UNIVERSIDAD BOLIVIANA DE INFORMATICA

INGENIERÍA DE SISTEMAS

INVESTIGACIÓN SOLID

Asignatura:

Taller de Sistemas

Docente:

Ing. Daniel Alejandro Coronel Berrios

Estudiante:

Manuel Alejandro Choque Sanjines

La Paz – Bolivia

2022

SOLID

Los **5 principios SOLID** de diseño de aplicaciones de software son:

- **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)

Entre los objetivos de tener en cuenta estos 5 principios a la hora de escribir código encontramos:

- Crear un **software eficaz**: que cumpla con su cometido y que sea **robusto y estable**.
- Escribir un **código limpio y flexible** ante los cambios: que se pueda modificar fácilmente según necesidad, que sea **reutilizable** y **mantenible**.
- Permitir **escalabilidad**: que acepte ser ampliado con nuevas funcionalidades de manera ágil.

En definitiva, desarrollar un **software de calidad**.
En este sentido la aplicación de los principios SOLID está muy relacionada con la comprensión y el uso de **patrones de diseño**, que nos permitirán mantener una **alta cohesión** y, por tanto, un **bajo acoplamiento** de software.

¿Qué son la cohesión y el acoplamiento?

Son dos conceptos muy relevantes a la hora de diseñar y desarrollar software. Veamos en qué consisten.

Acoplamiento

El acoplamiento se refiere al **grado de interdependencia que tienen dos unidades de software entre sí**, entendiendo por unidades de software: clases, subtipos, métodos, módulos, funciones, bibliotecas, etc.
Si dos unidades de software son completamente independientes la una de la otra, decimos que están desacopladas.

Cohesión

La cohesión de software es el **grado en que elementos diferentes de un sistema permanecen unidos para alcanzar un mejor resultado** que si trabajaran por separado. Se refiere a la forma en que podemos agrupar diversas unidades de software para crear una unidad mayor.

1. Principio de Responsabilidad Única

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

La S del acrónimo del que hablamos hoy se refiere a **Single Responsibility Principle (SRP)**. Según este principio "una clase debería tener **una, y solo una, razón para cambiar**". Es esto, precisamente, "razón para cambiar", lo que Robert C. Martin identifica como "responsabilidad".
El principio de Responsabilidad Única es **el más importante y fundamental de SOLID**, muy sencillo de explicar, pero el más difícil de seguir en la práctica.
El propio Bob resume cómo hacerlo: *"Gather together the things that change for the same reasons. Separate those things that change for different reasons"*, es decir: "Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian por razones diferentes".

2. Principio de Abierto/Cerrado

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

El segundo principio de SOLID lo formuló Bertrand Meyer en 1988 en su libro "Object Oriented Software Construction" y dice: "Deberías ser capaz de extender el comportamiento de una clase, sin modificarla". En otras palabras: las clases que usas deberían estar **abiertas para poder extenderse y cerradas para modificarse**.
En su blog Robert C. Martin defendió este principio que a priori puede parecer una paradoja. Es importante tener en cuenta el **Open/Closed Principle (OCP)** a la hora de desarrollar **clases, librerías o frameworks.**

3. Principio de Sustitución de Liskov

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

La L de SOLID alude al apellido de quien lo creó, Barbara Liskov, y dice que **"las clases derivadas deben poder sustituirse por sus clases base".**
Esto significa que los objetos deben poder ser reemplazados por instancias de sus subtipos sin alterar el correcto funcionamiento del sistema o lo que es lo mismo: si en un programa utilizamos cierta clase, **deberíamos poder usar cualquiera de sus subclases** sin interferir en la funcionalidad del programa.
Según Robert C. Martin incumplir el **Liskov Substitution Principle (LSP)** implica violar también el principio de Abierto/Cerrado.

4. Principio de Segregación de la Interfaz

"Make fine grained interfaces that are client specific."

En el cuarto principio de SOLID, *el tío Bob* sugiere: "Haz interfaces que sean específicas para un tipo de cliente", es decir, para **una finalidad concreta**.

En este sentido, según el **Interface Segregation Principle (ISP),** es preferible contar con muchas interfaces que definan pocos métodos que tener una interface forzada a implementar muchos métodos a los que no dará uso.

5. Principio de Inversión de Dependencias

"Depend on abstractions, not on concretions."

Llegamos al último principio: "**Depende de abstracciones**, no de clases concretas". Así, Robert C. Martin recomienda:

1. Los módulos de alto nivel **no deberían depender de módulos de bajo nivel**. Ambos deberían depender de abstracciones.
2. **Las abstracciones no deberían depender de los detalles**. Los detalles deberían depender de las abstracciones.

El objetivo del **Dependency Inversion Principle (DIP)** consiste en reducir las dependencias entre los módulos del código, es decir, alcanzar un bajo acoplamiento de las clases.

Conclusión. –

Aplicar estos cinco principios puede parecer algo tedioso, pero a la larga, mediante la práctica y echarles un vistazo de vez en cuando, se volverán parte de nuestra forma de programar.
Nuestro programa será más sencillo de mantener, pero no solo para nosotros, si no más aún para los desarrolladores que vengan después, ya que verán un programa con una estructura bien definida y clara.

Los principios SOLID son eso: principios, es decir, **buenas prácticas** que pueden ayudar a escribir un mejor código: más limpio, mantenible y escalable.
Como indica el propio Robert C. Martin en su artículo "Getting a SOLID start" **no se trata de reglas, ni leyes, ni verdades absolutas**, sino más bien soluciones de sentido común a problemas comunes. **Son heurísticos, basados en la experiencia**: "se ha observado que funcionan en muchos casos; pero no hay pruebas de que siempre funcionen, ni de que siempre se deban seguir."