Clean Code

Luis Fernando Mendez Marques
College Trainee

Bootcamp Web UI
19/02/2024

---

1. Optimización de Código

La optimización en el sector de TI es el proceso de modificar un sistema de software para ampliar su funcionalidad, haciendo que la aplicación se ejecute más rápida y eficazmente minimizando el uso de recursos informáticos. En otras palabras, un programa informático puede optimizarse para reducir el tiempo de ejecución, el uso de memoria, el espacio en disco, el ancho de banda e incluso consumir menos energía (entre otros). [1]

Un concepto importante en la optimización de programas es la idea de que una optimización suele tener algún tipo de precio. Un ejemplo de ello es que, cuando se optimiza un fragmento de código para que funcione más rápido, el aumento de velocidad puede producirse a costa de la legibilidad del código, el uso de memoria, la flexibilidad del programa o una serie de otros costes. Esto significa que la optimización de programas debe ser un proceso dirigido, con la intención de hacer que un aspecto de un programa funcione mejor, estando dispuesto a sacrificar la eficiencia de otros aspectos. [2]

Niveles de Optimización [3]

- *Diseño*

    - El diseño del sistema puede optimizarse para aprovechar al máximo los recursos disponibles, los objetivos fijados y la carga prevista.

    - El diseño arquitectónico de un sistema desempeña un papel fundamental y afecta de forma abrumadora al rendimiento del sistema.

  Por ejemplo, un sistema que tiene latencia de red puede optimizarse para minimizar las peticiones de red, idealmente haciendo una única petición en lugar de varias.

- *Algorítmos y estructuras de datos*

    - Para garantizar que un sistema está optimizado, debemos asegurarnos de que los algoritmos son constantes O(1), logarítmicos O(log n), lineales O(n) o log-lineales O(n log n).

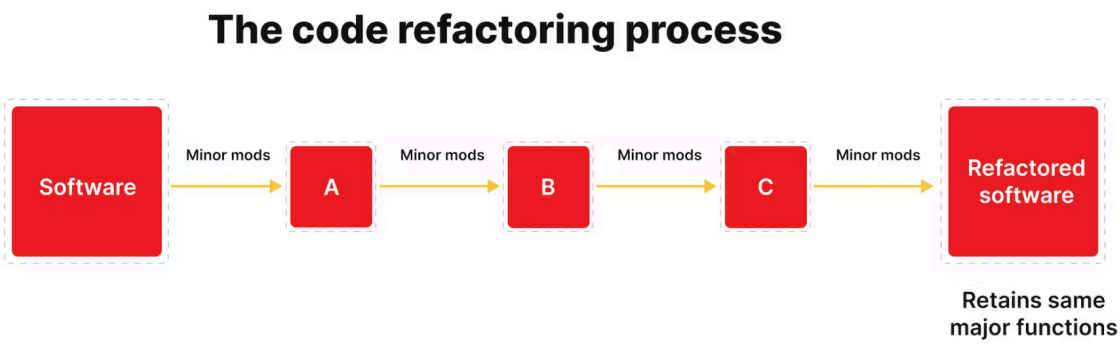
    - Los algoritmos de complejidad cuadrática O(n^2) no escalan de forma eficiente.

    - Los tipos de datos abstractos son más eficientes para las optimizaciones del sistema.

2. Refactorización de Código

La refactorización es un proceso orientado a mejorar el funcionamiento del código fuente a través de la reducción y corrección de acumulaciones de estructuras deficientes (bugs). Todo ello sin agregar funcionalidades al código. [4]

El objetivo principal de la refactorización es abordar la deuda técnica y hacer que el código sea más fácil de entender y trabajar. La deuda técnica se refiere a decisiones de diseño o implementación subóptimas tomadas durante el desarrollo, que pueden obstaculizar futuras actualizaciones o mejoras del software. Al refinar y refactorizar continuamente el código, los desarrolladores pueden gestionar la deuda técnica y mantener una base de código saludable que pueda adaptarse fácilmente a los requisitos cambiantes y los desafíos imprevistos. [5]



Tomado de MadDevs.io

3. Continuous Integration/Continuous Deployment (CI/CD)

La Integración Continua y la Entrega Continua (CI/CD) es un enfoque de desarrollo de software cuyo objetivo es mejorar la velocidad, la eficiencia y la fiabilidad de la entrega de software. Este enfoque implica la integración frecuente del código, pruebas automatizadas y el despliegue continuo de los cambios de software a la producción. Antes de la adopción de CI/CD en la industria del desarrollo de software, el enfoque común era un modelo tradicional de desarrollo de software en cascada. [6]

- *Integración Continua*

  Es una práctica por la cual los desarrolladores integran o combinan el código en un repositorio común, facilitando la realización de test o pruebas para detectar y resolver posibles errores. Con la CI se impide que se desarrollen distintas divisiones de una aplicación que luego puedan tener conflictos entre sí. [7]

- *Entrega Continua*

  Está relacionada con la integración continua y consiste en la automatización del proceso de entrega del software, permitiendo que pueda ser implementado en producción de forma confiable y sencilla. De forma práctica se puede entender la CD como la entrega de actualizaciones de software a los usuarios o clientes de forma sólida y continua. [7]

4. DRY (Don't Repeat Yourself)

El Principio Dry (acrónimo de Don’t Repeat Yourself) es un principio de programación que indica que el código no debe contener líneas redundantes o duplicadas. El objetivo es escribir códigos limpios, legibles y mantenibles, en los que se evite la repetición innecesaria de información. En lugar de ello, se aconseja tratar la información con abstracciones y métodos que permitan reutilizarla en varias partes del programa. [8]

