Code Smells

About Me

Gabriel Coronado

Ingeniero de Software con más de 5 años de experiencia causando errores en producción.

Hincha del Stack Overflow FC.

linkedin.com/in/gabo-cs

github.com/gabo-cs

gabo-cs.github.io

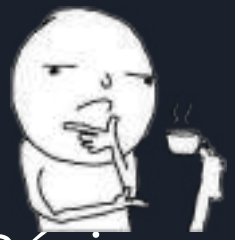
¿Qué son los Code Smells?

Un “code smell” es cualquier característica en el código fuente de un programa que posiblemente indica un problema más profundo.

— What? How can code "smell"??

— Well it doesn't have a nose... but it definitely can stink!

Un “code smell” es un indicador de que algo puede estar mal.



Básicamente, ¿son bugs?

¡No! Los “code smells” no son técnicamente incorrectos y no impiden el funcionamiento del programa.

En cambio, indican debilidades en el diseño que pueden ralentizar el desarrollo o aumentar el riesgo de errores o fallas en el futuro.

Los Code Smells están identificados

Nombre:
Categoría:
Soluciones:

Algunos Code Smells populares:

- Long Method
- Large Class
- Feature Envy
- Shotgun Surgery
- Divergent Change
- Case Statements
- Long Parameter List
- Uncommunicative Name

Categorías

Change Preventers
(Preventores de cambios)

Couplers
(Acopladores)

Bloaters
(“Infladores”)

Object-Orientation Abusers
(Abusadores de la Orientación a Objetos)

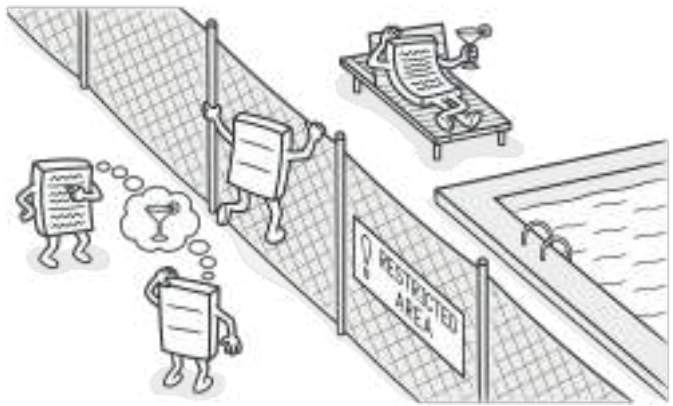
Dispensables

Refactoring Guru

Ejemplo: Feature Envy

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    question = answer.question
    result + question.score(answer.text)
  end
end
```



Este método sufre de feature envy: Hace referencia a *answer* más de lo que hace referencia a métodos o variables de su propia clase.

Ejemplo: Feature Envy

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    question = answer.question
    result + question.score(answer.text)
  end
end
```

Comencemos extrayendo un método:

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    result + score_for_answer(answer)
  end
end

private

def score_for_answer(answer)
  question = answer.question
  question.score(answer.text)
end
```

Ejemplo: Feature Envy

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    question = answer.question
    result + question.score(answer.text)
  end
end
```

Ahora que hemos aislado el feature envy, solucionémoslo moviendo el método:

```ruby
# app/models/answer.rb

def score
  question.score(text)
end
```

Ejemplo:
Feature Envy

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    question = answer.question
    result + question.score(answer.text)
  end
end
```

Antes

```ruby
# app/models/completion.rb

def score
  answers.inject(0) do |result, answer|
    result + answer.score
  end
end


# app/models/answer.rb

def score
  question.score(text)
end
```

Después

Ejemplo (Doble): Case Statements & Shotgun Surgery

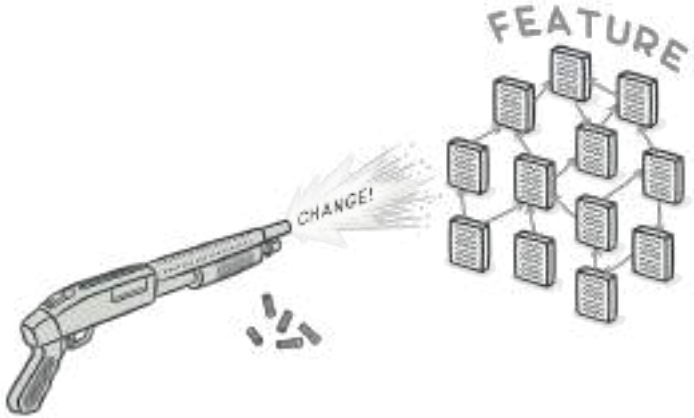
```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    case params[:id]
    when 'breakdown'
      Breakdown.new(user: current_user)
    when 'most_recent'
      MostRecent.new(user: current_user)
    when 'user_answer'
      UserAnswer.new(user: current_user)
    else
      raise "Unknown summary type: #{params[:id]}"
    end
  end
end
```

Los Case/Switch Statements son una señal de que un método contiene mucho conocimiento. Piensa dos veces si de verdad lo necesitas.



Shotgun Surgery ocurre cuando tienes que hacer un mismo cambio en diferentes lugares.

Ejemplo (Doble): Case Statements & Shotgun Surgery

