Principios Yagni, Tell don't ask, DRY y KYSS

DD

Doctrina del desarrollador

Indice

- Principios
  - Yagni
    - ¿Que es?
    - ¿Cómo cumplir con Yagni?
    - Ventajas
  - Tell don't ask
    - ¿Que es?
    - ¿Que problema podemos resolver?
    - Ventajas
  - DRY
    - ¿Que es?
    - Situaciones donde duplicamos código
    - Ejemplo
  - KYSS
    - ¿Que es?
    - Recomendaciones

Yagni

¿Que es Yagni?

¿Qué es?

Es un principio que surgió de la programación extrema y que establece que un programador no debe agregar funcionalidad hasta que lo considere necesario

Objetivo

Solo se debe desarrollar lo que se vaya a usar en el momento, que se piense si se va a usar la funcionalidad y si se va a necesitar en un futuro inmediato. No se debe desarrollar algo que no se vaya a usar «pronto», aunque se sepa con certeza que se va a tener que implementar en un futuro

You aren't gonna need it

DD

Doctrina del desarrollador

¿Cómo cumplir con Yagni?

- Centrarse en los requerimientos, la mejor manera de hacerlo es con TDD (Test Driven Development)
- No crear soluciones complejas que no se ajusten a los requerimientos actuales del proyecto y que no sean necesarias para cumplir con los objetivos del negocio
- No desarrollar código que no se utiliza actualmente en el proyecto, y que no se espera que se utilice en el futuro cercano
- Eliminar el código comentado, si un código esta comentado significa que no lo necesitas ahora, así que puedes eliminarlo. Si más adelante surge la necesidad de recuperar este código, se puede hacer fácilmente con las herramientas de control de versiones
- Hacer Pull Requests pequeñas

Ventajas

- Se ahorra tiempo de desarrollo: no se destina a funcionalidad «no útil»
- No se pierde tiempo de prueba ni de documentación asociado a la funcionalidad no necesaria
- Se evita pensar en las restricciones de la funcionalidad no necesaria
- El código no crece más de lo necesario, y no se complica
- Evita que se añade más funcionalidad similar a la que no se va a usar

Tell don't ask

¿Que es Tell don't ask?

¿Qué es?

Es un principio de la programación orientada a objetos (POO) que nos recuerda que no tenemos que usar los objetos para pedirles cosas o datos y según la información que nos devuelven tomar decisiones.
Por el contrario, el principio nos sugiere es decirles a los objetos que hagan cosas y estos objetos internamente tomaran sus propias decisiones según de su estado actual. encaja perfectamente con la idea de que un objeto se define por su comportamiento y además es fundamental para sacar provecho al polimorfismo y la inversión de control

¿Que problema podemos resolver?

Problema

Si no aplicamos el principio podemos pecar del acoplamiento, esto sucede al acoplarnos al objeto que estamos instanciando (es más fácil de entender con objetos, pero puede ser cualquier cualquier cosa) lo cual esto genera un problema a nuestro código que afectará a la reutilización y al mantenimiento: el acoplamiento

```
class Bad {
    // Bad implementation
    // This is a bad implementation if we need to add other logic to save the result
    // because we are going to couple ourselves to the implementation of this method
    // where we instantiate the object
    setFinalDate(date) {
        this.finalDate = date;
    }
}


class Good {
    // Good implementation
    setFinalDate(date) {
        const finalDateAux = date;
        const houre = Math.floor(((this.fechaFinal.getTime() - this.fechaInicio.getTime()) / 3600000));
        if (houre > 12) {
            this.finalDate = finalDateAux;
        }
    }
}
```

Ventajas

- Independencia del objeto
- Independencia de la clase/función/etc del objeto que estamos instanciando
- Responsabilidades bien definidas y desacoplamiento
- Polimorfismo

DRY

¿Que es DRY?

¿Qué es?

Según este principio toda pieza de información nunca debería ser duplicada debido a que la duplicación incrementa la dificultad en los cambios y evolución posterior, puede perjudicar la claridad y crear un espacio para posibles inconsistencias

Objetivo

El objetivo principal de DRY es reducir el número de repeticiones de código. Esto mejora la calidad del código y facilita que todos puedan agregar otras funciones o realizar cambios en el futuro

Situaciones donde duplicamos código

- La duplicación puede parecer impuesta: El desarrollador puede pensar que el sistema parece requerir duplicación
- La duplicación puede ser involuntaria: El desarrollador puede no darse cuenta de que está duplicando información
- La duplicación por impaciencia: El desarrollador puede tener flojera y duplicar porque parece lo más fácil. Eventualmente esto te afectará más adelante
- La duplicación por falta de coordinación: Múltiples desarrolladores en un equipo o diferentes equipos pueden duplicar información

Ejemplo

función existente hace "casi" lo mismo

En muchas ocasiones, nos encontramos que una función existente hace "casi" lo mismo que queremos, pero necesitamos que haga algo extra o que no lo haga. Podemos aplicar estas opciones:

- Revisa si tu lenguaje de programación detecta el número de parámetros recibidos y mediante esa función lee el nuevo parámetro y condiciona la nueva lógica
- Hacer algo de "refactorización"
- Renombrar las funciones y adaptar tu código, es doloroso, pero a la larga te beneficiarás
- Si usas algún lenguaje de Programación Orientada a Objetos haz uso de la sobrecarga de funciones

A veces se necesita repetir algún código

En cuanto al código, la idea no es evitar la repetición de fragmentos de código en todos los casos. A veces, tiene sentido y es menos complicado permitir que algunas líneas se repitan en diferentes clases. Donde realmente debes aplicar el principio de evitar la repetición es cuando estás creando representaciones funcionales de tu aplicación. Por ejemplo, si tienes una tarea recurrente que implica recuperar datos del cliente y aplicar una cierta transformación a esos datos en diferentes partes de la aplicación, en ese caso, es valioso crear una función llamada, por ejemplo, "getClientInfo"

KYSS

¿Que es DRY?

¿Qué es?

Es un principio de diseño y programación en el que la simplicidad del sistema se declara como objetivo o valor principal y debe evitarse una complejidad innecesaria

Objetivo

Seguir el principio Keep It Simple, Stupid (KISS) permite desarrollar soluciones fáciles de usar y mantener

Keep It Simple, Stupid

DD

Doctrina del desarrollador

Recomendaciones

- Mantén los métodos y las clases pequeños
- Usa nombres claros para las variables
- No reutilices variables
- Divide el problema en partes más pequeñas
- No abuses de los comentarios
- Evita el código duplicado

Referencias

- **Disrupciontecnologica:**
  https://www.disrupciontecnologica.com/tell-dont-ask/
- **Paradigmadigital:**
  https://www.paradigmadigital.com/dev/5-reglas-diseno-software-simple/#:~:text=3%20YAGNI%3A%20You%20aren't%20gonna%20need%20it&text=De%20esto%20se%20trata%20YAGNI,lo%20he%20hecho%20muchas%20veces%20.
- **Adictosaltrabajo**
  https://www.adictosaltrabajo.com/2015/10/12/yagni/
- **Codeyourapps**
  https://codeyourapps.com/el-principio-dry-no-te-repitas/
- **Joaquin.medina**
  http://joaquin.medina.name/web2008/documentos/informatica/documentacion/logica/OOP/Principios/2012_07_30_OopNoTeRepitas.html
- **Manuelzapata**
  https://manuelzapata.co/principio-kiss-keep-it-simple-stupid/