

# Clean Code Avanzado

Alberto Basalo, Vitae Digital

# Contenido

- 1** Principles of Software Design
- 2** SOLID Principles
- 3** Design Patterns
- 4** Software Architecture

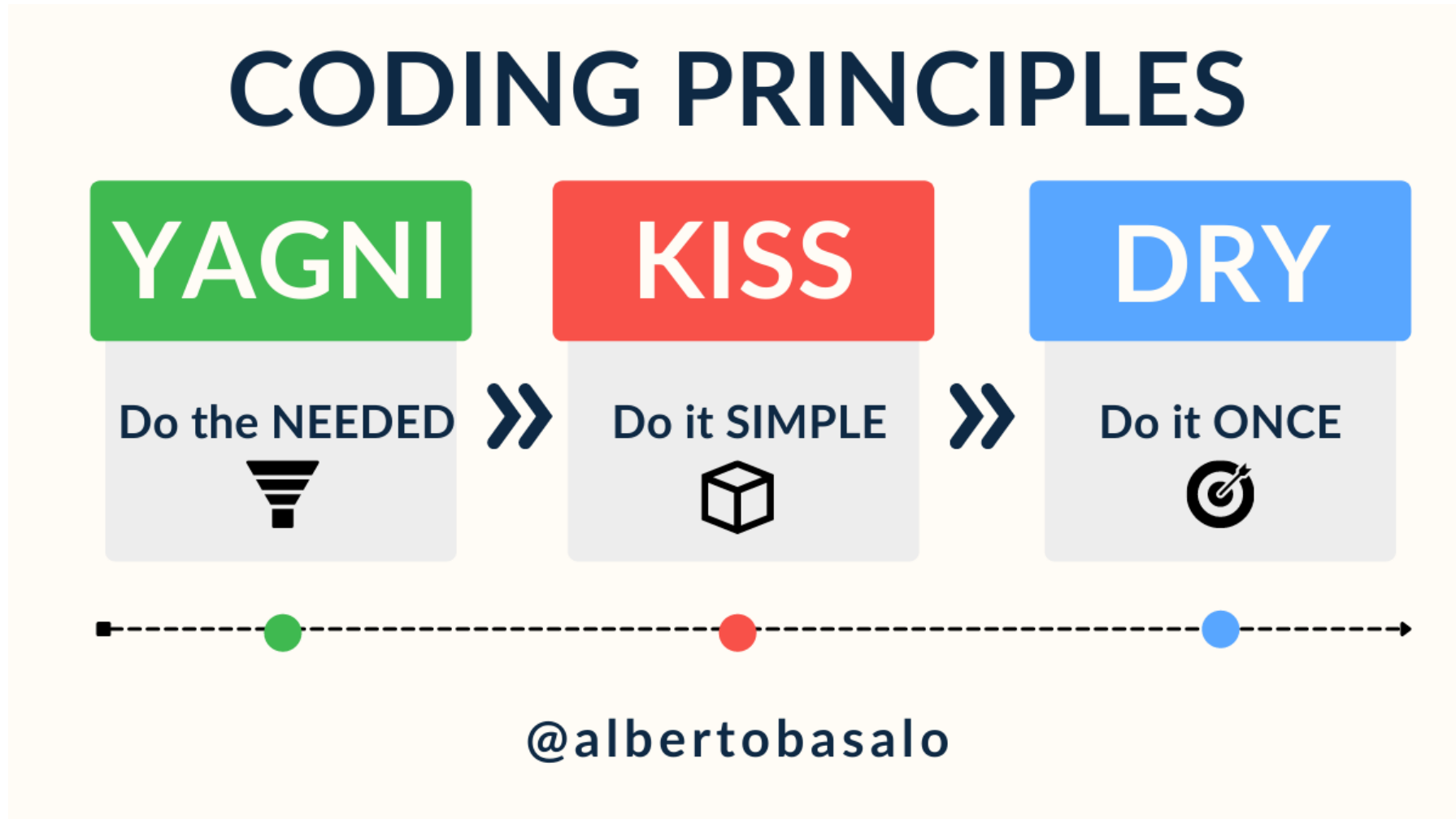
# ❶ Principles of Software Design

1 YAGNI, KISS, DRY

2 Obstacles for change

3 Metrics, preventers, and facilitators

# 1 YAGNI, KISS, DRY



# **1** 2 Obstacles for change

## **Rigidez**

Un cambio afecta a muchas partes. Al cambiar un objeto hay que cambiar otros muchos.


## **Fragilidad**

Las errores saltan en lugares inesperados. Los cambios en un objeto tienen efectos en otros muchos.

