

# Tidy First!, Kent Beck

---

ENOL RODRÍGUEZ HEVIA - U0287935

ALFREDO JIROUT CID - U0288443

CARLOS CABRERA MORAL - U0288595



# INTRODUCCIÓN

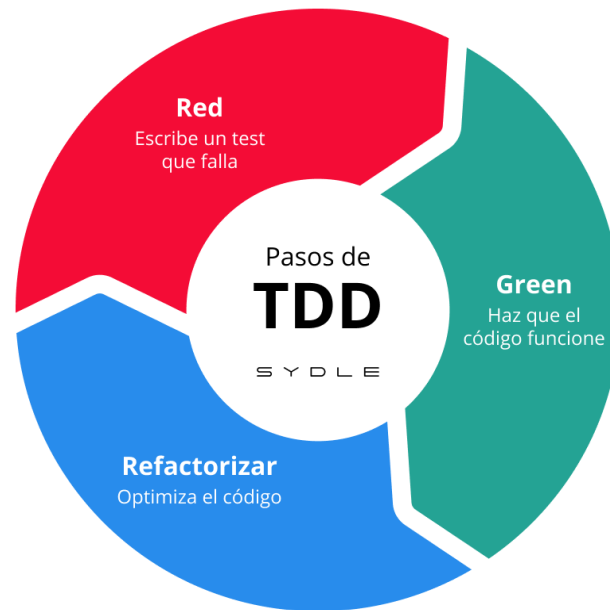
O'REILLY®

## Tidy First?

A Personal Exercise in Empirical Software Design



Kent Beck



MANAGEMENT



# P

Extreme programming

---

# Crítica a la rigidez normativa



Reglas como el número de líneas o de parámetros.



Cuanto mayor sea el sistema, mayor será el código.



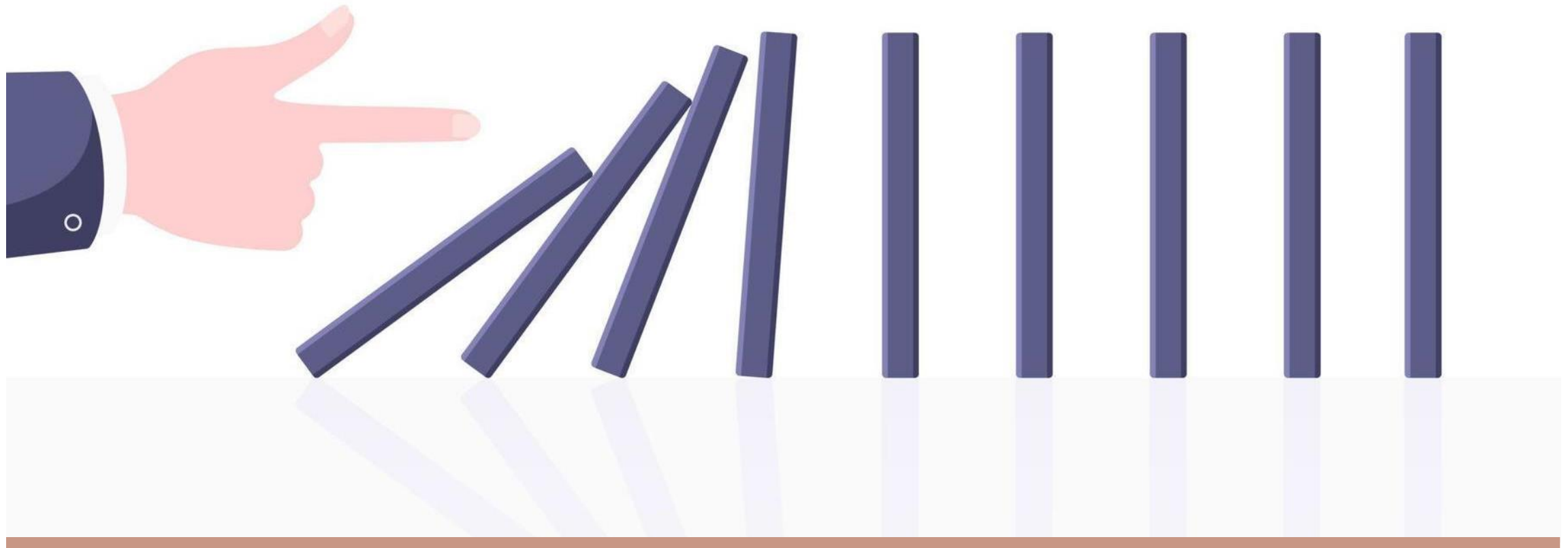
El software es un proceso natural

---

# Conceptos principales de "Tidy First?"

- Acoplamiento
- Cohesión
- Tidying





# Acoplamiento

- Relación entre elementos
- Si cambio un elemento me obliga a cambiar otro
- Cambios en cascada
- Coste elevado



