IIC2113 - Diseño Detallado de Software

Antipatrones

Sebastián Butorovic
Benjamín Ibarra
Gerardo Olmos

Definición

Antipatrón

Es una solución que presenta **consecuencias negativas**, para un problema recurrente.

Presenta más problemas que soluciones

Tipos de antipatrones

- Antipatrones de arquitectura
- Antipatrones de gestión de proyectos
- **Antipatrones de diseño de software**
- Antipatrones de diseño orientado a objetos
- Antipatrones de programación
- Antipatrones de gestión de la configuración
- Antipatrones organizacionales
- Antipatrones sociales

Antipatrones de arquitectura

- Reinvent the wheel
- Vendor lock-in
- Stovepipe enterprise

Antipatrones de gestión de proyectos de software

- The mythical month man
- Project miss-management
- Corncob

Antipatrones de diseño de software

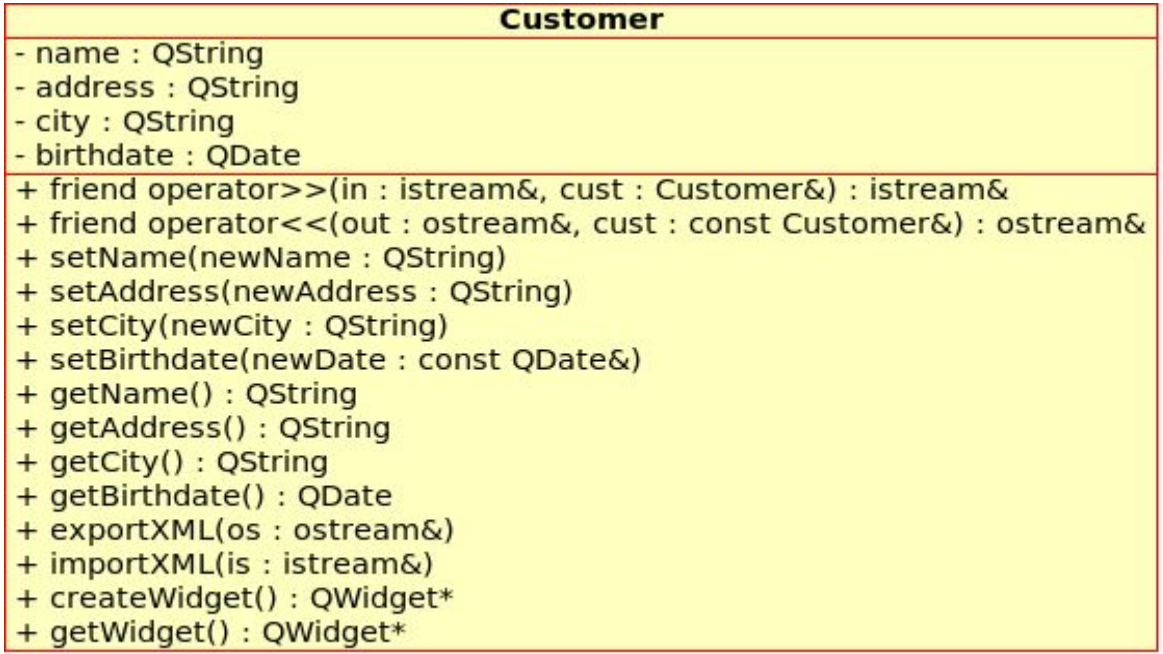
- The God
- Lava flow
- Functional decomposition
- Poltergeist
- Golden hammer
- Spaghetti code
- Copy and paste programming

Antipatrón: The god

Un único módulo o clase que hace completamente todo lo que se desea, código complejo, difícil de leer y de muchas líneas

```
Class main(){

    Def __init__(self, a, b, c, d, e …)

        Self.a = a

        Self.b = b

        …

    Def some_interesting_method()

        What do i do?...
```

Antipatrón: The god



Fuente:
Alan Ezust, Paul Ezust: Introduction to Design Patterns in C++ with Qt, 2011

Antipatrón: Lava Flow

Gran cantidad de código desordenado, poca documentación y claridad.

- Declaración de variable no justificadas
- Clases o bloques de códigos muy grandes
- Inconsistente y difuso estilo de evolución de la arquitectura implementada
- Existen muchas áreas de código incompletas
- Código abandonado

Antipatrón: Functional Decomposition

Clases que son programadas pensando en un paradigma de programación funcional. Como resultado se tienen clases que:

- Tienen todos sus atributos privados
- Tienen una sola funcionalidad, como los métodos
- Diagramas de clase no tienen sentido

Antipatrón: Poltergeist

Clases que tienen funcionalidades tan limitadas que actúan brevemente y luego no vuelven a aparecer dentro de un programa.

- Obstruyen el flujo de las demás clases
- Por lo general pueden ser absorbidas por otra clase

Antipatrón:Golden Hammer

Aferrarse a un paradigma, por ejemplo un framework de programación o lenguaje.

