UD6. Refactorización

Una breve introducción

Gran parte de las clases, métodos y módulos que forman parte de una aplicación han sido implementados, surge ahora una pregunta. ¿Podemos mejorar la estructura del código y que sea de mayor calidad, sin que cambie su comportamiento? ¿Cómo hacerlo? ¿Qué patrones hay que seguir?

La refactorización es una disciplina técnica, que consiste en realizar pequeñas transformaciones en el código de un programa, para mejorar la estructura sin que cambie el comportamiento ni funcionalidad del mismo. Su objetivo es mejorar la estructura interna del código. Es una tarea que pretender limpiar el código minimizando la posibilidad de introducir errores.

Con la refactorización se mejora el diseño del software, hace que el software sea más fácil de entender, hace que el mantenimiento del software sea más sencillo, la refactorización nos ayuda a encontrar errores y a que nuestro programa sea más rápido.

Cuando se refactoriza se está mejorando el diseño del código después de haberlo escrito. Podemos partir de un mal diseño y, aplicando la refactorización, llegaremos a un código bien diseñado. Cada paso es simple, por ejemplo, mover una propiedad desde una clase a otra, convertir determinado código en un nuevo método, etc. La acumulación de todos estos pequeños cambios puede mejorar de forma ostensible el diseño.

El concepto de refactorización de código se base en el concepto matemático de factorización de polinomios.

FACTORIZACIÓN DE POLINOMIOS
$X^2 - 1 = (X + 1)(X - 1)$

- ✔ **Refactorización**: Cambio hecho en la estructura interna del software para hacerlo más fácil de entender y fácil de modificar sin modificar su comportamiento.

- ✔ **Campos encapsulados**: Se aconseja crear métodos getter y setter, (de asignación y de consulta) para cada campo que se defina en una clase. Cuando sea necesario acceder o modificar el valor de un campo, basta con invocar al método getter o setter según convenga.

- ✔ **Refactorizar**: Reestructurar el software aplicando una serie de refactorizaciones sin cambiar su comportamiento.

El propósito de la refactorización es hacer el software más fácil de entender y de

modificar. Se pueden hacer muchos cambios en el software que pueden hacer algún pequeño cambio en el comportamiento observable.

Solo los cambios hechos para hacer el software más fácil de entender son refactorizaciones.

Hay que diferenciar la refactorización de la optimización. En ambos procesos, se pretende mejorar la estructura interna de una aplicación o componente, sin modificar su comportamiento. Sin embargo, cuando se optimiza, se persigue una mejora del rendimiento, por ejemplo, mejorar la velocidad de ejecución, pero esto puede hacer un código más difícil de entender.

Hay que resaltar que la refactorización no cambia el comportamiento observable del software. El software sigue cumpliendo la misma función que hacía antes.

No obstante, hay ocasiones en las que no debería refactorizar en absoluto. Nos podemos encontrar con un código que, aunque se puede refactorizar, sería más fácil reescribirlo desde el principio. Si un código no funciona, no se refactoriza, se reescribe.