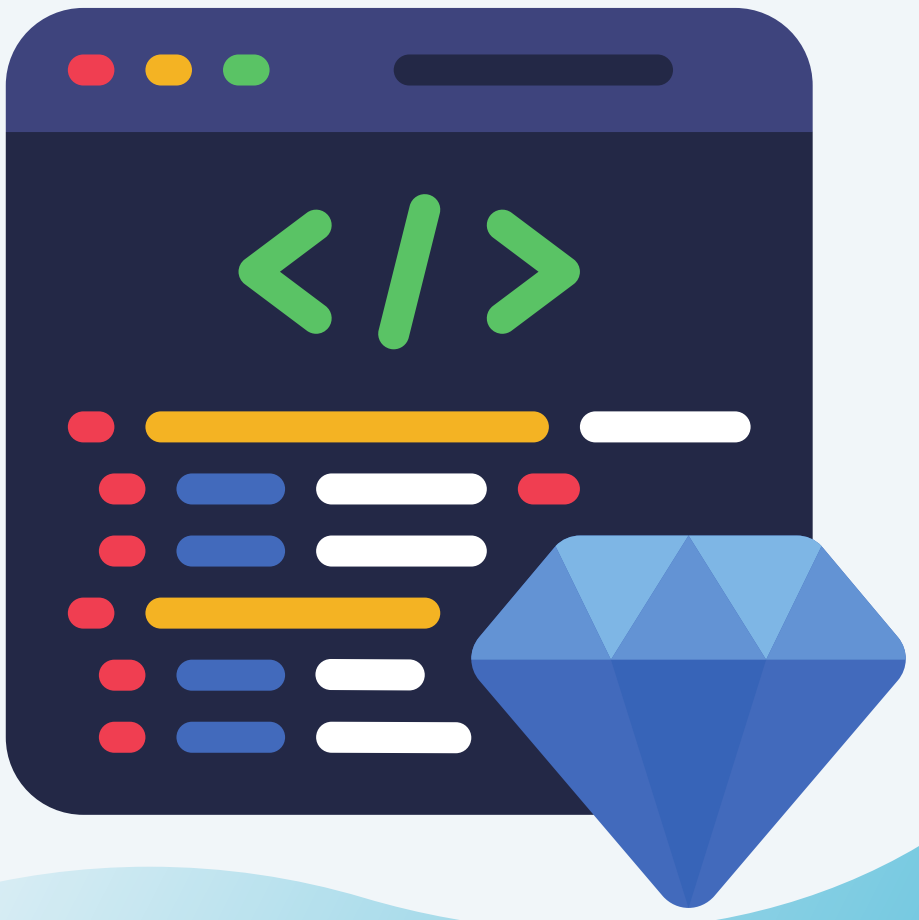
Clean Code

(Guía completa, y por qué aplicarlo te va a llevar al siguiente nivel como desarrollador/a)



PABLO DEL ÁLAMO

¿Qué es el Clean Code?

El clean code no son reglas estrictas, sino principios que te ayudan a escribir código claro y fácil de modificar. La idea es que cualquier desarrollador pueda entenderlo sin problemas. Para que sea fácil de adaptar, debe tener:

- Lógica clara y estructura simple.
- Relaciones entre las partes del código visibles.
- Clases, funciones y variables comprensibles.

PABLO DEL ÁLAMO

Un código fácil de modificar es flexible y escalable, lo que también facilita corregir errores. Las características clave son:

- Clases y métodos pequeños, con una sola responsabilidad.
- Predecibles y bien documentados mediante APIs.
- Cubiertos por pruebas unitarias.

La ventaja: cualquier programador puede trabajar con ese código, evitando problemas de código heredado y simplificando el mantenimiento. Los bugs son más fáciles de encontrar y corregir.

PABLO DEL ÁLAMO

El nombre lo es todo

Los nombres de variables y funciones claros y descriptivos son la base.

Olvídate de los 'x' y 'tmp', y nombra cada cosa con un sustantivo que claramente lo identifique.

Por ejemplo, si tienes una variable que representa un objeto de tipo Casa:

Mal: Casa x = new Casa();

Bien: Casa casa = new Casa();

PABLO DEL ÁLAMO

Comenta solo lo necesario

Los comentarios son como los post-it de la nevera: ¡útiles, pero no abuses de ellos!

Un buen código, que se ha creado siguiendo principios de clean code, debería de poder explicarse solo

PABLO DEL ÁLAMO

Funciones cortas y específicas

Cada función debe estar centrada en un objetivo específico y nada más que eso. Nada de tener funciones infinitas que lo hacen todo.

