Universidad de San Carlos
Facultad de Ingeniería
ITCOE
Software Avanzado
Virginia Chavarría G.
2009-20081

TAREA #2

SOLID

Representa cinco principios básicos de la programación orientada a objetos y el diseño. Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de mantener y ampliar con el tiempo. Los principios SOLID son guías que pueden ser aplicadas en el desarrollo de software para eliminar código sucio provocando que el programador tenga que refactorizar el código fuente hasta que sea legible y extensible. Debe ser utilizado con el desarrollo guiado por pruebas o TDD, y forma parte de la estrategia global del desarrollo ágil de software y desarrollo adaptativo de software.

- **Single Responsibility Principle**: Cada objeto debe tener una sola responsabilidad y esa responsabilidad debe estar completamente encapsulada por la clase. Aplicar SRP nos lleva a tener diseños altamente cohesivos y débilmente acoplados.
- **Open Closed Principle**: Todas las entidades de software deben ser abiertas para ser extendidas, pero cerradas para ser modificadas. Hacer un diseño OCP agrega flexibilidad, reusabilidad y mantenibilidad.
- **Liskov Substitution Principle**: Los subtipos deben ser sustituibles por sus tipos base. LSP se aplica a jerarquías de herencia, especificando que se debe diseñar clases de manera que las dependencias de estas puedan ser sustituidas por subclases sin que el cliente sepa acerca del cambio.
- **Interface Segregation Principle**: No forzar al cliente a depender de cosas que no necesita. Mantener interfaces simples y bien enfocadas.
- **Dependency Inversion Principle**: Módulos de alto nivel no deben depender de módulos de bajo nivel, ambos deben depender de abstracciones. Las abstracciones no deben depender de los detalles, los detalles deben depender de las abstracciones.

DRY

Es un principio del desarrollo de software dirigido a reducir la repetición de patrones de software, reemplazándolos con abstracciones; y varias copias de los mismos datos, utilizando la normalización de datos para evitar la redundancia.

Universidad de San Carlos
Facultad de Ingeniería
ITCOE
Software Avanzado
Virginia Chavarría G.
2009-20081

El principio DRY se establece como "Cada conocimiento debe tener una representación única, inequívoca y autorizada dentro de un sistema". El principio ha sido formulado por Andy Hunt y Dave Thomas en su libro The Pragmatic Programmer. Lo aplican bastante ampliamente para incluir " esquemas de bases de datos, planes de prueba, el sistema de compilación, incluso la documentación". Cuando el principio DRY se aplica con éxito, una modificación de cualquier elemento individual de un sistema no requiere un cambio en otros elementos lógicamente no relacionados. Además, los elementos que están lógicamente relacionados cambian de manera predecible y uniforme, y por lo tanto se mantienen sincronizados. Además de usar métodos y subrutinas en su código, Thomas y Hunt confían en los generadores de código, los sistemas automáticos de compilación y los lenguajes de scripting para observar el principio DRY en todas las capas.

YAGNI

Esto quiere decir que sólo se desarrolle lo que se vaya a usar en el momento, que se piense si se va a usar la funcionalidad y si se va a necesitar en un futuro inmediato. No se debe desarrollar algo que no se vaya a usar “pronto”, aunque se sepa con certeza que se va a tener que implementar en un futuro.

Resumiendo: No se implementa en el momento, se implementa cuando se necesita.

¿Cuáles son las ventajas de este principio?

- Se ahorra tiempo de desarrollo: no se destina a funcionalidad “no útil”.
- No se pierde tiempo de prueba ni de documentación asociado a la funcionalidad no necesaria.
- Se evita pensar en las restricciones de la funcionalidad no necesaria: no se cierra el proyecto en función de la funcionalidad no necesaria.
- El código no crece más de lo necesario, y no se complica.
- Evita que se añada más funcionalidad similar a la que no se va a usar.

Según el YAGNI, incluso si estás totalmente seguro, 200%, de que lo vas a necesitar en el futuro… no lo hagas aún, espérate a que llegue ese momento, no vaya a ser que al final no lo necesites o que con el tiempo cambien las necesidades. *“-Hacerlo ahorrará tiempo en el futuro-”*, pues será en el

futuro, si ocurre, porque lo que es ahora… ahorro, lo que es ahorro no está generando. Además, empleas ese tiempo en cosas más productivas en este momento y no engordas las aplicaciones, recuerda aquello que hablamos en el post de si eres responsable de un software ten muy claros los peligros de medir líneas de código, el esfuerzo no crece de manera lineal según crece el tamaño de una aplicación y cuanto mayor es el software (en número de líneas de código) más cuesta mantener cada línea de código. Todo esto habla de no recargar, de no gastar tiempo, de seguir cosas que han dicho muchos, desde foros ágiles hasta desde el Lean, sin que ello implique no poner los mínimos de flexibilidad y cambiabilidad para el futuro.