```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    case params[:id]
    when 'breakdown'
      Breakdown.new(user: current_user)
    when 'most_recent'
      MostRecent.new(user: current_user)
    when 'user_answer'
      UserAnswer.new(user: current_user)
    else
      raise "Unknown summary type: #{params[:id]}"
    end
  end
end
```

Todas las clases en *summarizer* se instancian de la misma manera, extraigamos un método:

```ruby
# app/controllers/summaries_controller.rb

def summarizer
  sumarizer_class.new(user: current_user)
end

def sumarizer_class
  case params[:id]
  when 'breakdown'
    Breakdown
  when 'most_recent'
    MostRecent
  when 'user_answer'
    UserAnswer
  else
    raise "Unknown summary type: #{params[:id]}"
  end
end
```

Ejemplo (Doble): Case Statements & Shotgun Surgery

```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    case params[:id]
    when 'breakdown'
      Breakdown.new(user: current_user)
    when 'most_recent'
      MostRecent.new(user: current_user)
    when 'user_answer'
      UserAnswer.new(user: current_user)
    else
      raise "Unknown summary type: #{params[:id]}"
    end
  end
end
```

Convention over Configuration:

```ruby
# app/controllers/summaries_controller.rb

def summarizer
  sumarizer_class.new(user: current_user)
end

def sumarizer_class
  params[:id].classify.constantize
end
```

Muy bonito y todo pero… ojo con las vulnerabilidades .

Ejemplo (Doble): Case Statements & Shotgun Surgery

```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    case params[:id]
    when 'breakdown'
      Breakdown.new(user: current_user)
    when 'most_recent'
      MostRecent.new(user: current_user)
    when 'user_answer'
      UserAnswer.new(user: current_user)
    else
      raise "Unknown summary type: #{params[:id]}"
    end
  end
end
```

Scoping *constantize*.

```ruby
# app/controllers/summaries_controller.rb

def summarizer
  sumarizer_class.new(user: current_user)
end

def sumarizer_class
  "Summarizer::#{params[:id].classify}".constantize
end
```

Ejemplo (Doble): Case Statements & Shotgun Surgery

```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    case params[:id]
    when 'breakdown'
      Breakdown.new(user: current_user)
    when 'most_recent'
      MostRecent.new(user: current_user)
    when 'user_answer'
      UserAnswer.new(user: current_user)
    else
      raise "Unknown summary type: #{params[:id]}"
    end
  end
end
```

Antes

```ruby
# app/controllers/summaries_controller.rb

class SummariesController < ApplicationController
  def show
    @survey = Survey.find(params[:survey_id])
    @summaries = @survey.summarize(summarizer)
  end

  private

  def summarizer
    sumarizer_class.new(user: current_user)
  end

  def sumarizer_class
    "Summarizer::#{params[:id].classify}".constantize
  end
end
```

Después

¿Cómo eliminarlos?

Code Smell	Principles it violates	Solutions (Refactoring Recipes)
Large Class	- **S**OLID: Single Responsibility Principle - Convention over Configuration	- Move Method - Extract Class - Extract value object - Replace conditional with polymorphism - Replace subclasses with strategies
Feature Envy	- Law of Demeter - Tell, don't ask	- Extract Method - Move Method - Inline class
Shotgun Surgery	- DRY - SOLI**D**: Dependency Inversion Principle	- Replace conditional with polymorphism - Extract Decorator - Replace conditional with Null Object - Introduce parameter object

¿Cómo encontrarlos?

Aparte de leer sobre ellos, conocerlos, y tenerlos presente:

Gems:

- Reek
- Flog
- RubyCritic


Hosted Services:

- Code Climate
- SonarQube


Don’t Live with Broken Windows

The Pragmatic Programmer

Referencias

- ***Ruby Science.*** Thoughtbot, Ferris, J., & Ward, H. (2014). Thoughtbot.
- ***The Pragmatic Programmer***. Thomas, D., & Hunt, A. (2019). Addison-Wesley Professional.
- *Code Smells* - Refactoring Guru. https://refactoring.guru/refactoring/smells
- *Code Smell* - Wikipedia. https://en.wikipedia.org/wiki/Code_smell
- RailsConf 2016 - Get a Whiff of This. Sandi Metz. https://www.youtube.com/watch?v=PJjHfa5yxlU

```
require 'thanks'


Thanks.call('es')
=> "¡Muchas Gracias!"


Thanks.call('en')
=> "Thank you so much!"
```