

```
1  
2  
3 Clean Code {  
4
```

```
5 [ASW Grupo 5];  
6  
7
```

```
8 < Rubén Rubio Del Castillo U0276255 >  
9
```

```
10 < Pablo Barrero Cruz U0289642 >  
11
```

```
12 < Alfonso González-Lamuño de la Calle U0276976 >  
13
```

```
14 };  
15
```

Índice;

- * **1 - Definición**
- * **2 - Código limpio vs rendimiento**

```
1
2  01 {
3
4
5
6      [Definicion];
7
8
9
10 }
11 ;
12
13
14
```

```
1 Definición 'Contenido' {
```

```
2  
3  
4     01  Facil de leer y de mantener;
```

```
5  
6         02  Robusto;
```

```
7  
8  
9         03  Flexible;
```

```
10  
11  
12     };  
13  
14
```

```
1
2 02 {
3
4
5      [ Código limpio vs
6      Rendimiento ];
7
8
9  }
10 ;
11
12
13
14
```

```
1 Mas limpio no significa mejor </1> {
2
3
4   < Los principios del código limpio a veces
5   pueden afectar al rendimiento >
6
7   < Los programas no siempre estan limitados
8   por la E/S >
9
10  }
11
12
13
14
```

La seguridad también puede ser un problema {

< Ocultar detalles puede dificultar la comprensión y el rendimiento >

< Dar detalles de bibliotecas o frameworks ayuda a la optimización y a conocer el sistema >

}

1. Foreing Code: {

Step 01 ¿Que necesidad tenemos de
implementar codigo limpio?

Step 02 Motivos por los que necesitaríamos
entender codigo ajeno.

Step 03 Legibilidad vs Comprensibilidad

}


```
1  Uso del codigo ajeno{
2
3      ¿Necesidad de implementar codigo
4      limpio?
5          - Entenderlo nosotros mismos.
6          - Que lo entiendan los demás.
7
8      Motivos por los que entender codigo
9      ajeno:
10         - Error en bibliotecas.
11         - Expectativas erroneas.
12
13 }
14
```

Legibilidad </1> {



< Se refiere a la facilidad con la que una persona puede comprender y seguir el código fuente.

}

Comprensibilidad </2> {



< Se refiere a la capacidad de entender el propósito, la lógica y el funcionamiento del mismo. >

}

2. Clean Code Pillars: {

Step 01 Ventajas del uso del código limpio

Step 02 Principios para un buen código limpio


Step 03 Criterios para evaluar la bondad de un código

Step 04 Vinculos entre el código limpio y la optimización


}

Principios{


COMPRESION

 “El código se debe leer más o menos como lo que realmente hace el programa”

MANTENIBILIDAD

 “La modificación de nuestro código tiene que minimizar la repercusión en otras partes del programa”

OPTIMIZACION

 “Escribir código que no impida que un futuro optimizador haga que el código se ejecute rápidamente”

}

Bondad de un codigo{



Legibilidad



Encapsulamiento



Redimiento



Dependencia

}

```
1  {  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12 }  
13  
14
```

(CleanCode === Dinero) ?

```
1 { ¿ Qué hacer al comienzo de un
2
3
4
5
6     < “Daría todo lo que sé por la
7     mitad de lo que ignoro” >
8     — René Descartes
9
10
11
12 }
13
14
```

Ley de Conway {

< Las organizaciones que diseñan sistemas se ven obligadas a producir diseños que son copias de las estructuras de comunicación de estas organizaciones >

— Melvin Conway

Microservicios -> Solución
para la Ley de Conway

}


```
1 Medidas de calidad del código {
2
3
4     Rendimiento
5
6     Legibilidad
7
8     Velocidad a la que puede escribirse
9
10    Sigue los estándares
11
12
13 }
14
```

Optimización {

< Premature optimization is the
root of all evil >

— Sir Tony Hoare

No quiere decir que no hace falta
pensar en el rendimiento

¿Dónde optimizar?

}

Decisiones Meditadas {

	Data Structure	Time Complexity								Space Complexity
		Average				Worst				Worst
		Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
1	Array	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
2	Stack	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$\theta(1)$	$\theta(1)$	$O(n)$
3	Queue	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$\theta(1)$	$\theta(1)$	$O(n)$
4	Singly-Linked List	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$\theta(1)$	$\theta(1)$	$O(n)$
5	Doubly-Linked List	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$\theta(1)$	$\theta(1)$	$O(n)$
6	Skip List	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$
7	Hash Table	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
8	Binary Search Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
9	Cartesian Tree	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
10	B-Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
11	Red-Black Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
12	Splay Tree	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
13	AVL Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
14	KD Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$