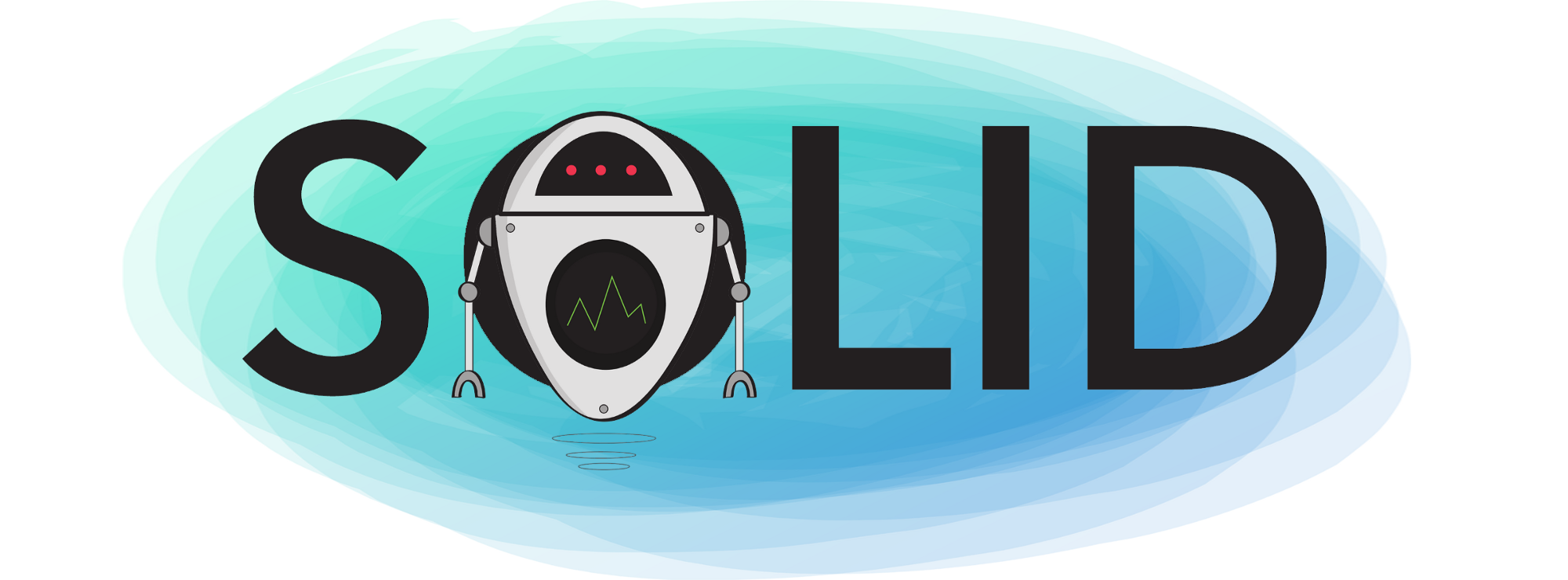
**The Understanding Of The SOLID Principles**



**Introduction:-**

* **SOLID** principles come into picture because these are the design principles that help us in encouraging creating more maintainable, understandable, and flexible software.
* **As our applications grow in size, we can reduce their complexity** by using **SOLID** principles.
* **SOLID** Principles:-

The following five concepts make up our SOLID principles:

1. **S**ingle Responsibility
2. **O**pen/Closed
3. **L**iskov Substitution
4. **I**nterface Segregation
5. **D**ependency Inversion

* These five software development principles are guidelines to follow when building software so that it is easier to scale and maintain. They were made popular by a software engineer, [Robert C. Martin](https://en.wikipedia.org/wiki/Robert_C._Martin).

