Developing a Nose for Code-Smells

Anibal Velarde
Sr. Solutions Architect @ Aon

Hi I'm Anibal

- Anibal Velarde (like **Hannibal** without the "H" sound)
- You can find me
  - On Twitter: @anibalvelarde
  - On GitHub: https://www.github.com/anibalvelarde
  - On Linked In: https://www.linkedin.com/in/anibalvelarde

Up ahead…

- Why is important to get this right
- The dreadful code reviews
- Tools that can help us identify
  - – Identify / Classify Code smells
  - – Apply Patterns for Refactoring
  - – Bonus tools: “practice makes perfect”

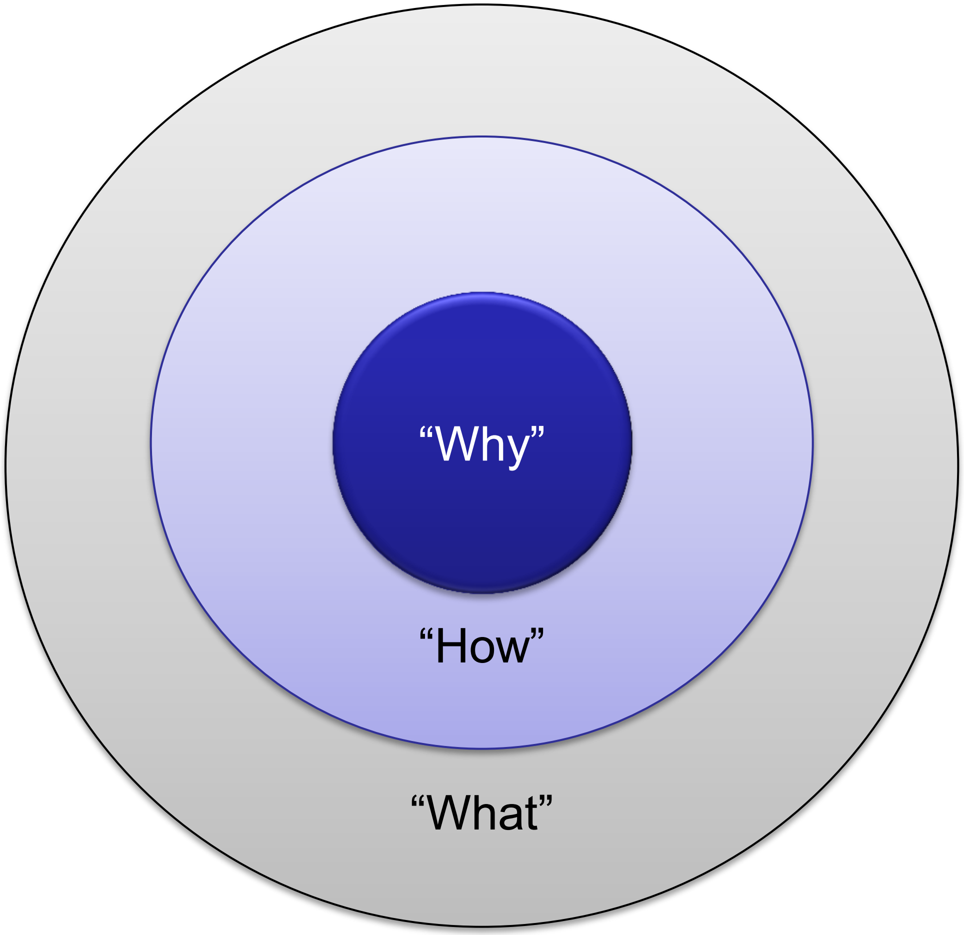
Notice: All cool imagery is courtesy of **refactoring.guru**

Define your “why”…

The Golden Circle:

- “What” you do
- “How” you do it
- “Why”
- Communicate from the inside out!

