



Introduction to SOLID Design Principles

Get introduced to the SOLID design principles.

We'll cover the following

- Introduction
- Why use SOLID principles?
- Design principles

Introduction

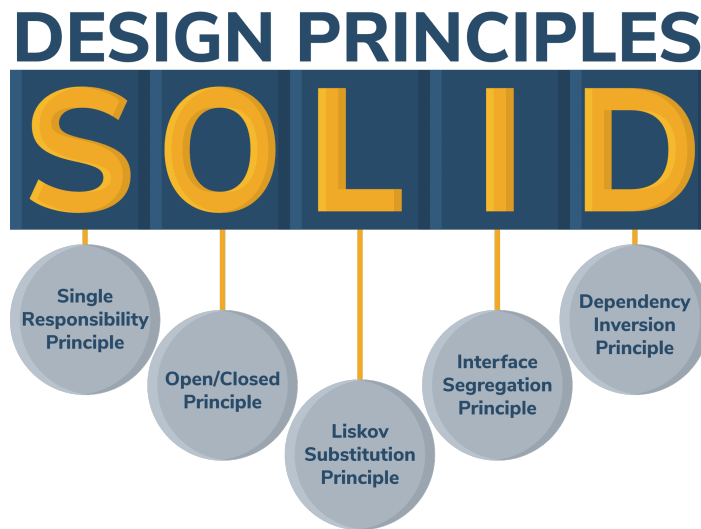
When creating software, we can follow good practices to avoid issues to make our code easier to understand, robust, and maintainable. Few of these practices are often termed as principles, e.g., the **SOLID** principles refer to the best practices to be followed in OOD.

SOLID is an acronym for the first five object-oriented design (OOD) principles by *Robert C. Martin*, also known as *Uncle Bob*, the author of *Clean Code: A Handbook of Agile Software Craftsmanship*.

The illustration below represents the acronym for S

Got any feedback? Get in touch with us.





The SOLID design principle in object-oriented design

Why use SOLID principles?

Let's look at the possible issues below that may occur in the code if we don't adhere to the SOLID principles.

- The code may become tightly coupled with several components, which makes it difficult to integrate new features or bug fixes and sometimes leads to unidentified problems.
- The code will be untestable, which effectively means that every change will need end-to-end testing.
- The code may have a lot of duplication.
- Fixing one issue results in additional errors.

However, if we adhere to the SOLID principles, we are able to do the following:

Got any feedback? Get in touch with us.

- Reduce the tight coupling of the code, which reduces errors.
- Reduce the code's complexity for future use.
- Produce more extensible, maintainable, and understandable software code.
- Produce the code that is modular, feature specific, and is extremely testable.


?

T

☾

Design principles

Let's look at the definition of the five design principles.

- 
- In the **Single Responsibility Principle (SRP)**, each class should be responsible for a single part or functionality of the system.
 - In the **Open Closed principle (OCP)**, software components should be open for extension but closed for modification.
 - In the **Liskov Substitution Principle (LSP)**, objects of a superclass should be replaceable with objects of its subclasses without breaking the system.
 - The **Interface Segregation Principle (ISP)** makes fine-grained interfaces that are client specific.
 - The **Dependency Inversion Principle (DIP)**, ensures that the high-level modules are not dependent on low-level modules. In other words, one should depend upon abstraction and not concretion.

In the next few lessons, we will explain these five design principles in detail.

[← Back lesson](#)[✓ Completed](#)[Next →](#)

Got any feedback? Get in touch with us.