**BENEFITS OF SOLID PRINCIPLES-**

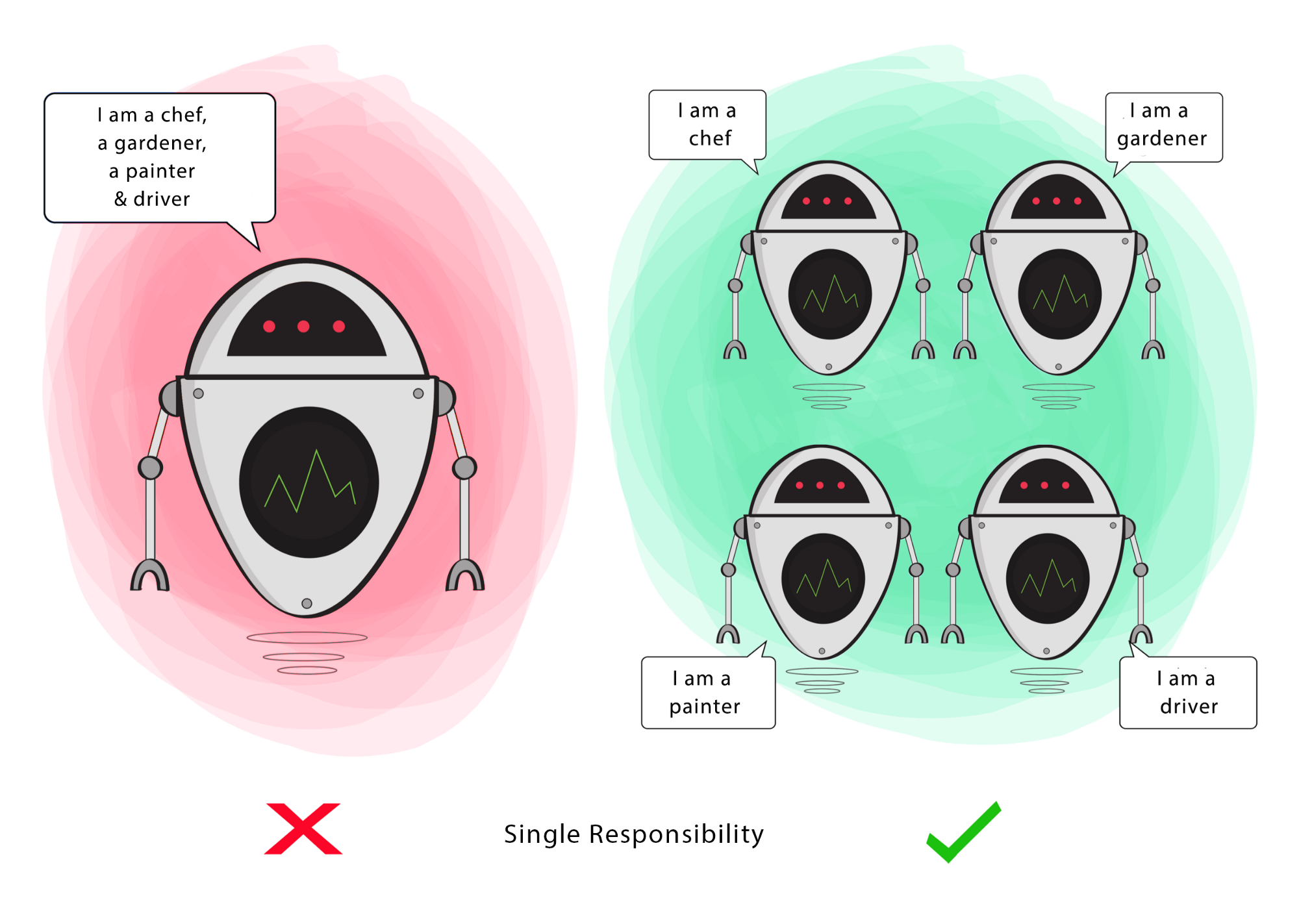
Some of the benefits SOLID Principle holds are as follows:-