## **Inmovilidad**

No se puede reutilizar el código fuera de su entorno. Los cambios en un objeto dependen de otros muchos.

# 1 3 Metrics, Preventers and Facilitators

-  El **acoplamiento** es el principal enemigo del cambio
  - Se puede **medir**
  - Se puede **evitar**
  - Se puede **limpiar**



# Reduce métricas de acoplamiento

- **Eferente**

- ¿Cuántas dependencias uso?

- **Aferente**

- ¿Cuántos dependientes me usan?

- **Exposición**

- ¿Cuántas funcionalidades usan?

- **Tamaño**

- ¿Cuántas instrucciones tengo?

# ✖ Change preventers

- **Feature envy**
  - Use your properties
- **Inappropriate Intimacy**
  - Reduce knowledge
- **Primitive obsession**
  - Aportar cohesión con invariantes
- Divergent change.
  - Una clase que se cambia por diferentes razones.
- Shotgun surgery.
  - Un cambio que requiere muchos cambios. Difícil encontrarlos, fácil olvidarse.
- Cyclomatic complexity.
  - Número de rutas únicas por anidamiento, switches y condiciones complejas



# Change Facilitators

- **Tell don't Ask**
  - Reduce calls
- **Law of Demeter**
  - Don't talk to strangers
- **Command-Query Separation**
  - Cada método debe un comando o una consulta; pero no ambos.
- **P.O.L.A. Principle Of Least Astonishment.**
  - Ni me sorprendas ni me hagas pensar.
- **H.P. Hollywood principle.**
  - No nos llames, ya te llamaremos.
- **CoC Convention over configuration.**
  - Establecer convenios que minimicen decisiones.

## 2 SOLID principles

S.O.L.I.D.


# Single responsibility

 *Nunca debe haber más de una razón para que una clase cambie.*

 Clases grandes, muy tentador seguir añadiendo funcionalidad

 Genera clases pequeñas de alta cohesión.



## Open / Closed

 *Las entidades de software deben estar abiertas a la extensión, pero cerradas a la modificación.*

 Las condiciones if switch

 Agregar (o eliminar) módulos (clases) mejor que manipular líneas.

# Liskov substitution

 *Si  $S$  es un subtipo de  $T$ , entonces los objetos de tipo  $T$  pueden reemplazarse con objetos de tipo  $S$  sin alterar ninguna de las propiedades deseables del programa.* 

 Herencia en lugar de composición

 Debería poder sustituir una cosa por otra, si se declara que esas cosas se comportan de la misma manera.

# Interface segregation

 *Muchas interfaces específicas son mejores que una interfaz de uso general.*

 Pensar en términos de lo que soy, en lugar de lo que puedo hacer.

 Muestra a tus clientes sólo lo que necesitan ver.

# 🙄 Dependency Inversion

🧙 *Depende de abstracciones, no de concreciones.*

😈 El new

👶 Evita crear directamente tus dependencias.



# Mandamientos

- ⊘ **No abrumes:** toda abstracción tiene un coste.
- ⊘ **No defraudes:** cumple tus contratos.
- ⊘ **No confundas:** mantén los niveles de abstracción.
- ⊘ **No lées:** junta lo que cambian por la misma razón.
- ⊘ **No expolies:** usa lo imprescindible.





# Code Labs

- Repositorio en GitHub
  - <https://github.com/LabsAdemy/CleanCodeAdvanced>
  - Ramas por lección (1, 2...)
- TypeScript
  - JavaScript con tipos; similar a Java y C#
  - Ejemplos síncronos y constructores estilo Java C#
  - Fakes para infraestructura (DB, HTTP y SMTP)
- Edición local
  - git clone <https://github.com/LabsAdemy/CleanCodeAdvanced.git>
  - npm i
  - git checkout 1\_PSD

# Astro Bookings

- Aplicación de reservas para empresa de viajes espaciales
  - <https://github.com/AstroBookings/.github/wiki/Context-Diagram>
  - <https://raw.githubusercontent.com/wiki/AstroBookings/.github/requirements/AstroBookings-event-storm.png>
- Funcionalidad:
  - Solicitar una reserva `bookings.solicite()`
  - Anular una reserva `bookings.annulate()`
  - Ofertar viaje `trips.offer()`
  - Cancela viaje `trips.cancel()`
  - Listar viajes `trips.getList()`

# bookings.solicite()

1. Validar petición
2. Consultar disponibilidad operador
3. Obtener precio
4. Pagar con tarjeta
5. Confirmar reserva al operador
6. Notificar reserva al viajero