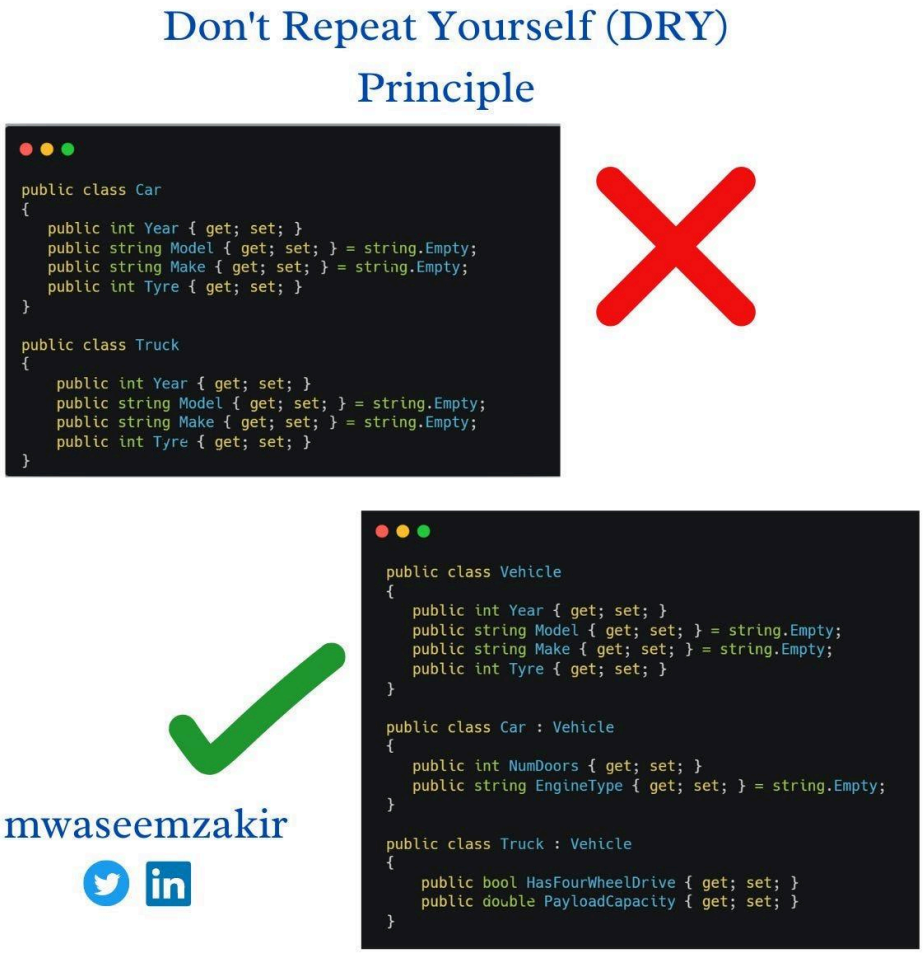
Implementar el Principio DRY [8]

- Utiliza funciones o clases en lugar de repetir el mismo código.
- Escribe una sola función para realizar varias tareas similares.
- Utiliza variables globales cuando sea necesario y evita escribir varias veces el mismo valor.

- Evita copiar y pegar bloques de código en diferentes partes del programa. En su lugar, escribe una sola función que se puede llamar desde los diferentes puntos del programa.

- Crea nombres descriptivos para las variables, funciones y clases para mejorar la legibilidad del código.

- Documenta tu código para hacerlo más comprensible para otros usuarios y desarrolladores.



Tomado de [9]

5. Referencias

[1] "General," Codecademy. [Online]. Available: https://www.codecademy.com/resources/docs/general/programming-optimization. [Accessed: 19-Feb-2024].

[2] E. P., "What Is Program Optimization?," Easy Tech Junkie, 17-Dec-2011. [Online]. Available: https://www.easytechjunkie.com/what-is-program-optimization.htm. [Accessed: 19-Feb-2024].

[3] D. Odhiambo, "Performance optimization in software development - the Andela way - medium," The Andela Way, 24-Sep-2018. [Online]. Available: https://medium.com/the-andela-way/performance-optimization-in-software-development-ae7952ab885e. [Accessed: 19-Feb-2024].

[4] icr_edit, "Refactorización: cómo mejorar el código fuente," icaria Technology, 25-May-2023. [Online]. Available: https://icariatechnology.com/refactorizacion/. [Accessed: 19-Feb-2024].

[5] R. Walker, "¿Qué es la refactorización de software?," Appmaster.io, 12-Sep-2023. [Online]. Available: https://appmaster.io/es/blog/refactorizacion-de-software. [Accessed: 19-Feb-2024].

[6] G. Cocca, "What is CI/CD? Learn continuous integration/continuous deployment by building a project," freecodecamp.org, 07-Apr-2023. [Online]. Available: https://www.freecodecamp.org/news/what-is-ci-cd/. [Accessed: 19-Feb-2024].

[7] Ilimit.com. [Online]. Available: https://ilimit.com/blog/integracion-continua-entrega-continua-despliegue-continuo/. [Accessed: 19-Feb-2024].

[8] Marujita, "Principio Dry," Muy Tecnológicos, 26-May-2023. [Online]. Available: https://www.muytecnologicos.com/diccionario-tecnologico/principio-dry. [Accessed: 19-Feb-2024].

[9] M. Waseem, "Don't repeat yourself (DRY) Principle," Become .NET Pro !, 16-Feb-2023. [Online]. Available: https://medium.com/net-tips/dont-repeat-yourself-dry-principle-3c572f88733d. [Accessed: 19-Feb-2024].