OOP HW18 Software Engineering Principles

DATE: 01th february 2021

The 5 Principles That Will Help You Develop Quality Software

If we talk about application design and development, SOLID principles are words you should know as one of the fundamentals of software architecture and development.

The SOLID Principles

The 5 SOLID principles of software application design are:

- S - Single Responsibility Principle (SRP)

- O - Open/Closed Principle (OCP)

- L - Liskov Substitution Principle (LSP)

- I - Interface Segregation Principle (ISP)

- D - Dependency Inversion Principle (DIP)

Among the objectives of taking these 5 principles into account when writing code we find:

- To create efficient software: that fulfills its purpose and that is robust and stable.

- To write a clean and flexible code in the face of changes: that can be easily modified as needed, that is reusable and maintainable.

- Allow scalability: that accepts to be extended with new functionalities in an agile way.

In short, to develop quality software.

In this sense, the application of SOLID principles is closely related to the understanding and use of design patterns, which will allow us to maintain a high cohesion and, therefore, a low software coupling.

What are cohesion and coupling?

They are two very relevant concepts when designing and developing software. Let's see what they consist of.

Coupling

Coupling refers to the degree of interdependence that two software units have on each other, meaning by software units: classes, subtypes, methods, modules, functions, libraries, etc.

If two software units are completely independent of each other, we say that they are decoupled.

Cohesion

Software cohesion is the degree to which different elements of a system stay together to achieve a better result than if they worked separately. It refers to the way in which we can group different software units together to create a larger unit.

Principle of Single Responsibility

"A class should have one, and only one, reason to change."

The S in the acronym we are talking about today refers to Single Responsibility Principle (SRP). According to this principle "a class should have one, and only one, reason to change." It is this, precisely, "reason to change," that Robert C. Martin identifies as "responsibility."

The Single Responsibility principle is the most important and fundamental principle of SOLID, very simple to explain, but the most difficult to follow in practice.

Bob himself summarizes how to do it: "Gather together the things that change for the same reasons. Separate those things that change for different reasons", i.e.: "Gather together the things that change for the same reasons. Separate those things that change for different reasons".

2. Open/Closed Principle

"You should be able to extend a classes behavior, without modifying it."

The second SOLID principle was formulated by Bertrand Meyer in 1988 in his book "Object Oriented Software Construction" and says: "You should be able to extend a classes behavior, without modifying it". In other words: the classes you use should be open to be extended and closed to be modified.

In his blog Robert C. Martin defended this principle that a priori may seem a paradox. It is important to take into account the Open/Closed Principle (OCP) when developing classes, libraries or frameworks.

3. Liskov's Substitution Principle

"Derived classes must be substitutable for their base classes."

The L in SOLID refers to the surname of its creator, Barbara Liskov, and states that "derived classes must be substitutable for their base classes".

This means that objects must be able to be replaced by instances of their subtypes without altering the correct functioning of the system or, in other words: if we use a certain class in a program, we should be able to use any of its subclasses without interfering with the functionality of the program.

According to Robert C. Martin, violating the Liskov Substitution Principle (LSP) also implies violating the Open/Closed principle.

4. Interface Segregation Principle.

"Make fine grained interfaces that are client specific."

In the fourth SOLID principle, Uncle Bob suggests, "Make interfaces that are client specific," i.e., for a specific purpose.

In this sense, according to the Interface Segregation Principle (ISP), it is preferable to have many interfaces that define few methods than to have an interface forced to implement many methods that it will not use.

5. Dependency Inversion Principle

"Depend on abstractions, not on concretions."

We come to the last principle: "Depend on abstractions, not on concretions classes".

Thus, Robert C. Martin recommends:

1. High-level modules should not depend on low-level modules. Both should depend on abstractions.
2. Abstractions should not depend on details. Details should depend on abstractions.

The objective of the Dependency Inversion Principle (DIP) is to reduce dependencies between code modules, i.e., to achieve low coupling of classes.