Solid principles

The SOLID principles are a set of five design principles that help guide software developers in writing well-structured, maintainable, and scalable object-oriented code. These principles were introduced by Robert C. Martin and are widely recognized and applied in software engineering. Each principle addresses a specific aspect of software design, aiming to promote modularity, flexibility, and ease of maintenance. Here's a brief overview of each SOLID principle:

Single Responsibility Principle (SRP):

This principle states that a class should have only one reason to change, meaning it should have a single responsibility or purpose. Separating concerns into distinct classes or modules enhances maintainability and makes the code easier to understand and modify.

Open/Closed Principle (OCP):

The Open/Closed Principle emphasizes that software entities (classes, modules, functions) should be open for extension but closed for modification. In other words, you should be able to add new functionality without altering existing code. This is achieved through interfaces, abstract classes, and design patterns like the Strategy pattern.

Liskov Substitution Principle (LSP):

The Liskov Substitution Principle states that objects of a derived class should be substitutable for objects of the base class without affecting the correctness of the program. In simpler terms, subclasses should adhere to the behavior of their parent classes, ensuring that polymorphism works as expected.

Interface Segregation Principle (ISP):

The Interface Segregation Principle suggests that clients should not be forced to depend on interfaces they do not use. In other words, it's better to have smaller, specialized interfaces rather than large, general-purpose ones. This avoids forcing classes to implement methods they don't need.

Dependency Inversion Principle (DIP):

The Dependency Inversion Principle encourages high-level modules to depend on abstractions (interfaces) rather than concrete implementations. It also suggests that low-level modules should depend on the same abstractions. This promotes loose coupling, flexibility, and easier testing.