DISEÑO DE CÓDIGO

El costo de poseer código no mantenible se mostrará en incremento en el futuro cuando debamos realizar cambios o actualizar alguna funcionalidad.

Objetivos:

- **Mantenibilidad**
- **Simplicidad**
- **Claridad**
- **Flexibilidad**
- **Legibilidad**

Aspectos a cumplir:

Nombres

- Nombres significativos, pronunciables y que revelen la intención de lo que se quiere nombrar.
- No dejar valores numéricos fijos, usar constantes con nombres claros.
- Nombres dentro del dominio del problema.

Funciones

- Con respecto a las funciones/métodos, que sean pequeñas (aprox. 8 líneas), que realicen una sola cosa (Single Responsibility Principle).
- Comprender los distintos niveles de abstracción para reducir el tamaño de funciones y hacer una sola cosa por función.
- Evitar el uso de switch (muchas responsabilidades).
- Argumentos: uno es bueno, cero es mejor.
- Respetar DRY y el “Principle of Least Surprise”.

Comentarios

- Legales
- Informativos
- Clarificación
- Advertencias

- TO DO
- Ampliar información
- Malos comentarios: redundancias, errores, ruido, marcadores de posición, código muerto.

Excepciones

- Usarlas en vez de códigos de error.
- En general, no retornar NULL.

Test unitarios

- Casos de prueba aislados, con un único método assert.
- Convención para los nombres de los casos de prueba.

"Cualquier tonto puede escribir código que una computadora puede comprender. Buenos programadores escriben código que otros humanos pueden entender."

CRITERIOS DE BUEN DISEÑO

Análisis ≠ Diseño

El análisis es acerca del **qué** y el diseño del **cómo**.

¿Por qué necesitamos un buen diseño?

- Para manejar el cambio
- Para tener un delivery rápido
- Para lidiar con la complejidad

¿Cómo sabemos que un diseño es malo?

- Rigidez
- Fragilidad
- Inmovilidad

Cosas causadas por las incorrectas dependencias entre módulos.

Un buen diseño debe tener alta cohesión y bajo acoplamiento.

Para eso, existe el principio de diseño SOLID.

SOLID: cómo lograr un buen diseño

- **S**ingle Responsibility Principle (SRP): única responsabilidad.
- **O**pen-Closed Principle (OCP): abierto para extensión y cerrado para cambios.
- **L**iskov Substitution Principle (LSP): toda interfaz hija debe respetar a la interfaz del padre.
- **I**nterface Segregation Principle (ISP): separación de interfaces según clientes, ver lo mínimo necesario.
- **D**ependency Inversion Principle (DIP): módulo de alto nivel no puede depender de otro de bajo nivel.