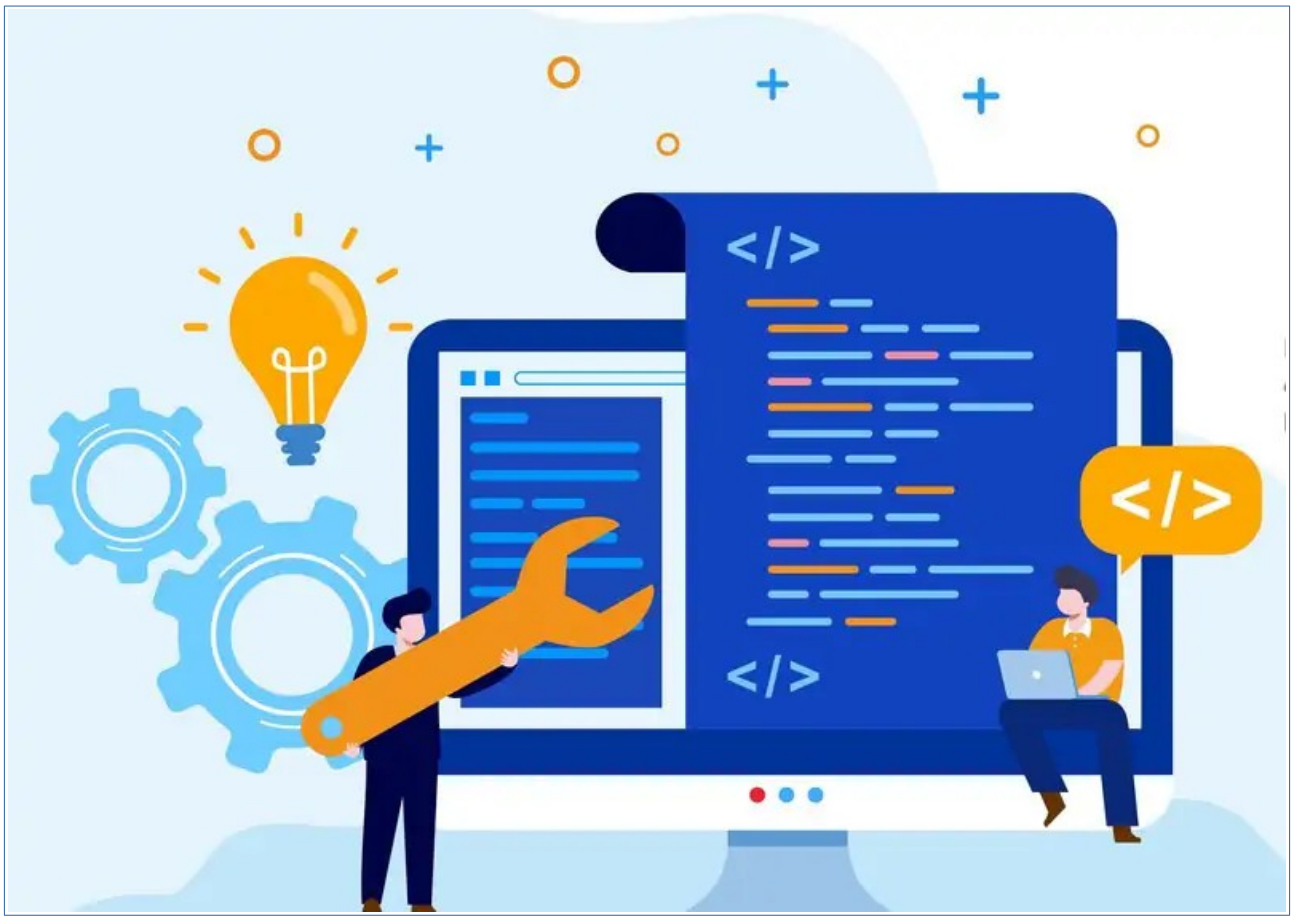
Refactorización de código



Índice

1. ¿Cuál es la definición de refactorización de código?

A groso modo es hacer más legible el código a nivel humano, para que quien vaya a leer el código, o hasta tu mismo en un futuro, pueda entender el código que has escrito.

2. ¿Cuál es la motivación principal detrás de la refactorización de código?

Hacer código comprensible que, a largo plazo no sea un galimatias de leer y en consecuencia, otras personas puedan utilizarlo ya sea para restructurarlo, o entender en mayor profundidad que metodologías utilizas y como se interconectan entre si.

3. Describe dos tipos de refactorización de métodos mencionados en el texto (PDF) y proporciona un ejemplo para cada uno.

- *Extract Method*

  Esta técnica se basa en agarrar, y hacer varios métodos que hagan funciones especificas para no alargar tanto el código.

- *Evitar clausulas else*

  Básicamente, el concepto de evitarlas si es posible es porque llegan a ser un batiburrillo a la hora de intentar leerlo todo, y hasta llega a liar, por eso, por ejemplo, en muchos casos en lugar de un if-else, hay gente que suele recomendar hacer switch porque en general es mucho más legible y comprensible.

4. ¿Qué principios se sugieren para aplicar la refactorización de métodos de manera efectiva?