# Cohesión

- Si modifico algo dentro de un elemento, tengo que modificarlo todo
- Juntar elementos acoplados
- Limitamos la propagación del acoplamiento

# Tidying u Organizar



- Escala más pequeña de diseño
- Pequeños cambios estructurales
- Grandes cambios en pequeños pasos seguros
- Should I tidy first? - ¿Debería organizar primero?
- Haz cambios fáciles para hacer fácil el cambio



## CÓDIGO LIMPIO

- Metáfora que sugiere que un software es más un jardín que una escultura perfecta





80% de los cambios que tengas que hacer los vas a tener que hacer sobre 20% del código.

## PRINCIPIO DE PARETO

---

# Grandes refactorizaciones vs Refactorizaciones incrementales

- Nivel de confianza
- Visibilidad de los cambios



---

# Cambios estructurales vs cambios de comportamiento

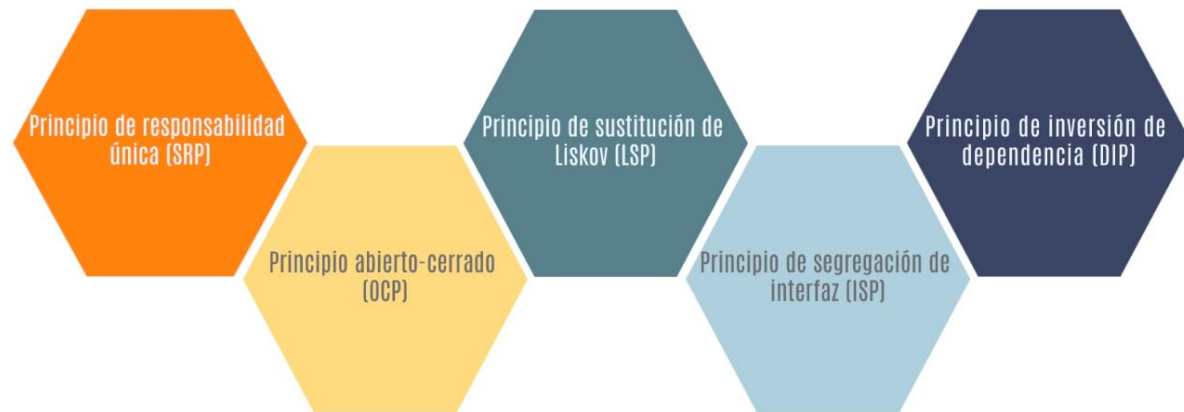
Cambios estructurales	Cambios de comportamiento
Suelen ser reversibles	Pueden tener efectos irreversibles
Reorganizar funciones	Errores en pagos o reportes



