Reporte de lectura: Object-Oriented Analysis And Design —Design Principles (Part 6)

Las buenas prácticas de la orientación a objetos no se imponen automáticamente, queda a nosotros decidir. Puede no sean reglas específicas, pero sí tenemos líneas de guía, y tenemos principios que podemos utilizar.

Estos principios no son tan genéricos como los conceptos de abstracción, polimorfismo, herencia y encapsulación. Estos usan estas ideas como punto de partida y dan algunas pautas más para tener un mejor diseño.

KISS

Se define como “Keep It Simple, Stupid”. Lo que este principio afirma es que “la mayoría de los sistemas funcionan mejor si se mantienen simples en lugar de hacerlo complejos; Por lo tanto, la simplicidad debe ser una meta clave en el diseño y la complejidad innecesaria debe evitarse”.

DRY

“Dont Repeat Yourself”. Tratar de evitar cualquier duplicado, en su lugar, ponerlos en una sola parte del sistema, o de un método.

YAGNI

“You Ain’t Gonna Need It”. La adición de características extras significa añadir más código para escribir, para mantener, para probar y depurar.

SOLID

S-ingle Principio de Responsabilidad

Un objeto debe tener una y sólo una responsabilidad. Un objeto puede tener muchos comportamientos y métodos, pero todos ellos son relevantes para su única responsabilidad.

O-pen / Closed Principle

Entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para la extensión, pero cerrado para modificación. Siempre que necesite añadir comportamientos adicionales, o métodos, no se tiene que modificar la existente, en cambio, escribir nuevos métodos.

L-iskov Principio de Sustitución

Una super clase puede ser sustituido por cualquiera de sus subclases heredadas en cualquier parte del sistema sin ningún cambio en el código. Esto significa que las subclases deben ampliar la funcionalidad de la superclase y sin que sea sobrescrito.

I-nterface Principio de Segregación

Interfaces deben ser específicos en lugar de hacer muchas y diferentes cosas. Grandes interfaces deben descomponerse en otras más pequeñas, y más específicas.

D-ependency Inversión Principio

Trate de minimizar la dependencia entre objetos mediante el uso de la abstracción.

GRASP

Información de expertos

Cuando se asigna una responsabilidad en la forma de un método o campo, se asigna al objeto que tiene la mayor información al respecto.

Creador

Se trata de determinar que está tomando la responsabilidad de crear los objetos. Se intenta responder a estas preguntas: ¿Quién es responsable de crear los objetos ?, ¿Cómo se crean los objetos en primer lugar? Un objeto contiene a otro.

Bajo Acoplamiento

Esto significa que se intenta reducir la dependencia entre sus objetos.

Alta Cohesión

Cuanto más se tiene una clase que tiene responsabilidades relevantes y enfocadas, mayor es la cohesión que tendrá.

Controlador

Muy común es crear una clase controlador con el solo propósito de gestionar la conexión entre la interfaz de usuario y los objetos de negocio relacionados.

Fabricación pura

En lugar de forzar a que el comportamiento en una clase existente donde no pertenece, lo que significa que se decrementa la cohesión, inventamos en su lugar, creamos una nueva clase.

Indirección

Esta es la idea de que podemos reducir el acoplamiento entre objetos. Lo que podemos hacer es reducir las conexiones directas por poner un objeto de indirección entre ellos para simplificar la cantidad de conexiones que cada objeto tiene que hacer.

Polimorfismo

Tener un objeto que puede tomar la forma de varios objetos. Esto nos permite activar el comportamiento correcto para cada situación.

Código Apestoso

Código Apestoso es un gran termino para cuando se lee código, el código puede ser válido, puede trabajar, pero hay algo en él que simplemente no huele bien.