**Principios DRY, KISS, YAGNI, WET**

Aumentar la calidad interna de nuestro código

Conocidos principios solid

Deuda tecnologica

**Principio DRY, DONT REPEAT YOURSELF o No te repitas a ti mismo**

No te repitas, Evitar código duplicado o repetitivo, Una pieza de código debe ser única, no ambigua, tener un objetivo y responsabilidad definida.

**Ventajas**

* Mantenibilidad del código: es mucho más fácil cambiar el código de una sola parte que de todas las veces en donde lo hayamos escrito
* Legibilidad: es mucho más fácil de leer
* Reusabilidad: cuando necesitemos hacer uso de este código, lo llamamos, no debemos volverlo a escribir
* Testeabilidad: Mucho más fácil de testear, como nuestro código esta en un solo lugar es mas sencillo hacer las pruebas unitarias

Incrementa la velocidad del desarrollo (Reusabilidad)

**Ejemplo práctico:**

**Principio WET, Write Everything Twice, Escribir todo dos veces**

Repetir cada cosa, es un **ANTI-PRINCIPIO** es lo contrario al anterior visto, principio DRY. Siempre debemos evitarlo

**Principio KISS, Keep it simple, stupid, Mantenlo sencillo, estupido**

Establece que la mayoría de los sistemas funcionan mejor si se mantienen simples que si se hacen complejos, por ello, la simplicidad debe ser mantenida como un objetivo clave del diseño.

Mantener las cosas simples,

Aplicar a otras áreas de la vida, no solo al software

La implementación debe ser sencilla,

Evitar soluciones excesivamente completas o sobre diseños cuando no sean necesarios

**Ventajas:**

* Legibilidad del código: ya que todo es más sencillo
* Incrementar la velocidad de desarrollo ya que nos enfocamos en lo estrictamente correcto
* Mantenibilidad: Como el código es mas sencillo, es más fácil hacer revisiones o modificaciones en el mismo, que cuando hacemos partes complejas de código
* Código de mayor calidad y probablemente de menor deuda técnica: Es mucho mejor enfocarnos en lo que nos es solicitado solamente

**Ejemplo práctico:**

Método para devolver el día de la semana según un número, implementar la misma solución de una forma más sencilla.

Switch con 7 retornos

**Principio YAGNI, You Arent gonna need it, No vas a necesitarlo**

No agregar funcionalidades que no se requieran, evitar el código inflado, dejar de meter “código por si acaso”, enfocarnos en lo que realmente vamos a necesitar

**Desventajas:**

* Incremento del tiempo de desarrollo: Puesto que estamos malgastando el tiempo en crear funcionalidades que no son requeridas.
* Mayor complejidad para testear: Al tener funcionalidades agregadas, aumenta la complejidad de testeo.
* Dificulta la legibilidad: Cuando queremos leer el código funcional nos topamos con la existencia de código que no se está utilizando, entonces empezamos a hacernos preguntas como “Esto para que sirve” y si mas adelante queremos deshacernos de esto porque enserio no lo utilizamos, va a generar mayores problemas
* Puede incrementar los costes generales sin generar beneficio a cambio: Al añadir funcionalidades que no nos están requiriendo pues, los clientes no nos van a pagar mas por hacerlo, ellos nunca las solicitaron, entonces su escritura no nos genera beneficio alguno

**Ejemplo práctico:**

**Aplicar dry y Kiss**

**Evitar wet y yagni**

DRY – WET

KISS - YAGNI