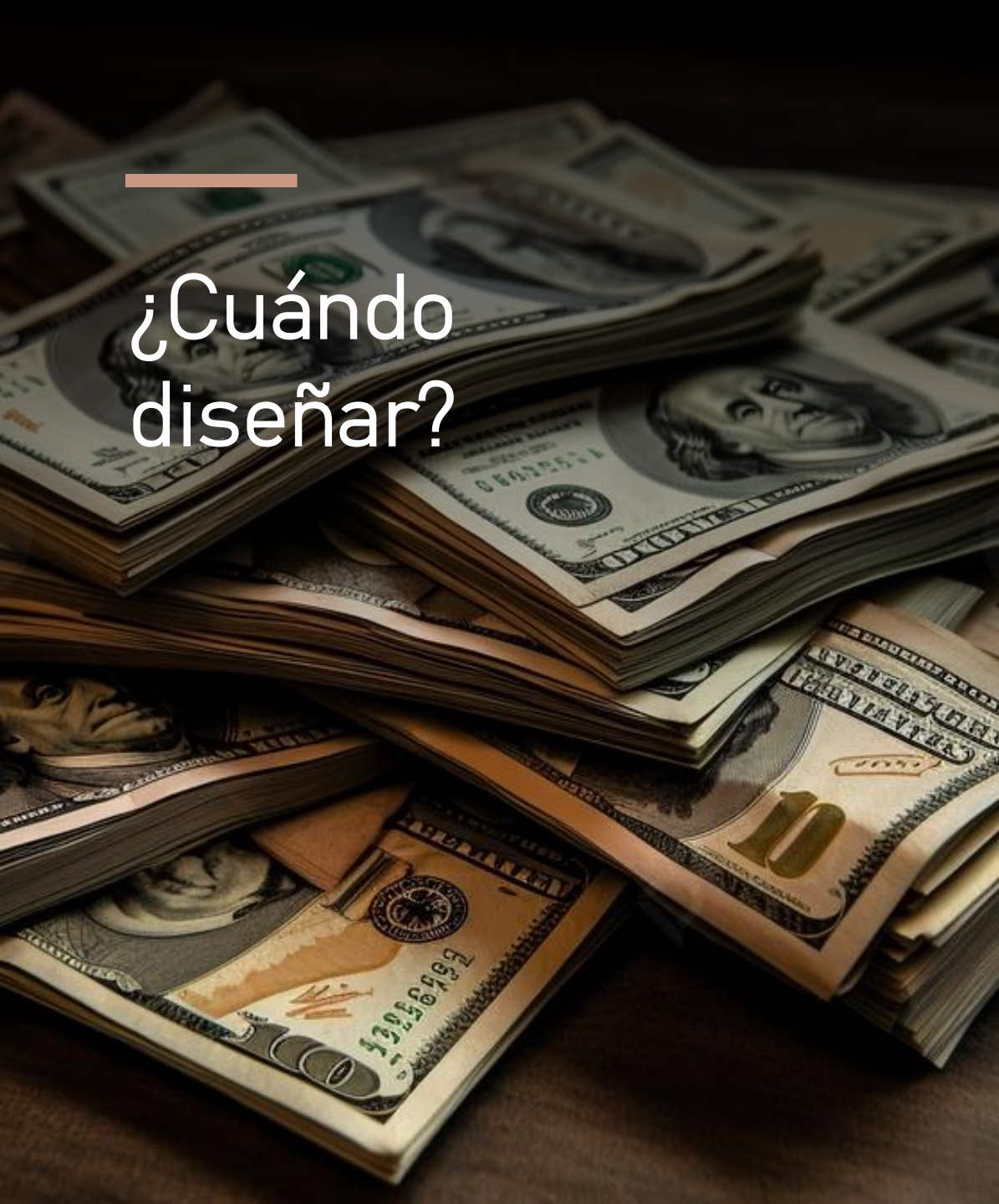
- Puede haber algunos cambios estructurales que pueden ser difíciles de revertir
- Minimizar el impacto con transiciones seguras, como mantener versiones paralelas antes de eliminar funciones antiguas.

# El impacto de los pequeños cambios en la arquitectura y el diseño

S.O.L.I.D.



- Diseño fractal
- Si se pueden realizar cambios estructurales con facilidad, la necesidad de diseño previo disminuye



