ASW-GRUPO 5-S.1

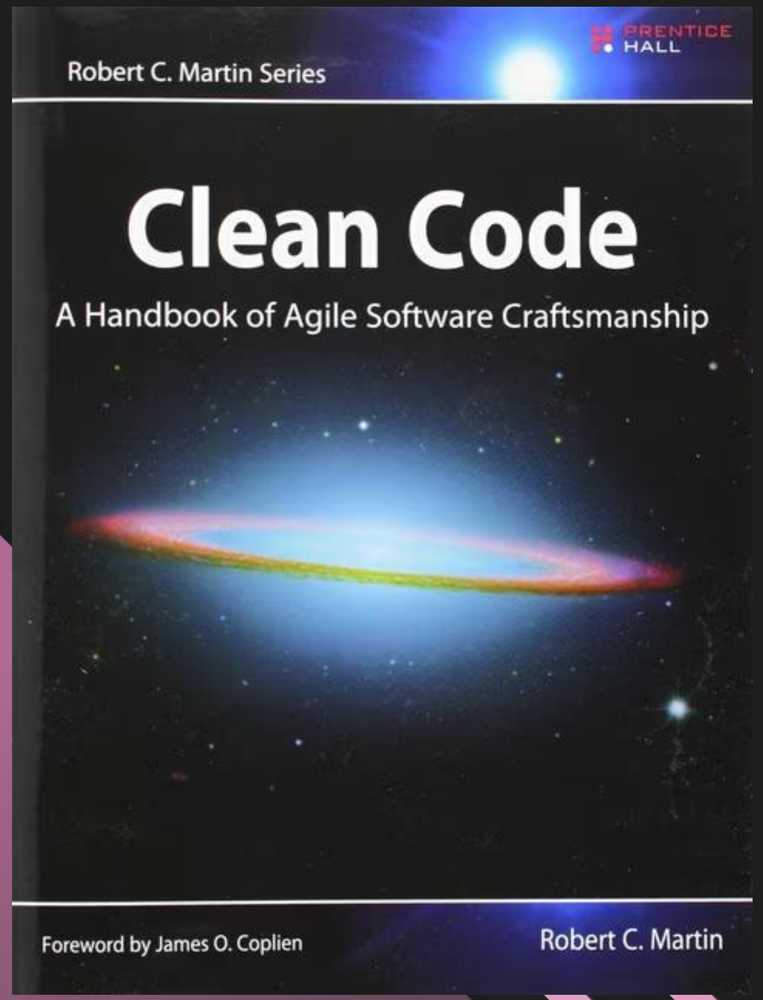
CLEAN CODE

Presentación del Podcast SE Radio 577: Casey Muratori

Presentado por:

Sonia Moro Lauda UO282189
Sandra García Morán UO283182
Lucía Villanueva Rodríguez UO283535

INDICE



Introducción

“If you are tired or distracted, do not code.”
Robert C.Martin

Video Clean “code”, horrible performance

¿Quién es Casey Muratori?

