UNIVERSIDAD AUTONÓMA DE

NUEVO LEÓN

FCFM

UNIDAD DE APRENDIZAJE: LABORATORIO DE DISEÑO ORIENTADO A OBJETOS.

S.O.L.I.D.

PROFESOR. MIGUEL ÁNGEL SALAZAR S.

ALUMNA. VALERIA MARTÍNEZ DE LA ROSA

MATRICULA. 1678575

San Nicolás de los garza, Nuevo León a 29 de marzo de 2017

CONTENIDO.

DESCRIPCIÓN.

Solid es un acrónimo inventado por Robert C. Martin para establecer los cinco principios básicos de la programación orientada a objetos y diseño. Este acrónimo tiene bastante relación con los patrones de diseño, en especial, con la alta cohesión y el bajo acoplamiento.

El objetivo de tener un buen diseño de programación es abarcar la fase de mantenimiento de una manera más legible y sencilla así como conseguir crear nuevas funcionalidades sin tener que modificar en gran medida código antiguo. Los costes de mantenimiento pueden abarcar el 80% de un proyecto de software por lo que hay que valorar un buen diseño.

DESARROLLO.

Son 5 principios basicos, los cuales son:

1. SRP: Single Responsibility Principle
2. OCP: Open/Closed Principle
3. LSP: Liskov Substitution Principle
4. ISP: Interfaze Segregation Principle
5. DIP: Dependency Inversion Principle

SRP: Este principio trata de destinar cada clase a una finalidad sencilla y concreta. En muchas ocasiones estamos tentados a poner un método reutilizable que no tienen nada que ver con la clase simplemente porque lo utiliza y nos pilla más a mano. En ese momento pensamos "Ya que estamos aquí, para que voy a crear una clase para realizar esto. Directamente lo pongo aquí".

El problema surge cuando tenemos la necesidad de utilizar ese mismo método desde otra clase. Si no se re-factoriza en ese momento y se crea una clase destinada para la finalidad del método, nos toparemos a largo plazo con que las clases realizan tareas que no deberían ser de su responsabilidad.

Con la anterior mentalidad nos encontraremos, por ejemplo, con un algoritmo de formateo de números en una clase destinada a leer de la base de datos porque fue el primer sitio donde se empezó a utilizar. Esto conlleva a tener métodos dificiles de detectar y encontrar de manera que el código hay que tenerlo memorizado en la cabeza.

OCP: Este principio atribuido a Bertrand Meyer que habla de crear clases extensibles sin necesidad de entrar al código fuente a modificarlo. Es decir, el diseño debe ser abierto para poderse extender pero cerrado para poderse

modificar. Aunque dicho parece fácil, lo complicado es predecir por donde se debe extender y que no tengamos que modificarlo. Para conseguir este principio hay que tener muy claro cómo va a funcionar la aplicación, por donde se puede extender y cómo van a interactuar las clases.

LSP: Este principio fue creado por Bárbara Liskov, y habla de la importancia de crear todas las clases derivadas para que también puedan ser tratadas como la propia clase base. Cuando creamos clases derivadas debemos asegurar de no re-implementar métodos que hagan que los métodos de la clase base no funcionasen si se tratara como un objeto de esa clase base.

ISP: Cuando se definen interfaces estos deben ser específicos a una finalidad concreta. Por ello, si tenemos que definir una serie de métodos abstractos que debe utilizar una clase a través de interfaces, es preferible tener muchos interfaces que definan pocos métodos que tener un interface con muchos métodos.El objetivo de este principio es principalmente poder reaprovechar los interfaces en otras clases. Si tenemos un interface que compara y clona en el mismo interface, de manera más complicada se podrá utilizar en una clase que solo debe comparar o en otra que solo debe clonar.

DIP: El objetivo de este principio conseguir desacoplar las clases. En todo diseño siempre debe existir un acoplamiento pero hay que evitarlo en la medida de lo posible. Un sistema no acoplado no hace nada pero un sistema altamente acoplado es muy difícil de mantener.

CONCLUSIÓN.

Estos principios, son básicos para el diseño orientado a objetos, en gestión de las dependencias. Estos ayudan al sistema a que sean mejor mantenidos y fáciles de extender durante el tiempo, ya que basándote en estos, se mantiene tu código “limpio”, de hacer tantos cambios que al final, ciertas clases hagan trabajo que no deberían.

Estos son principios que nosotros debemos de aplicar sí queremos un software de calidad.

BIBLIOGRAFIAS.

https://www.genbetadev.com/paradigmas-de-programacion/solid-cinco-principios-basicos-de-diseno-de-clases