* Loose Coupling
* Code Maintainability
* Dependency Management

# **The SOLID Principles:-**

## **S — Single Responsibility:**

A class should have a single responsibility



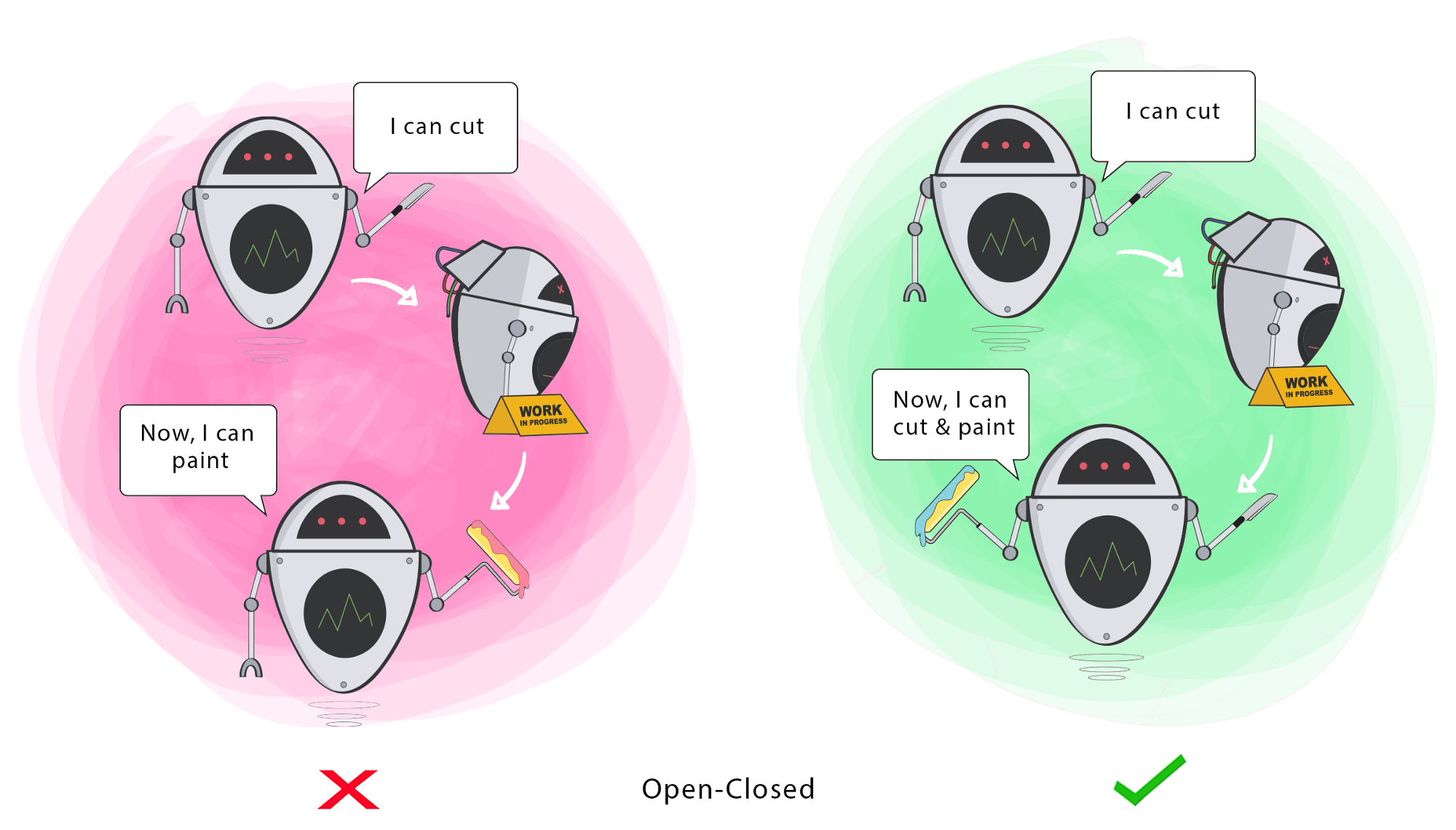
* If a Class has many responsibilities, it increases the possibility of bugs because making changes to one of its responsibilities could affect the other ones without you knowing.

**Goal:**

**This principle aims to separate behaviours so that if bugs arise as a result of your change, it won’t affect other unrelated behaviours.**