1935

HANDMADE HERO

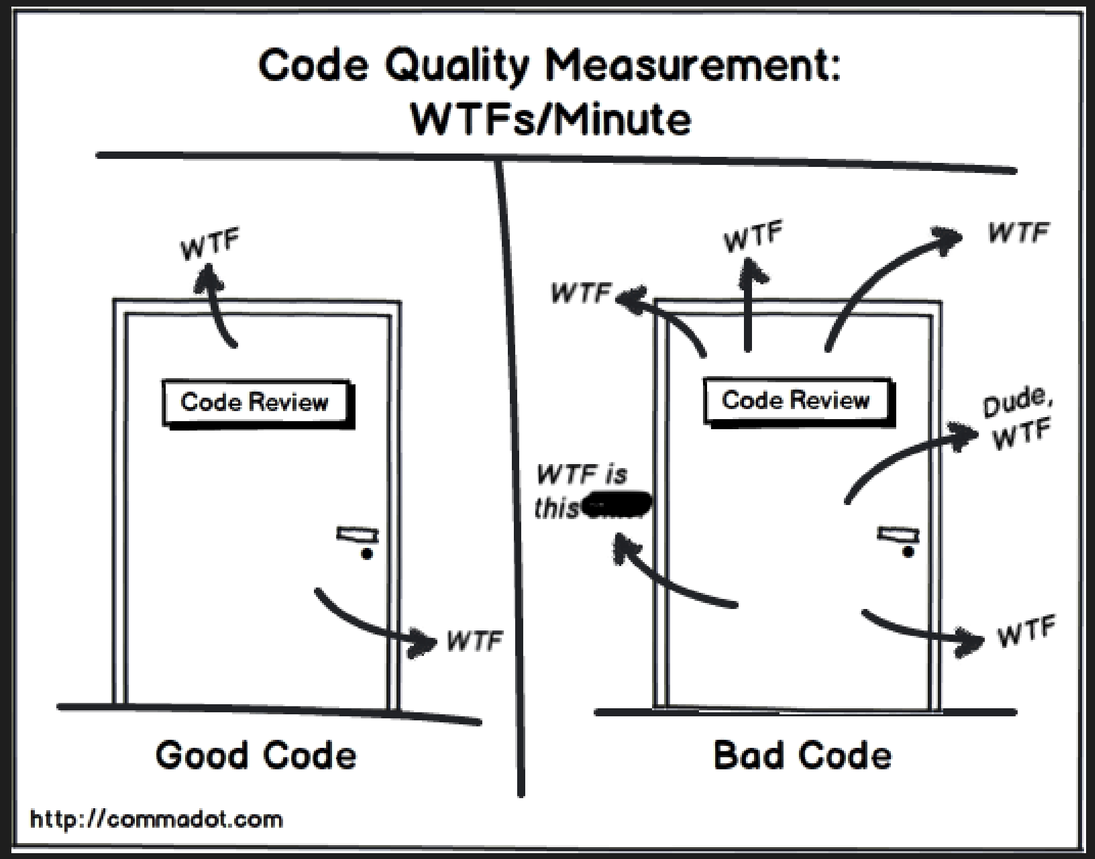
BINK 2

THE WITNESS

PROGRAMACION CONCURRENTE

FICCIÓN INTERACTIVA

¿Qué es Clean Code?



CLEAN CODE ES UN CONJUNTO DE PRINCIPIOS Y PRÁCTICAS PARA ESCRIBIR CÓDIGO QUE SEA FÁCIL DE ENTENDER, MODIFICAR Y MANTENER.

Características Clean Code

1 NOMBRES SIGNIFICATIVOS

```
  public static void main(String[] args) {
      int x = 5;
      int y = multiplyByTwo(x);
      System.out.println("Resultado: " + y);
  }
```

```
public static void main(String[] args) {
    int inputValue = 5;
    int doubledValue = doubleInput(inputValue);
    System.out.println("Resultado: " + doubledValue);
}
```

2 EVITA PALABRAS INNECESARIAS

```
def add_two_numbers(num1, num2):
    result = num1 + num2
    return result
```

```
def add_two_numbers(num1, num2):
    return num1 + num2
```

3 COMENTARIOS Y CLARIDAD

```
# initialize an empty list
my_list = []  # type: List[int]

# loop through the list and print each element
for i in my_list:
    # print the element
    print(i)  # prints the current element
```

4 LEGIBILIDAD Y SIMPLICIDAD

```
def calculate_average(numbers):
    total = 0
    for num in numbers:
        total += num
    return total / len(numbers)

def calculate_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total
```

5 ESTRUCTURA Y CONSISTENCIA

Principios Clean Code

KISS
KEEP IT SIMPLE STUPID

Código simple y directo.
Ayuda a reducir la complejidad.
Beneficios:

- Calidad del código
- Fácil de mantener
- Fácil de modificar

DRY
DON'T REPEAT YOURSELF

Evitar duplicación de código.
Usar refactorización, patrones de diseño.
Beneficios:

- Código mantenible
- Realizar cambios de manera sencilla

YAGNI
YOU AIN'T GONNA NEED IT

Surge a partir de la Programación Extrema (XP)
Implementar solamente lo que se necesita.
Beneficios:

- Ahorro de tiempo de implementación
- Facilitar adaptabilidad
- Evitar complicaciones de código
- Evitar exceso de diseño

Nombres descriptivos

EXPRESAN SU PROPÓSITO

Ayuda a la legibilidad del código y describen su intención.