Por ejemplo, si tienes una función hacerReceta(), no implementes toda la lógica ahí dentro. Crea sub funciones como comprarIngredientes(), pelarPatatas(), encenderSarten(), etc, que sean llamadas dentro de la función principal

PABLO DEL ÁLAMO

Piensa en la legibilidad

Si tu código fuera un libro, ¿sería fácil de leer?

Usa espacios y sangrías adecuadamente.

PABLO DEL ÁLAMO

Evita anidaciones profundas

Cuando en el código empezamos a anidar condicionales, es como crear un laberinto que complica la lectura y el mantenimiento.

A medida que las condiciones crecen, el código se vuelve más confuso y propenso a errores, ya que debemos seguir múltiples ramificaciones antes de llegar al final. Aquí es donde las guard clauses (o cláusulas de guarda) entran en juego.

PABLO DEL ÁLAMO

Ejemplo sin guard clauses:

```python
def procesar_orden(orden):
    if orden is not None:
        if orden.pago_completado():
            if orden.stock_disponible():
                print("Procesando la orden...")
                # Lógica para procesar la orden
            else:
                print("No hay stock disponible.")
        else:
            print("El pago no está completado.")
    else:
        print("La orden es inválida.")
```

PABLO DEL ÁLAMO

Ejemplo con guard clauses:

```python
def procesar_orden(orden):
    if orden is None:
        print("La orden es inválida.")
        return

    if not orden.pago_completado():
        print("El pago no está completado.")
        return

    if not orden.stock_disponible():
        print("No hay stock disponible.")
        return

    print("Procesando la orden...")
    # Lógica para procesar la orden
```

PABLO DEL ÁLAMO

Mantén la consistencia

Al escribir código, es importante mantener un estilo uniforme en todo el proyecto.

Esto significa que si decides seguir ciertas convenciones o reglas (como nombres de variables, uso de espacios, formato de funciones, etc.), debes mantenerlas en todo tu código.

Cambiar de estilo a mitad de camino puede hacer que el código sea confuso, difícil de leer y más propenso a errores.

PABLO DEL ÁLAMO

Mantén la consistencia

Ejemplo:

Código con estilos uniformes:

```
def calcular_area_rectangulo(base, altura):
    area = base * altura
    return area
```

Código con estilos mezclados:

```
def calcularAreaRect(base, altura):
    Area = base*altura;
    return Area
```

PABLO DEL ÁLAMO

Código DRY

No repitas código innecesariamente cuando puedes reutilizar y abstraer lógica.

Si ya tienes una funcionalidad en un lado, y ahora necesitas usarla en otro, no repitas el código, extráelo a un sitio común, y úsalo en ambos.

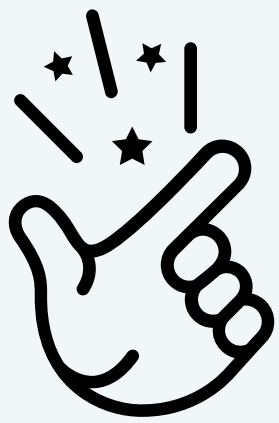
PABLO DEL ÁLAMO

Evita la tentación del código ingenioso

No intentes sorprender o demostrar lo mucho que sabes con códigos ultra complicados.

La simplicidad siempre es mejor y es más fácil de mantener.

Un/a buen/a desarrollador/a siempre debe buscar la solución más eficiente, sencilla y mantenible, no la más complicada.



PABLO DEL ÁLAMO

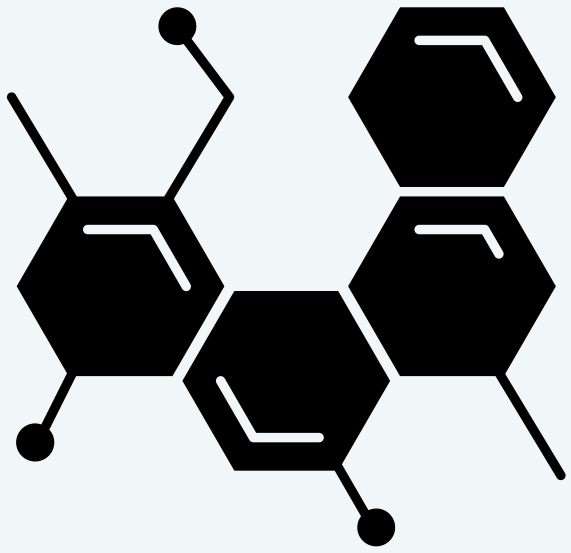
Detalles técnicos en su lugar

La lógica técnica (como la conexión a bases de datos, la configuración de servicios externos o la gestión de APIs) debe estar contenida en secciones bien definidas del sistema, y no mezclada con la lógica de negocio.

PABLO DEL ÁLAMO

Sigue los principios SOLID

Adopta los principios SOLID como un mantra para estructurar tu código de manera robusta y escalable
(tengo otro post sobre esto en el perfil 😉)



PABLO DEL ÁLAMO

Evita números “hardcodeados”

En lugar de meter números a pelo en tu código, define constantes para referenciarlos.

