-level Module (or Class): The tool that is needed to execute the action

Abstraction: Represents an interface that connects the two Classes.

Details: How the tool works