LONGITUD SEGÚN EL ÁMBITO

-Variables: Longitud proporcional a su ámbito.
-Clases y funciones: Si la clase es pequeña tendrá un nombre largo, y si es grande un nombre corto.

Funciones

01

PEQUEÑAS

Ayuda a la legibilidad y comprensión de código.
Promueve la reutilización.

02

RESPONSABILIDAD ÚNICA

Solo deben realizar una única tarea.
Extracción de métodos

03

POCOS ARGUMENTOS

Como máximo 3
Una palabra por argumento
con nombres significativos

04

EVITAR EFECTOS SECUNDARIOS

No debe causar cambios inesperados alterando otras partes del programa.

Otras técnicas

- **HACER PRUEBAS UNITARIAS**
  Pruebas TDD

- **REGLA BOYSCOUT**

  Dejar el código mejor de como te lo encuentras
  Pequeñas mejoras continuas del código

- **MANEJO DE ERRORES**

  Uso de excepciones
  Try-Catch
  Mensajes de error

- **DOCUMENTACIÓN Y COMENTARIOS**

- **ESTILO DE INDENTACIÓN**

Ejemplo Clean Code:

Version correcta

```java
public static void main(String[] args) {
    int[] numeros = {5, 2, 8, 1, 3};

    Arrays.sort(numeros);
    System.out.println("Array ordenado: " + Arrays.toString(numeros));

    int maximo = encontrarMaximo(numeros);
    System.out.println("El número más grande es: " + maximo);
}

private static int encontrarMaximo(int[] array) {
    if (array.length == 0) {
        throw new IllegalArgumentException("El array no puede estar vacío");
    }
    int maximo = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] > maximo) {
            maximo = array[i];
        }
    }
    return maximo;
}
```

Version incorrecta

```java
public static void main(String[] args) {
    int[] a = {5, 2, 8, 1, 3};

    int m = buscar(a);
    System.out.println("El número más grande es: " + m);
}

private static int buscar(int[] a) {
    Arrays.sort(a);
    System.out.println("Array ordenado: " + Arrays.toString(a));

    return a[a.length - 1];
}
```

Rendimiento

Resulta crucial ya que incide directamente en la eficiencia y la experiencia del usuario final.

1. IMPACTO DE LAS DECISIONES DE DISEÑO EN EL RENDIMIENTO
2. IMPORTANCIA DEL CONOCIMIENTO SOBRE RENDIMIENTO
3. MONITOREO Y OPTIMIZACIÓN DEL RENDIMIENTO
4. ROL DE LA ARQUITECTURA EN EL RENDIMIENTO

01

IMPACTO DE LAS DECISIONES DE DISEÑO EN EL RENDIMIENTO

Ayuda a la legibilidad y comprensión de código. Promueve la reutilización.

02

IMPORTANCIA DEL CONOCIMIENTO SOBRE RENDIMIENTO

Destaca la importancia de comprender las implicaciones de rendimiento al diseñar software.

03

MONITOREO Y OPTIMIZACIÓN DEL RENDIMIENTO

Resalta la necesidad de monitorear y optimizar constantemente el rendimiento del código

04

ROL DE LA ARQUITECTURA EN EL RENDIMIENTO

Explora cómo la arquitectura del software influye en el rendimiento del sistema en general

¿Qué tiene que ver todo esto con el clean code?

¿PREGUNTAS?