Ejemplo (antes y después):

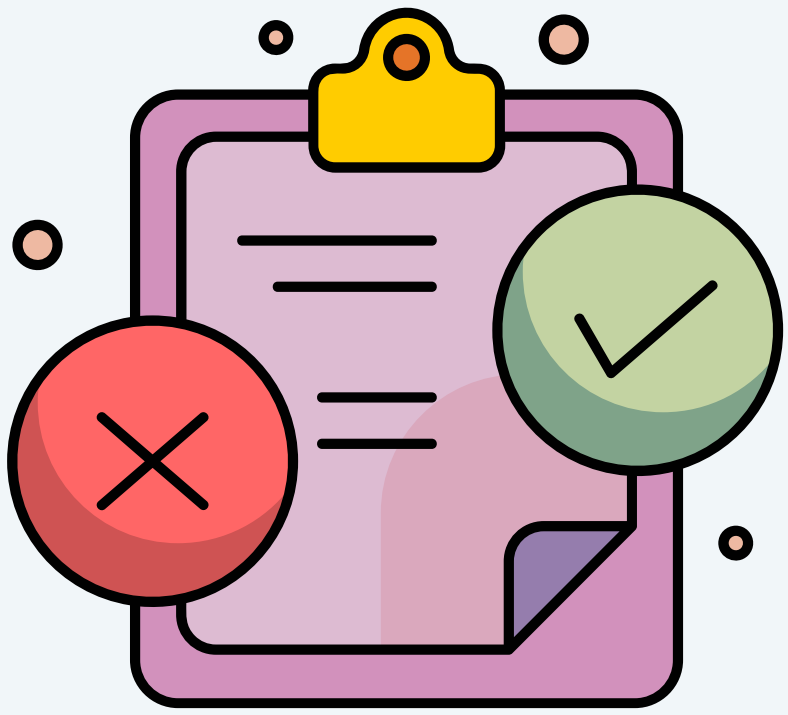
```
def calculate_discount(price):
    discount = price * 0.1 # 10% discount
    return price - discount

def calculate_discount(price):
  TEN_PERCENT_DISCOUNT = 0.1
  discount = price * TEN_PERCENT_DISCOUNT
  return price - discount
```

PABLO DEL ÁLAMO

Testea tu código

Código limpio también significa código probado.
Escribe tests unitarios siempre.



PABLO DEL ÁLAMO

Refactoriza constantemente

No te quedes con tu primer intento como definitivo. La refactorización es el camino a la perfección y la legibilidad

PABLO DEL ÁLAMO

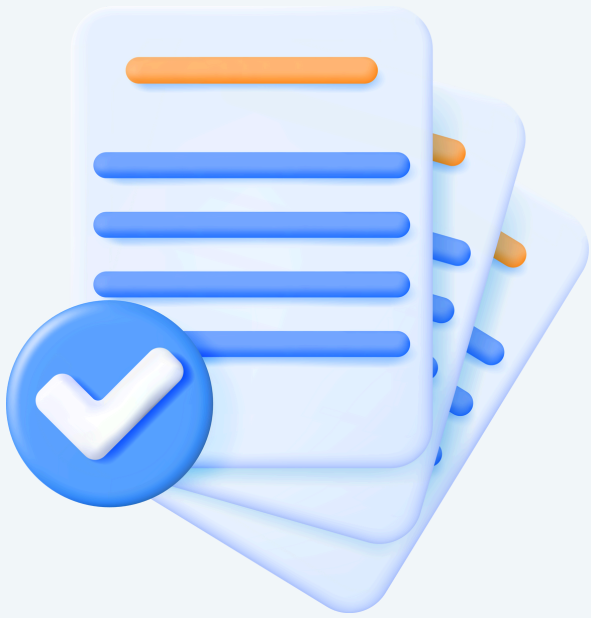
Usa herramientas de análisis

Herramientas como SonarQube o ESLint actúan como detectives para encontrar malos patrones en tu código.

PABLO DEL ÁLAMO

Documenta las funciones

Antes de cada función deberías generar un comentario explicando qué hace, que recibe y qué genera. Para ello existen ciertas herramientas como Javadoc en Java.



PABLO DEL ÁLAMO

Revisión con compañeros

Pide a tus compañeros que revisen tu código, y haz lo mismo con ellos. 4 ojos siempre ven más que 2.

PABLO DEL ÁLAMO

¿Te ha resultado útil?

- Comparte esta guía con tu equipo o amigos desarrolladores.
- Guárdala para tenerla siempre a mano.
- ¡Dale un like o comenta si tienes preguntas!

PABLO DEL ÁLAMO