**2. O — Open-Closed:**

**Classes should be open for extension, but closed for modification**



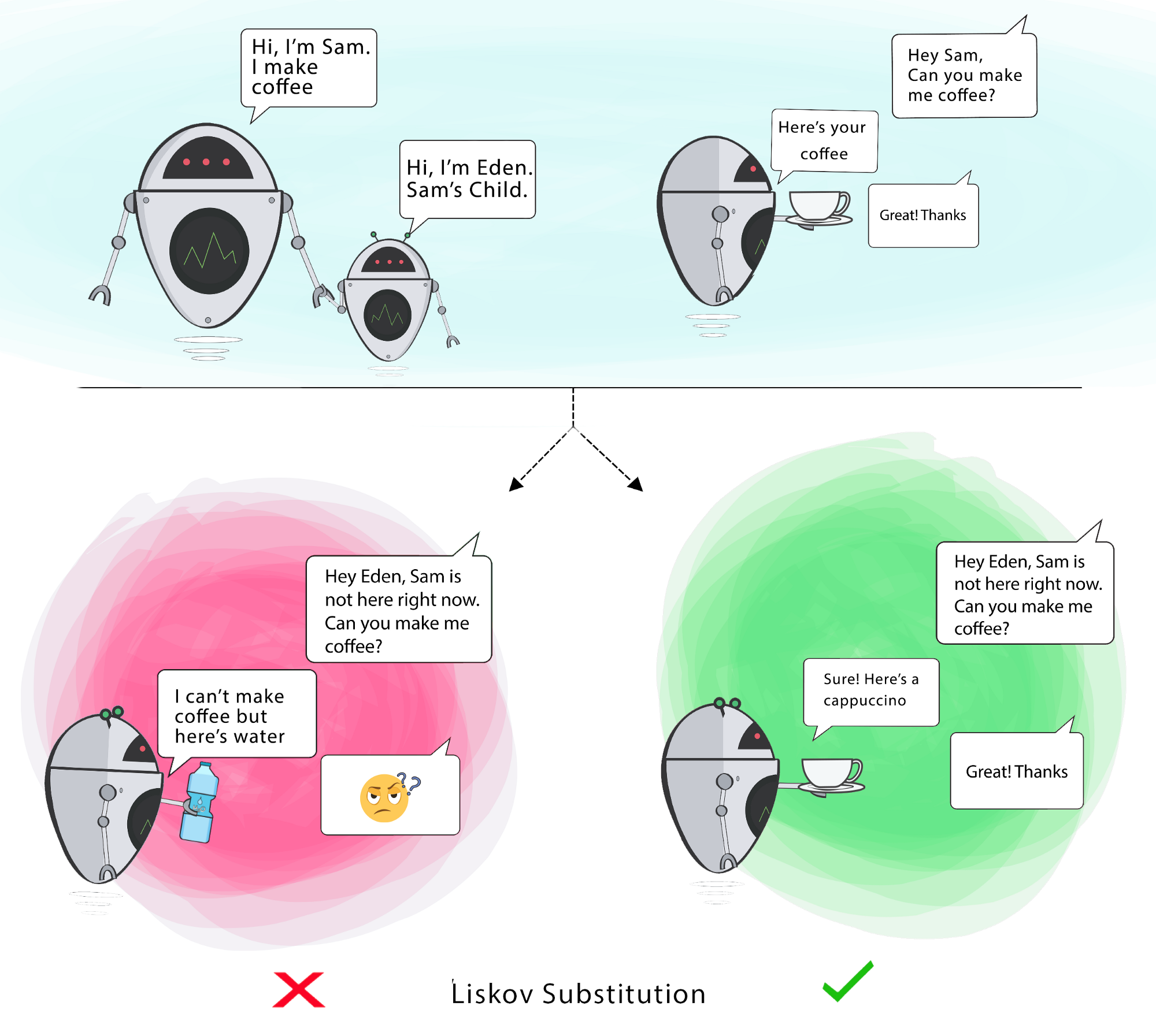
* Changing the current behaviour of a Class will affect all the systems using that Class.
* If you want the Class to perform more functions, the ideal approach is to add to the functions that already exist NOT change them.

**Goal:**

This principle aims to extend a Class’s behaviour without changing the existing behaviour of that Class. This is to avoid causing bugs wherever the Class is being used.