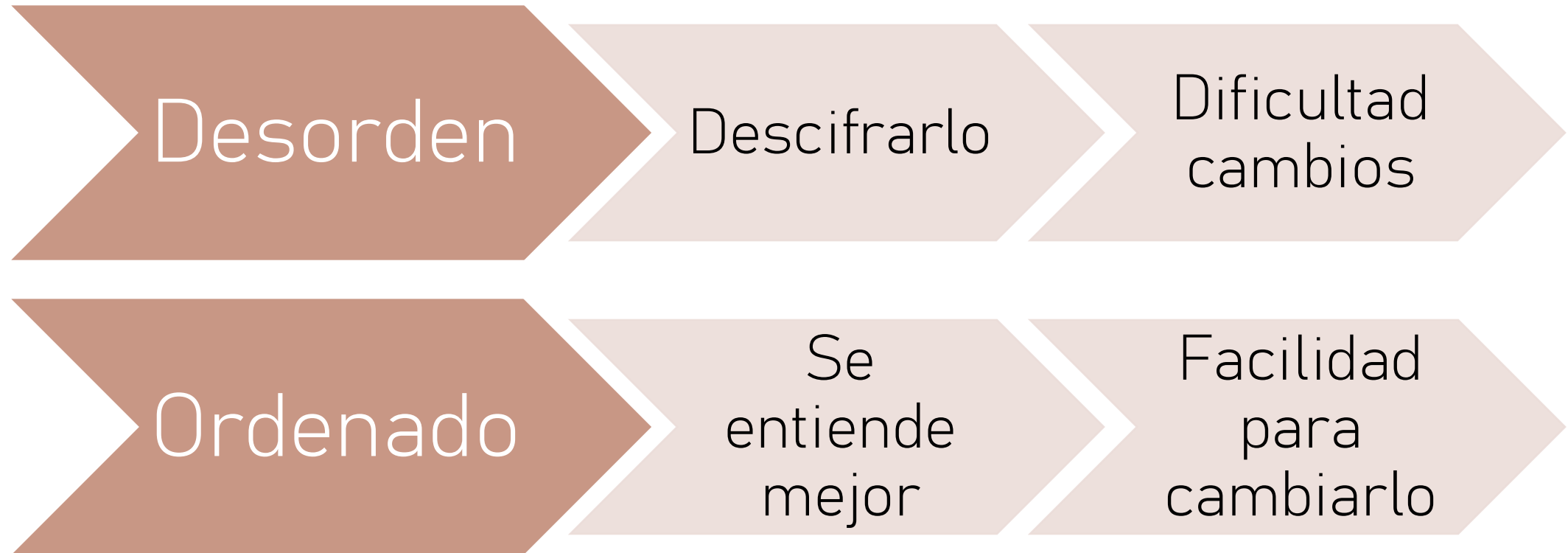
# ¿Cuándo diseñar?

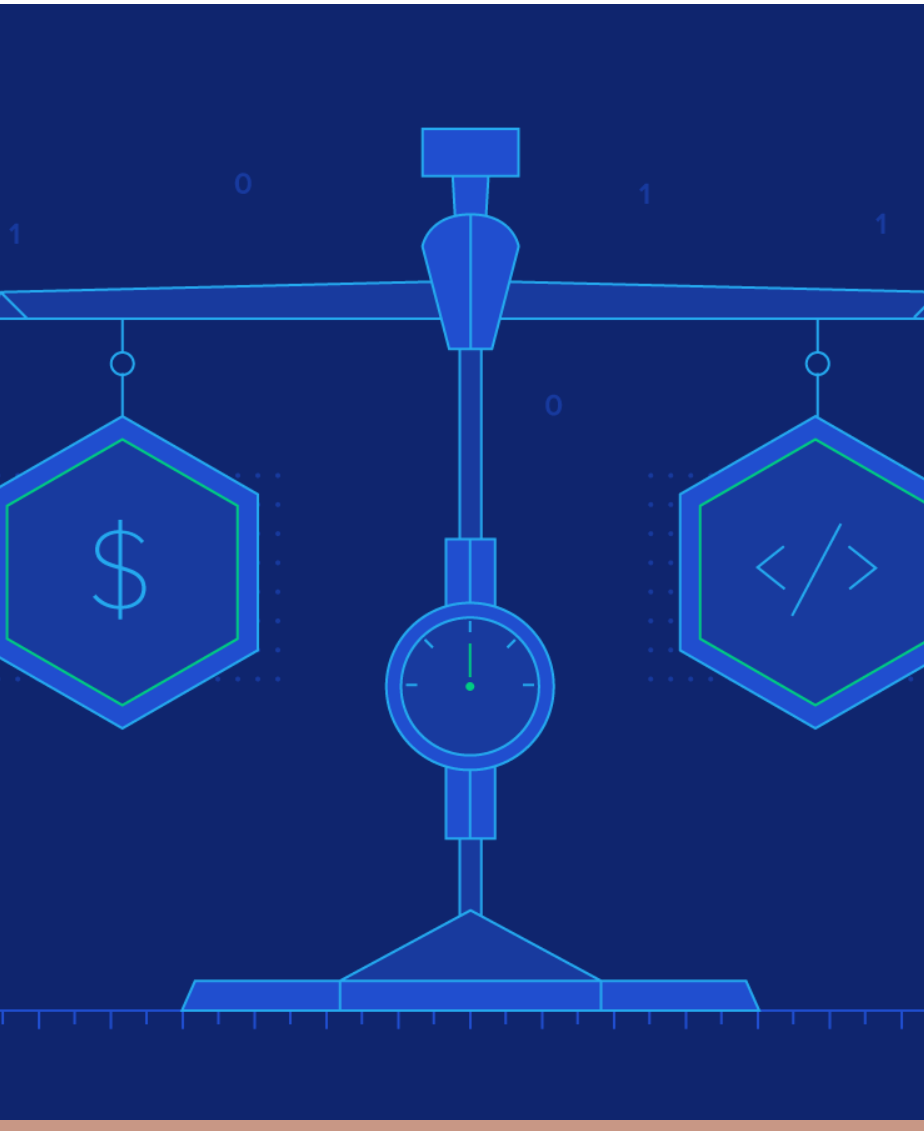
## Diseñar primero:

