Portafolio del Mes - Septiembre

Autor: Vicente Fuenzalida Marín
Fecha de Entrega: 02 de octubre de 2017
Tema: Principios Fundamentales de Diseño de Software

Code Smells en el desarrollo de software

El artefacto que escogí esta semana, es un breve artículo donde se habla acerca del *code smell* como síntoma de código con potencialidad de contener problemas. Se adjunta una [copia]() en formato PDF en el repositorio.

Artefacto

Escogí este artefacto debido a que no me quedó muy claro a qué se refería el término *code smell*, y si era o no distinto a los llamados *anti-patrones* mencionados también en clases.

Un anti-patrón consiste en un patrón que lleva a desarrollar **malas soluciones** para resolver prolemas, o dicho de forma más sencilla, son 'malas ideas'.

Por otra parte, el *code smell* hace referencia a un **síntoma** en el código fuente de un software, que posiblemente señala un problema más complejo. Son estructuras que pueden estar violando los *principios fundamentales de diseño de software* (abstracción, ocultamiento, cohesión y acoplamiento). En StackExchange se puede encontrar un comentario similar con las diferencias entre ambos términos.

En el artículo observado se mencionan varios *code smells* (aunque existen muchísimos más) como por ejemplo:

- *long methods* (demasiadas responsabilidades y difíciles de mantener).
- *refuse bequests* (herencia de clases con poca/nula utilización de los métodos de la superclase).
- *data clumps* (muchos métodos utilizan los mismos parámetros).
- *duplicate code*
- *middle man* (clases intermedias que solo sirven para delegar a otra).
- *primitive obsession* (usar tipos primitivos para valores que pueden tener un significado más complejo).
- *comments* (comentarios no actualizados o que no aportan información para los desarrolladores).

Reflexión

A partir de la información que entrega el artículo se puede ver la utilidad que brinda conocer estos *smells* y saber identificarlos, ya que permiten ahorrarse muchos problemas al mismo tiempo que hacen el código mantenible y legible por otros desarrolladores.

Lo interesante de los *code smells* es que están directamente relacionados a los *principios de diseño de software*, y si estos principios se aplican al desarrollo de un proyecto, permiten **prevenir** y/o **mitigar** los errores detectados mediante *code smells*.

Me parece particularmente útil estar al tanto de la 'lista completa' de *code smells*, ya que resumen el aprendizaje de otros programadores y de ese modo se puede evitar una equivocación innecesaria.

Desde mi propia experiencia, al leer las descripciones de *code smells* señaladas arriba sé que he visto estas señales o 'síntomas' en código que yo mismo he escrito, aunque la mayoría de las veces no me he detenido a corregirlo (**refactoring**) o a indagar más en el posible problema que se esconde detrás y por lo mismo, más adelante se hace muy tedioso reparar los errores dentro del código.

Haciendo una analogía con la metodología TDD (*test-driven development*), se podría afirmar que tanto escribir los tests antes del código como revisar el código en busca de *code smells* pueden parecer pérdidas de tiempo enormes e innecesarias, pero a la larga ambas proveen una protección frente a problemas de mayor envergadura y permiten producir código mucho más mantenible y fácil de entender, lo que agilizará el desarrollo de los proyectos así como su calidad (muy deseable).
Esto, sumado a la utilización de patrones de diseño adecuados y a buenas técnicas de *refactoring*, guiará el desarrollo de software hacia resultados mejor estructurados y reutilizables.