**3. L** — **Liskov Substitution:**

**If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program.**

****

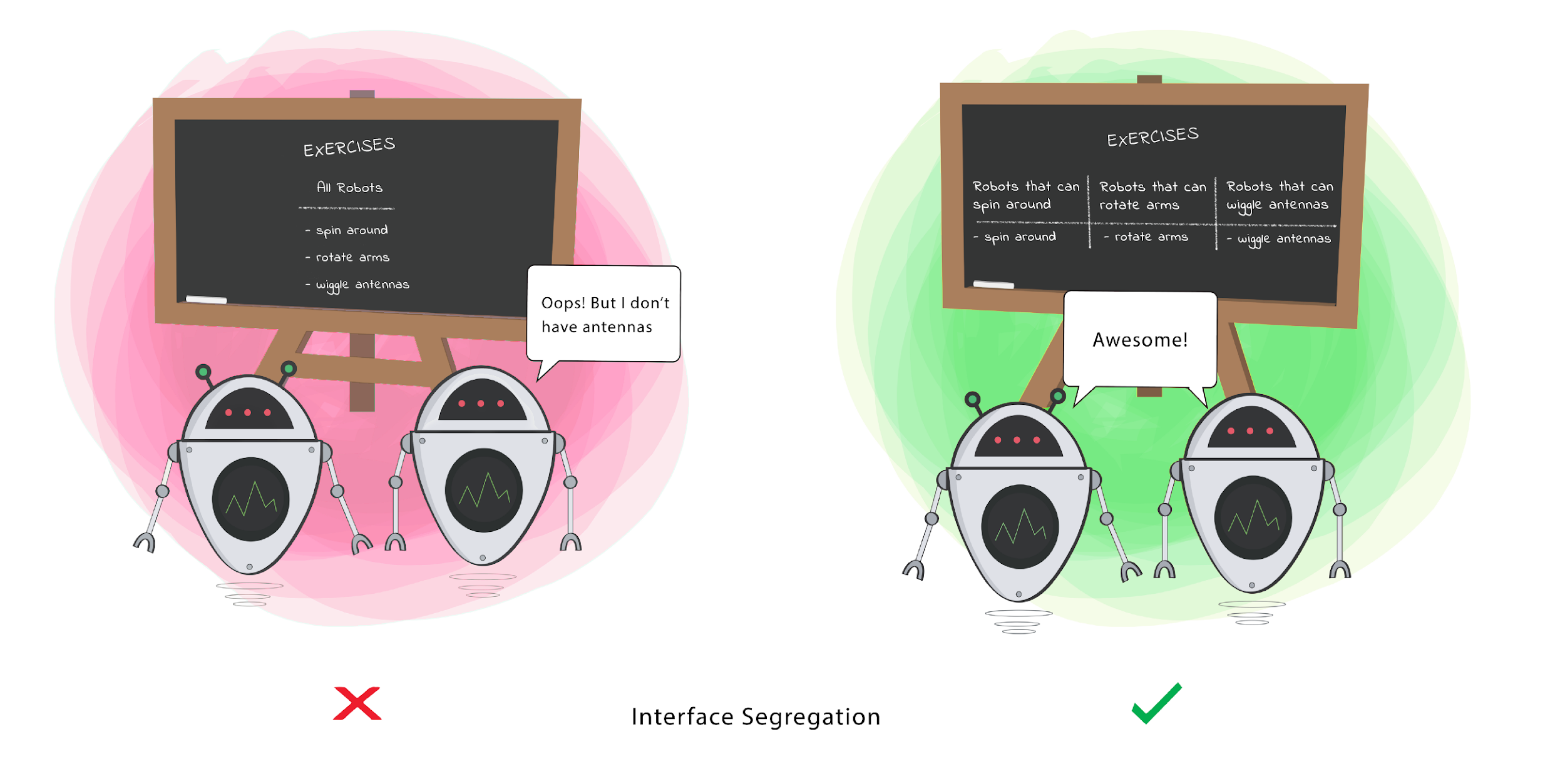
* When a child Class cannot perform the same actions as its parent Class, this can cause bugs.
* If you have a Class and create another Class from it, it becomes a parent and the new Class becomes a child. The child Class should be able to do everything the parent Class can do. This process is called Inheritance.
* The child Class should be able to process the same requests and deliver the same result as the parent Class or it could deliver a result that is of the same type.
* The picture shows that the parent Class delivers Coffee(it could be any type of coffee). It is acceptable for the child Class to deliver Cappuccino because it is a specific type of Coffee, but it is NOT acceptable to deliver Water.
* If the child Class doesn’t meet these requirements, it means the child Class is changed completely and violates this principle.

**Goal:**

This principle aims to enforce consistency so that the parent Class or its child Class can be used in the same way without any errors.

**4. I** — **Interface Segregation:**

**Clients should not be forced to depend on methods that they do not use.**

****

* When a Class is required to perform actions that are not useful, it is wasteful and may produce unexpected bugs if the Class does not have the ability to perform those actions.
* A Class should perform only actions that are needed to fulfil its role. Any other action should be removed completely or moved somewhere else if it might be used by another Class in the future.