- Gran Resistencia
- Tiempo de diseño -> no ganamos dinero
- Dinero Temprano -> más dinero
- Tasas de interés altas
- Sin Dinero al principio -> fracaso

---

# ¿Ordenar el código?



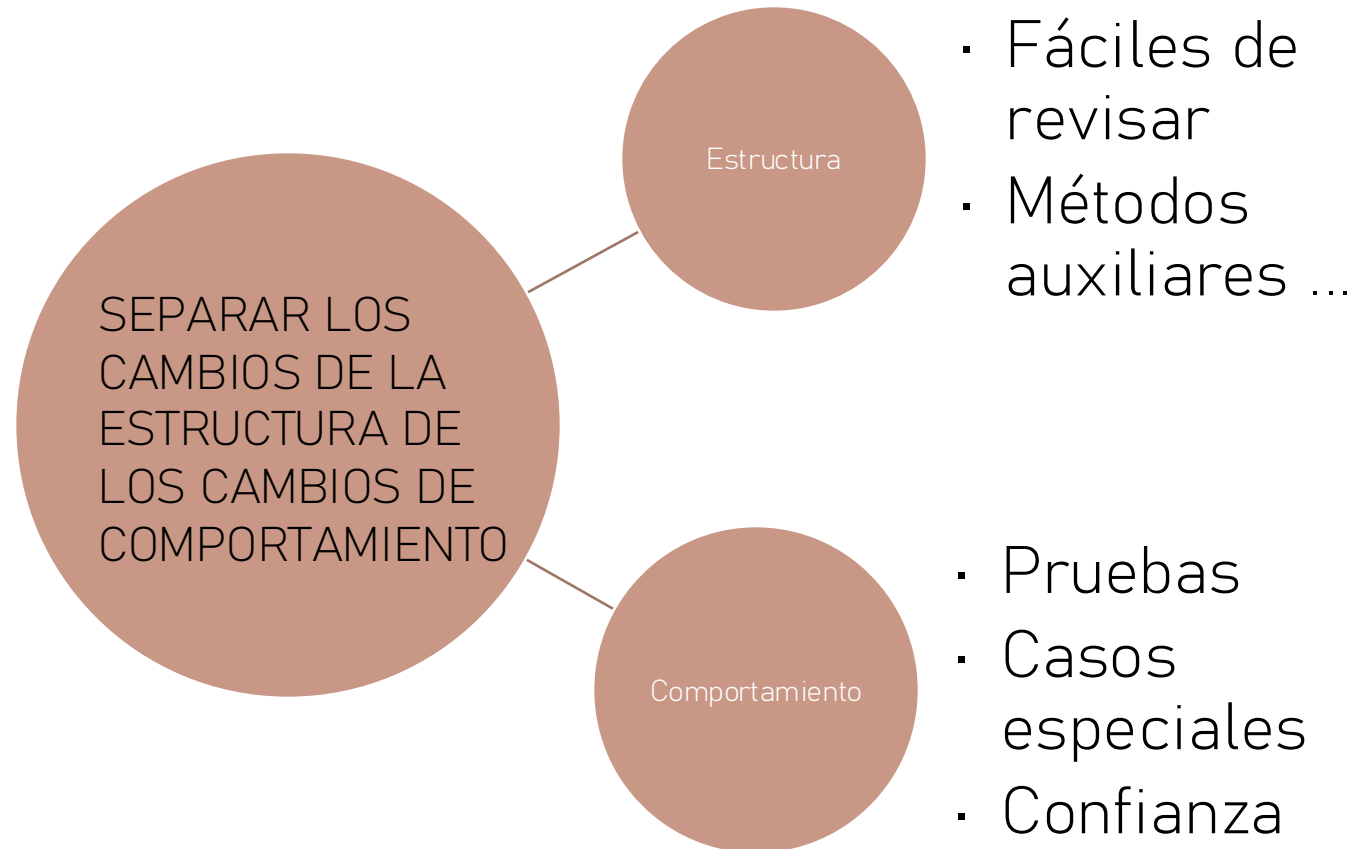