- Soluciones hechas con herramientas más adecuadas suelen ser más optimas
- Se depende de encontrar clientes específicos para nuevos desarrollos

Antipatrón: Spaghetti Code

Código largo y sin una estructura claramente identificable.

- Se pierde tiempo entendiendo el código
- Se genera dependencia del programador que lo programó para entenderlo o extenderlo

Antipatrón: Copy and Paste Programming

Código que se copia en métodos que son similares.

- Uso de código que no se entiende completamente
- Bugs en el código se repiten en las copias

Antipatrón: Copy and Paste Programming

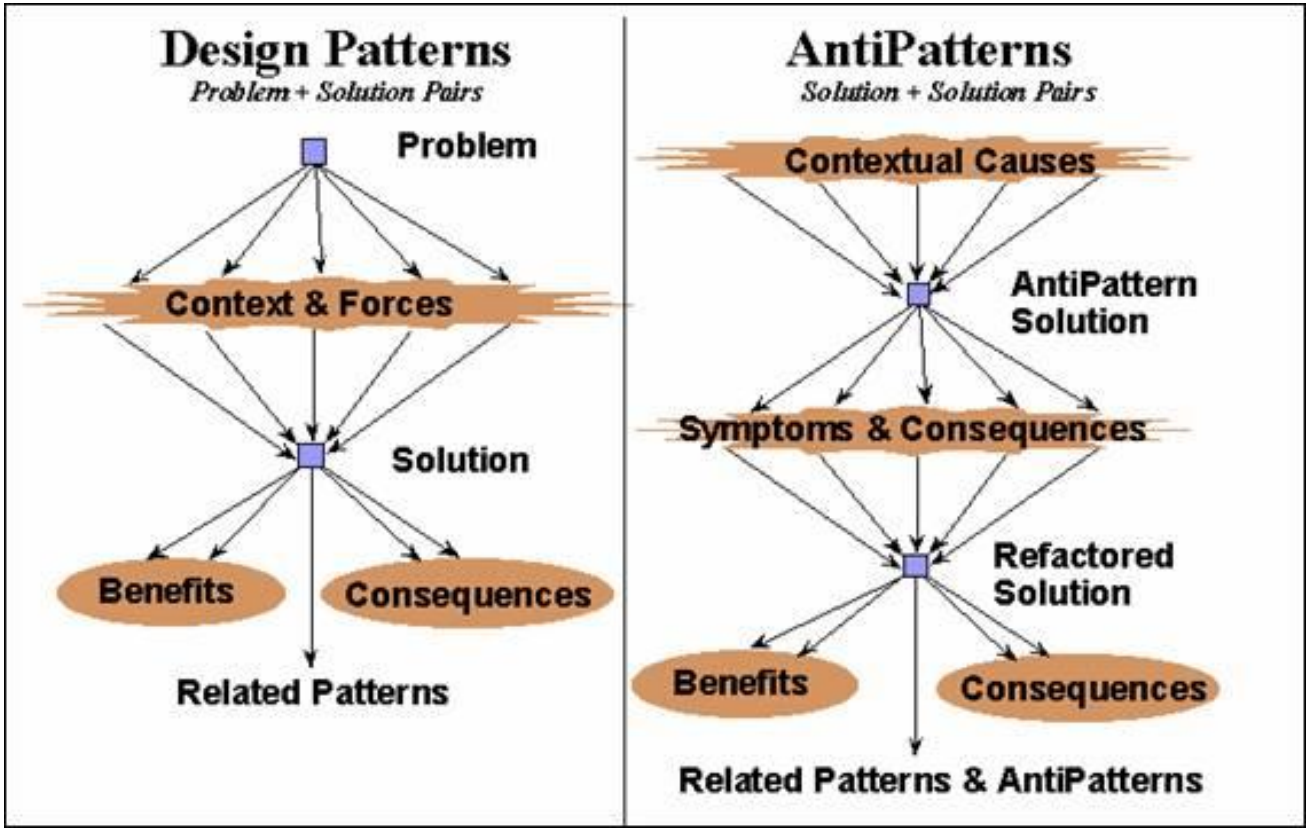
```ruby
# Para aumentar el ataque de un pokémon
def increase_attack(n)
  # Guarda si el pokémon fue encontrado
  f = false
  for pk in @pl
    if pk[0] == n
      pk[1] += 1
      f = true
    end
  end
  if !f
    puts n + " no encontrado"
  end
end
```

```ruby
# Para aumentar la defensa de un pokémon
def increase_defense(n)
  # Guarda si el pokémon fue encontrado
  f = false
  for pk in @pl
    if pk[0] == n
      pk[2] += 1
      f = true
    end
  end
  if !f
    puts n + " no encontrado"
  end
end
```

Antipatrones

No son tan malvados

Los antipatrones también pueden ser útiles para detectar fallas en el software. Explican por qué una solución **pudo ser atractiva** y cómo recuperarse.



Fuente:
McCormick, Hays: *Antipatterns Tutorial*, 1998

Los antipatrones nacen de una estructura distinta a la de los patrones

Conclusión

Los antipatrones son filete