Ser sencillo de leer, documentar lo mejor posible para el entendimiento de quien lo lee, por extensión poder facilitar el modificar código a largo plazo, y ser lo máximo eficiente en líneas de código posible.

A groso modo, ser sencillo de entender, fácil de modificar, y que ocupe lo menos posible.

5. Explica el concepto de "Principio Tell-Don't-Ask" y cómo se aplica en la refactorización de código.

Vale, alomejor me he equivoco, pero según he leído, es que cuando tu creas algo que quieras representar como un objeto, que en esencia no haya modificadores en otros métodos que manipulen directamente en ese objeto, si no, que el objeto se defina lo primordial de el dentro de si mismo.

6. ¿Cuál es la diferencia entre el método "Extraer una Variable" y el método "Renombrar un Método" en términos de su objetivo y aplicación práctica?

El de extraer una variable, se basa en agarrar cadenas de texto elevadas a través de métodos y demás cosas extensas que, si se repiten mucho, se guardan en una variable pa’ simplificar las cosas, ‘tonces a la hora de leerlo se hace mucho más sencillo darte cuenta de cuando están llamando un método.

Y lo de renombrar un método habla por si solo un poco, básicamente, es agarrar un método, y darle un nombre muy auto-descriptivo de lo que hace.

A nivel legible funciona bastante decentemente.

7. ¿Cuál es el papel del Test-Driven Development (TDD) en el proceso de refactorización y cómo ayuda a garantizar la calidad del código?

Es cuando en programación, agarras y antes de desarrollar code.

Su función a groso modo es básicamente, hacer código eficiente y ir escribiendo sobre el para hacer a posteriori un buen code.

8. Describe al menos tres ejemplos de "code smells" comunes y explica cómo cada uno de ellos puede indicar la necesidad de refactorización.

Esto es del rollo, generar código mal estructurado que a posteriori te puede dar problemas, tanto de implementación, interpretación, o errores en general.

Errores comunes de esto por lo leído, son el rollo de hacer métodos tremendamente largos, desperdigar parte de la funcionalidad de un objeto en varios, o tal vez repetir mucho funciones que se podrían resumir en métodos.

Cosas del estilo.

9. ¿Cómo se define la deuda técnica en el contexto del desarrollo de software y cuáles son las posibles consecuencias de no abordarla de manera adecuada?

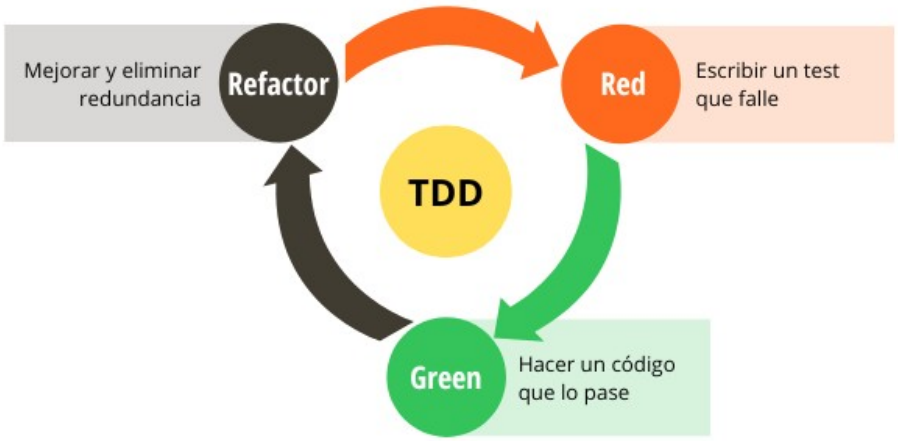
Básicamente, es ponerte a hacer malas praxis, y a raíz de eso, dificultarte a futuro poder editar, rehacer código, o simplemente leerlo.

Las consecuencias acabaran siendo lo dicho antes, que te se dificultará en general poder hacer lo que sea con el code

10. Explique el proceso de "Red, Green, Refactor" en Test-Driven Development (TDD) y cómo se integra con la refactorización del código.

Se empieza con el red, esto es hacer método en el vacío que se sostenga en nada, en el proceso, obviamente solo puede fallar si tratamos de hacer esto.

Tras esto, hacemos la parte de green, que es en la que hacemos el código necesario para sostener el método que hicimos previamente, y finalmente el refactor, que es cuando se agarra y se pone a mejorar todo ese código.



11. Hable sobre la importancia de la automatización de pruebas en el contexto de la refactorización, y cómo puede ayudar a reducir la deuda técnica y mejorar la calidad del código.

En lo que más ayudará al final, es que más tarde precisamente no haya esa deuda técnica, o que se minimice bastante.

Al final del día, cuando más puedas leer del código, menos problemas te dará más tarde.

12. ¿Cuál es la contribución más destacada de Martin Fowler al campo de la refactorización de código y cómo ha influenciado su trabajo en la industria del desarrollo de software?

Ahora mismo ni idea, por lo que he leído por encima, es un hombre que ha contribuido bastante en el tema del mundo de la programación y ofrecer una mejor tipo de praxis a la hora de programar.