SOFTWARE ARCHITECTURE

CLEAN CODE

Pablo Barrero Cruz – UO289642
Alfonso González-Lamuño de la Calle – UO276976
Rubén Rubio Del Castillo – UO276255

¿Qué es el codigo limpio?

El código limpio, conocido como Clean Code, es una filosofía de desarrollo de software que se enfoca en la creación de código que sea fácil de leer, mantener, robusto y flexible. Esta aproximación busca mejorar la calidad del software, facilitando su comprensión y modificación a lo largo del tiempo.

Codigo Limpio vs Rendimiento.

Recientemente, el desarrollador Casey Muratori planteó un argumento en contra de ciertos aspectos del código limpio, especialmente en lo que respecta al rendimiento. Muratori sugiere que algunas prácticas de código limpio, particularmente aquellas relacionadas con la arquitectura del código, podrían tener un impacto negativo en el rendimiento, especialmente en términos de utilización de la CPU.

Muratori argumenta que la mayoría de los programas no están verdaderamente limitados por E/S (entrada/salida), y que incluso en esos casos, un código mal escrito puede empeorar las cosas. Expone que ocultar excesivamente los detalles de implementación puede dificultar tanto la comprensión del código como la optimización por parte de los compiladores. Sugiere que exponer ciertos detalles internos de bibliotecas o frameworks puede ser beneficioso para mejorar el rendimiento, ya que permite una mejor optimización del código y ayuda a los desarrolladores a comprender el sistema en su totalidad.

Malas prácticas en Clean Code: Impacto en el rendimiento

El controvertido video de Casey Muratori sobre código limpio ha generado un debate en la comunidad de desarrollo de software. Muratori destaca cómo algunas prácticas comunes de código limpio pueden afectar negativamente el rendimiento, incluso en situaciones donde se enfrentan limitaciones de E/S. Utiliza ejemplos, como las interacciones con bases de datos, para ilustrar cómo un código mal escrito puede anular cualquier mejora de rendimiento en la base de datos.

Además, Muratori aborda el concepto de ocultar detalles de implementación y cómo esto puede afectar tanto el rendimiento como la comprensión del código. Argumenta en contra de la idea de ocultar en exceso los detalles de implementación, ya que esto podría obstaculizar tanto la optimización del rendimiento como la comprensión del código en su conjunto.

En resumen, aunque el código limpio busca mejorar la calidad del software, es importante tener en cuenta las implicaciones de rendimiento al aplicar ciertos principios. La discusión generada por el argumento de Casey Muratori destaca la necesidad de encontrar un equilibrio entre la legibilidad y mantenibilidad del código y el rendimiento del software en diferentes contextos de desarrollo.

Pablo Barrero Cruz – UO289642
Alfonso González-Lamuño de la Calle – UO276976
Rubén Rubio Del Castillo – UO276255

¿ Necesidad de implementar codigo limpio?

Casey nos explica que el podcast que la situación más común en las que vamos a necesitar revisar codigo de otra persona es cuando este no nos funciona y esto se puede deber a que el codigo no funciona correctamente o a que tenemos falsas expectativas de cómo funciona ese codigo. Sea como sea, si a la hora de revisar ese codigo nosotros no encontramos un codigo limpio es bastante probable que se complique la tarea de interpretarlo para encontrar una solución por lo que la ausencia de codigo limpio afectaría significativamente a nuestro rendimiento.

Ligado a lo anterior, es importante destacar que en muchas ocasiones menos líneas de codigo no implican un codigo más limpio. Por lo tanto, debemos de intentar no caer en la tendencia actual del abuso de capas de abstracción y encapsulación ya que el exceso de estas puede generar el mismo efecto perjudicial a al a hora de encontrar una solución al problema.

La conclusión a la que podemos llegar con esto es que un codigo limpio no solo debe ser legible sino comprensible. Ya que no es factible tener que rehacer todo un bloque de codigo desde cero debido a que no se entiende el funcionamiento de este.

Principios de implementación del codigo limpio

En el podcast, Casey, hace mucho hincapié en que el codigo limpio te ayuda no solo a entender mejor el codigo, sino que también a usar mejor ese codigo. Es decir, ayuda a la reutilización del codigo, y esto se consigue con el concepto de comprensibilidad, entendiendo no solo que hace el codigo sino como lo hace.

Debido a esto, el establece los siguientes 3 principios para realización de codigo limpio:

- **“El codigo se debe leer más o menos como lo que realmente hace el programa” :** Cuando implementamos un codigo, este debe ser transparente. Debemos no solo tener una idea de lo que hace el codigo, sino que debemos saberlo con seguridad. Uno de los consejos que nos dan en el podcast para lograr esto es huir de las estructuras jerárquicas ya que estas muchas veces tienden a ocultar mucho lo que ocurre en el programa.
- **“La modificación de nuestro codigo tiene que minimizar la repercusión en otras partes del programa”:** Con esto Casey nos quiere decir que hay que intentar que el programa no sea dependiente de nuestro codigo ya que en caso de serlo una pequeña modificación en este puede provocar otros fallos en diferentes partes del programa.
- **“Escribir código que no impida que un futuro optimizador haga que el código se ejecute rápidamente” :** Este principio puede ser el más controversial de todos ya que requiere que entiendas lo que la gente podría necesitar hacer para optimizarlo en un futuro. La importancia que tiene esto es no tener que desechar constantemente sistemas enteros y reescribirlos porque no estaban bien optimizados.

Como identificar codigo limpio:

Una vez definidos los principios del código limpio, hay que explicar también como identificarlos en un codigo y como evaluar si un codigo se pude categorizar como codigo limpio.

- **Legibilidad:** el codigo debe ser legible y no debemos caer la falsa creencia de que menos líneas de codigo implica una mayor legibilidad.
- **Encapsulamiento:** excesivo puede producir el efecto contrario ya que el ocultar el codigo puede complicarnos la labor de arreglarlo o entenderlo.
- **Rendimiento:** debemos constantemente estar pendientes de las gráficas de rendimiento para ser conscientes de cómo se desempeña nuestro codigo.
- **Dependencia:** que nuestro codigo dependa de otro o que dependan mucho de nuestro codigo puede ser un factor determinante para no poder cumplir los principios de un codigo limpio.

Qué hacer al comenzar un proyecto

Antes de empezar un proyecto de software debemos llevar a cabo una planificación teórica previa a la implementación.

Una de las primeras decisiones a tomar es qué lenguaje de programación se va a emplear, y esto debe hacerse analizando el problema que pretendemos solucionar. Se debe tomar una decisión fundada de por qué empleamos las tecnologías que empleamos, puesto que podemos llegar a un punto del desarrollo en el que no es posible continuar por una mala decisión de arquitectura.

Ley de Conway

Cualquier pieza de software refleja la estructura organizacional que la produjo. Los microservicios son una solución de ingeniería de software para la Ley de Conway.

Medidas de calidad de código

Hay otras medidas de calidad de código más allá del rendimiento, aunque esta es la más objetiva. Como la legibilidad o la velocidad a la que puede escribirse dicho código. También podemos decir que un código que sigue los estándares es mejor que uno que no los sigue, pero solo en cuestión de claridad y limpieza.

Sir Tony Hoare ~ Premature optimization is the root of all evil.

No debemos perder el tiempo en optimizar todos los detalles al máximo desde el principio, porque vamos a acabar con un código ilegible, se deben centrar los esfuerzos de optimización en los puntos clave.

Estos lugares clave suelen ser donde se emplean algoritmos complejos y estructuras de datos, eligiendo estas según las operaciones a realizar.