# Costo del software

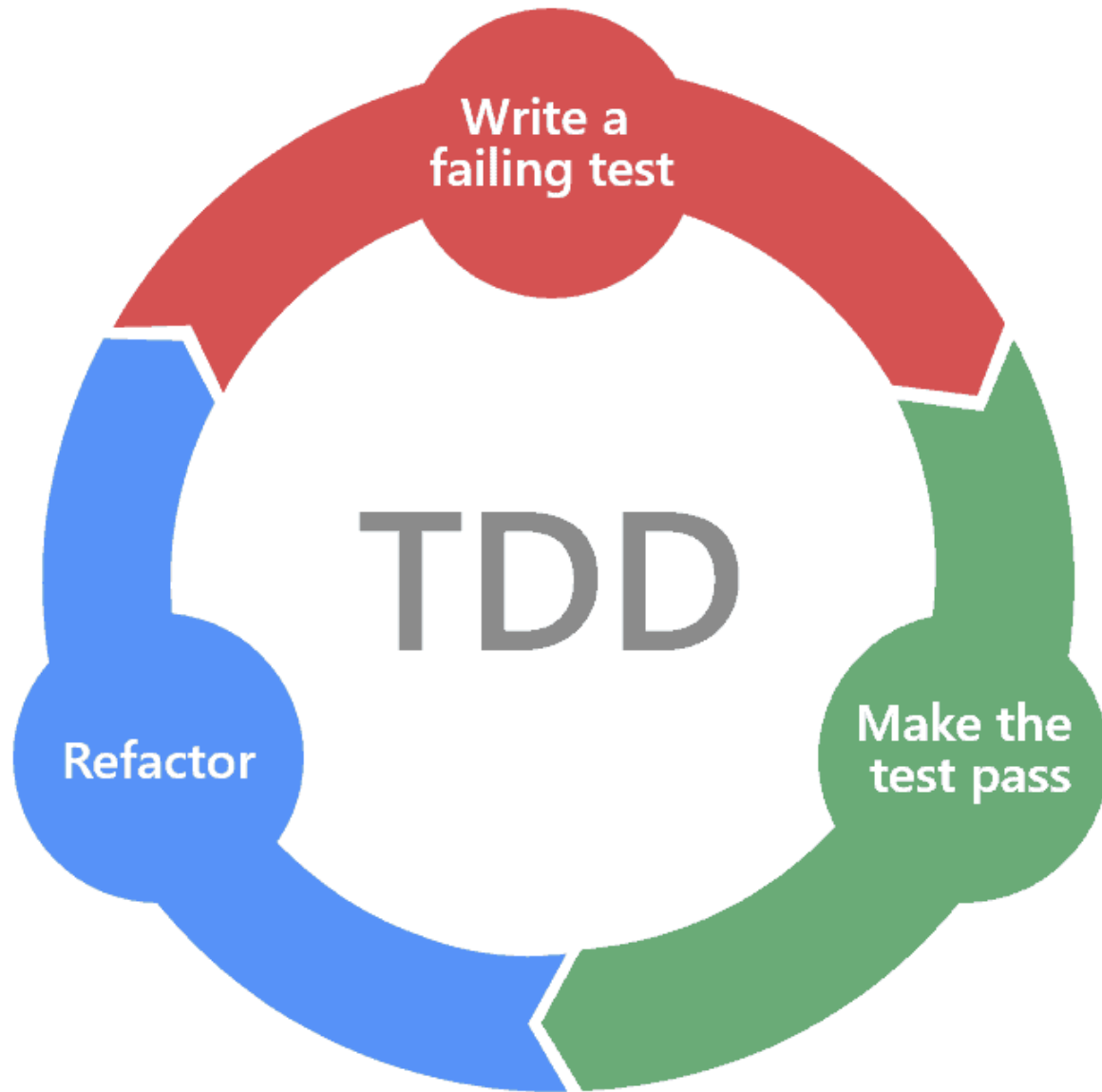


---

# Revisión del código







## TDD

- Mucha críticas por malentendidos
- Confusión con otros enfoques
- Puede ser útil



RONDA DE  
PREGUNTAS