Clean Code, Horrible Performance?

SE RADIO 577

Abel Menéndez Hernández– UO283274
Lucas Castro Antuña – UO289769
Jesús García Ruiz – UO285427

ARQUITECTURA DEL SOFTWARE | ESCUELA DE INGENIERÍA INFORMÁTICA DEL SOFTWARE

Código Limpio

Introducción:

En el mundo del desarrollo de software, la búsqueda de un equilibrio entre código limpio y rendimiento eficiente es una tarea constante. En esta presentación, exploraremos qué es el código limpio, cómo afecta al rendimiento eficiente y por qué es crucial encontrar un equilibrio entre ambos aspectos.

1. Qué es el Código Limpio: [1]

El código limpio es un concepto fundamental en la ingeniería de software que se refiere a la escritura de código de manera que sea fácil de entender y mantener. Según el libro homónimo de Robert C. Martin, libro al que hacen referencia en el podcast, el código limpio es legible, comprensible y elegante, lo que facilita su mantenimiento y evolución a lo largo del tiempo.

La organización y la legibilidad es importante a la hora de que tu código pueda ser entendido por otros desarrolladores, que comprendan lo que hace y como lo hace.

Un código limpio, es más fácilmente escalable y mantenible. Esto supone que está diseñado y estructurado para poder modificarse, corregirse y mejorar con facilidad en el tiempo y que pueda adaptarse sin problemas según crece y evoluciona el sistema.

También es importante la facilidad en la detección de errores que nos ofrece un código limpio para así reducir tiempo en depuración y ahorra costes.

2. Importancia del Rendimiento en el Software:

El rendimiento del software no es simplemente un detalle técnico; es un factor crucial que impacta directamente en la experiencia del usuario, la eficiencia operativa y la competitividad de las empresas en el mercado. Aquí hay varios puntos importantes a considerar:

Experiencia del usuario: El rendimiento del software tiene un impacto directo en la experiencia del usuario. Los tiempos de carga rápidos, las respuestas ágiles a las interacciones del usuario y la fluidez en la ejecución de tareas son aspectos clave que influyen en la satisfacción del usuario.

Competitividad en el mercado: En un mercado cada vez más competitivo, el rendimiento del software puede ser un factor diferenciador crucial. Las empresas que ofrecen aplicaciones rápidas y eficientes tienen una ventaja competitiva sobre sus competidores.

Sistemas embebidos y recursos limitados: En aplicaciones de sistemas embebidos o con recursos limitados, como dispositivos móviles o dispositivos IoT (Internet of Things), el rendimiento del software cobra una importancia aún mayor. Estos sistemas suelen tener restricciones de memoria y procesamiento, por lo que es fundamental optimizar el rendimiento para garantizar un funcionamiento eficiente.

3. Código limpio vs Rendimiento eficiente:[2]

Si bien la búsqueda de un código limpio puede mejorar la legibilidad, mantenibilidad y facilidad de comprensión del código, hay ciertas prácticas que pueden resultar en deficiencias en el rendimiento de la aplicación. Casey Muratori, el entrevistado en el podcast en el cual se basa esta presentación, habla de algunas de ellas:

Uno de los principios fundamentales del código limpio es ocultar los detalles de implementación de las personas que interactúan con él. Este enfoque permite que el código sea más comprensible y fácil de manejar para los desarrolladores. Sin embargo, este nivel de abstracción puede traducirse en una sobrecarga de rendimiento, ya que el ocultamiento de detalles puede requerir estructuras adicionales o un mayor número de llamadas a funciones para acceder a ciertas funcionalidades.

Adicionalmente, cuando se vuelve necesario investigar los detalles de una librería externa que se está usando, bien porque ha empezado a dar problemas con el código que ya existía o bien porque se ha vuelto necesario aumentar la eficiencia de la aplicación, cuantas más capas de abstracción y encapsulación haya entre el usuario y lo que realmente está haciendo el código, encontrar el fallo se vuelve más tedioso y difícil.

Otro principio del código limpio es minimizar el uso de declaraciones condicionales, como los “ifs” y “switches”, y en su lugar favorecer el uso de jerarquías de clases. Si bien esto puede hacer que el código sea más modular y fácil de extender, puede resultar en una menor eficiencia en términos de rendimiento. Casey señala que, a la hora de optimizar un programa en programación GPU, es mucho más sencillo de optimizar un switch, transformándolo en una operación vectorial, que cualquier jerarquía de clases.

Finalmente se menciona que este estilo de código con jerarquías de clases ayuda a hacer el código mucho más legible y fácil de entender, lo cual, si bien puede ser cierto, carece de una referencia, “más fácil de entender en cuanto a qué” y por lo tanto ese mismo código podría haberse escrito de otra forma procedimental igual de legible sin comprometer tanto el rendimiento.

4. Código Limpio y Buen Código:

Casey Muratori nos da tres pautas para entender lo que sería para él un Buen Código:

La primera explica que el código en general debería reflejar de forma clara y precisa lo que el programa realmente hace. Este criterio lo cumple a medias el “Codigo Limpio” pues la encapsulación y abstracción hacen que al leer un programa puedas suponer lo que hace, pero no sepas lo que realmente se está ejecutando, lo puede conllevar acciones inesperadas.

La segunda pauta es que el código debería minimizar la necesidad de realizar cambios en múltiples lugares. Cuando el código está estructurado de forma que los componentes están desacoplados y las dependencias son mínimas, los cambios en una parte del programa son menos propensos a causar efectos secundarios no deseados en otras partes.

La tercera y más importante pauta que se explica es que el código esté diseñado de manera que no impida realizar optimizaciones sensatas en el futuro. Esto significa que el código está escrito de manera eficiente y no contiene prácticas que puedan obstaculizar la mejora del rendimiento. Esta pauta si bien es la más importante también es la más difícil de cumplir pues requiere un mayor conocimiento a cerca de lo que podría ser necesario realizar en un sistema en un futuro. El no cumplimiento de esta pauta también ha llevado a grandes empresas a tirar sistemas enteros y construirlos de nuevo, lo que es muy costoso tanto en tiempo como en dinero.

Finalmente, Casey habla a cerca de los sistemas basados en microservicios y los diferentes lenguajes de programación.

En cuanto a los primeros los describe como una "solución de ingeniería de software para la ley de Conway y no para el ordenador". Se sabe que los Microservicios están muy de moda y hay muchas compañías que los usan, pero también muchas han regresado de un sistema basado en microservicios de nuevo a uno monolítico por las ventajas en cuanto a rendimiento que este proporciona. Estas ventajas se deben a que cualquier tipo de llamada será siempre más rápida si se realiza dentro del propio sistema que si se hace a través de una red.

En cuanto a los lenguajes de programación, aunque algunos como Python pueden llegar a ser hasta cien veces más lentos que otros como C, la elección de este no influye en la buena elaboración de un sistema si se conocen los riesgos y limitaciones de dichos programas y se valora el alcance e impacto que puede tener la utilización de uno respecto a otro. En este aspecto, lo único que no se debe hacer es escoger un programa sin haberse informado lo suficiente y esperar que todo vaya a salir bien, pues lo más seguro es que no sea así.

Bibliografía

[1] "Clean Code with Uncle Bob - YouTube." Accessed: Mar. 15, 2024. [Online]. Available: https://www.youtube.com/watch?v=8SMOB6k3hkM

[2] "SE Radio 577: Casey Muratori on Clean Code, Horrible Performance? – Software Engineering Radio." Accessed: Mar. 15, 2024. [Online]. Available: https://se-radio.net/2023/08/se-radio-577-casey-muratori-on-clean-code-horrible-performance/