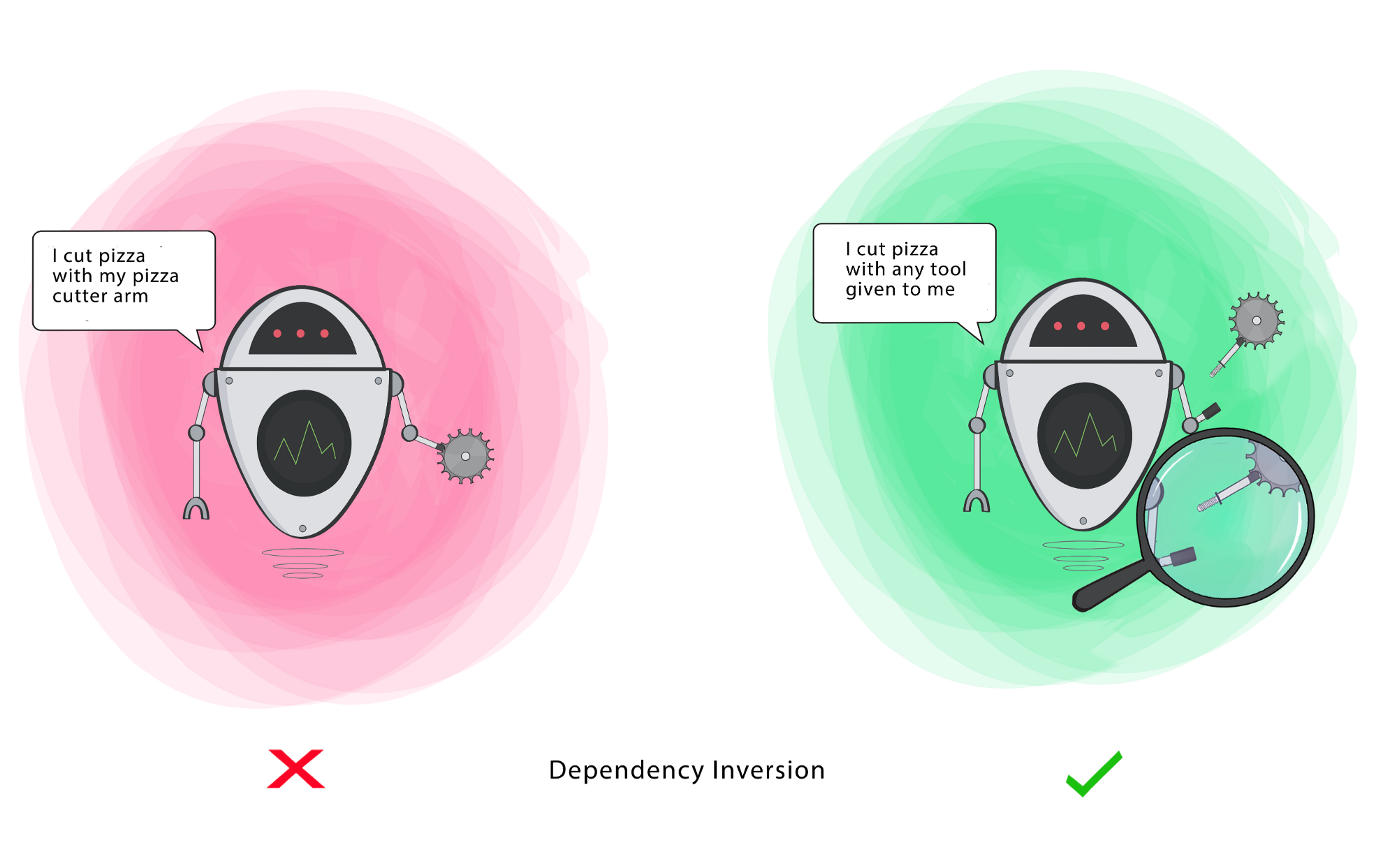
**Goals:**

This principle aims at splitting a set of actions into smaller sets so that a Class executes ONLY the set of actions it requires.

**5. D** — **Dependency Inversion:**

**- High-level modules should not depend on low-level modules. Both should depend on the abstraction.**

**- Abstractions should not depend on details. Details should depend on abstractions.**

****

* This principle says a Class should not be fused with the tool it uses to execute an action. Rather, it should be fused to the interface that will allow the tool to connect to the Class.
* It also says that both the Class and the interface should not know how the tool works. However, the tool needs to meet the specification of the interface.

**Goal:**

This principle aims at reducing the dependency of a high-level Class on the low-level Class by introducing an interface.

# **Summary:**

So far, we have discussed these five principles and highlighted their goals. They are to help you make your code easy to adjust, extend and test with little to no problems.