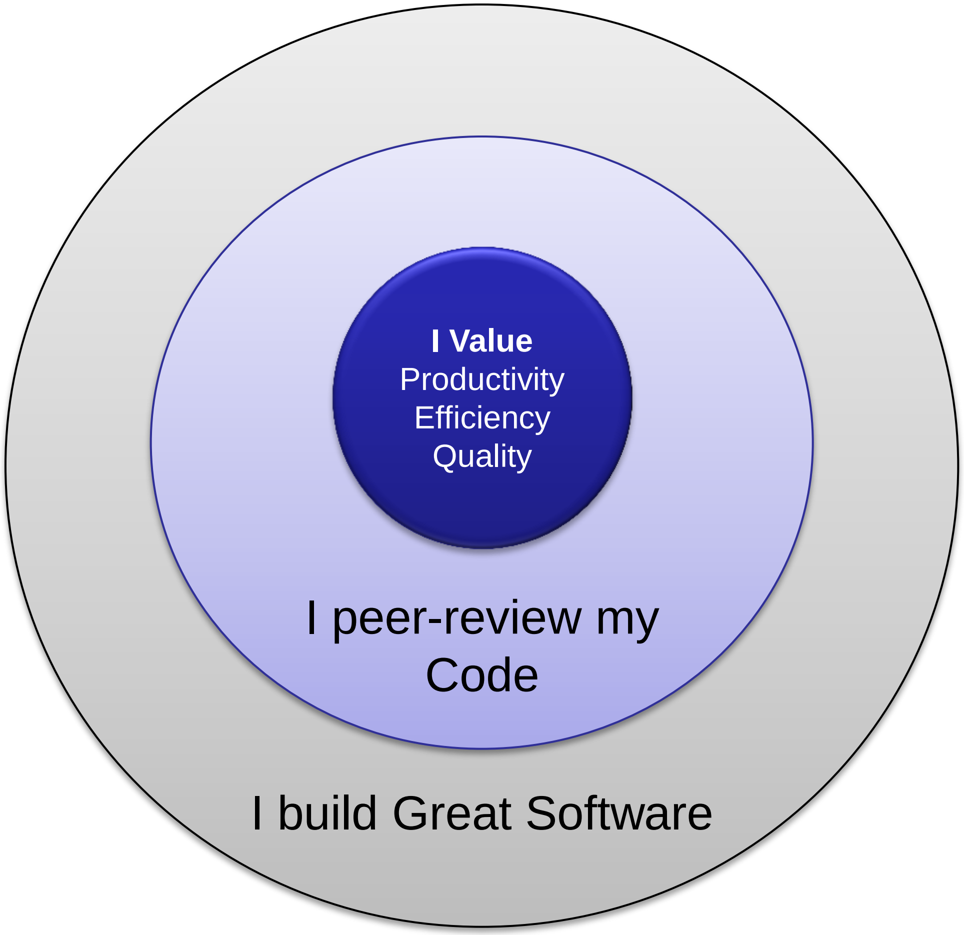
By Simon Sinek



Define your "why"…

Be Ready to Explain Why:

- Articulate you motivation
- Mentor others
- Inspire excellence

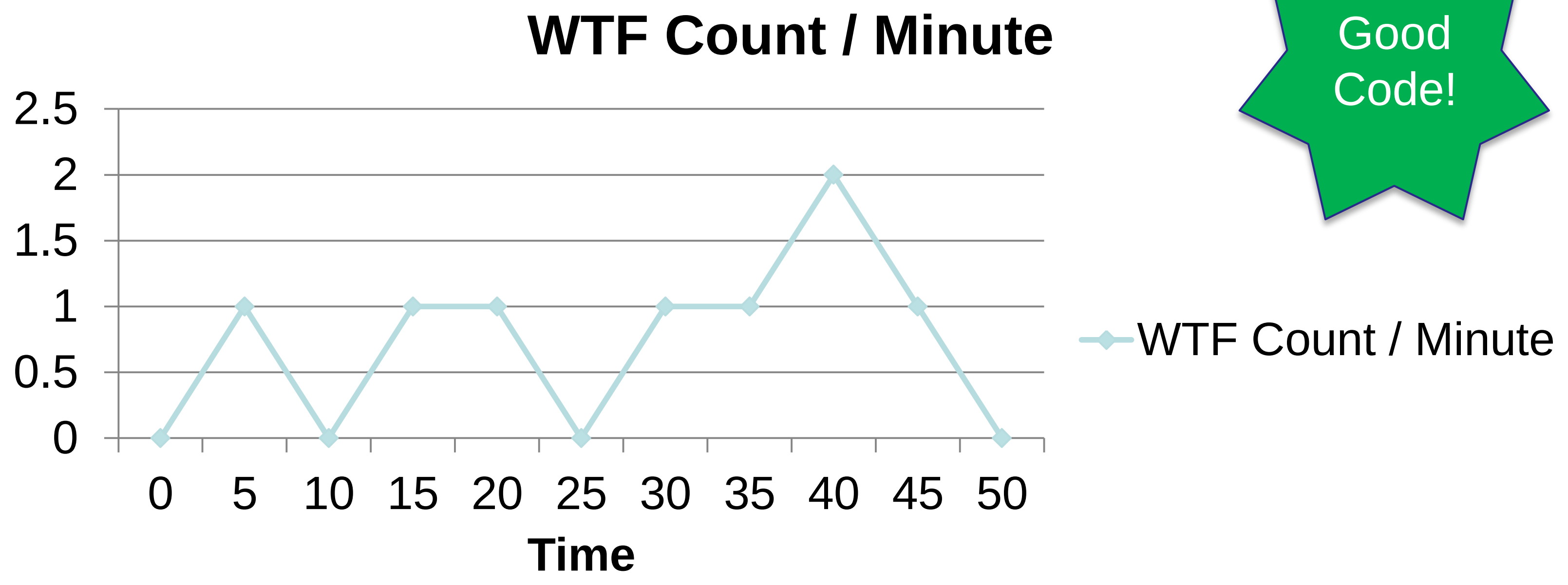


My Feelings Towards Code Reviews

- I don’t do them as often as I should
- I want to get better at it
- Could not find the time
- Everyone does it their way
- I don’t want to insult you but…
- I have no idea what this code is doing
- I’ve never used that C# keyword

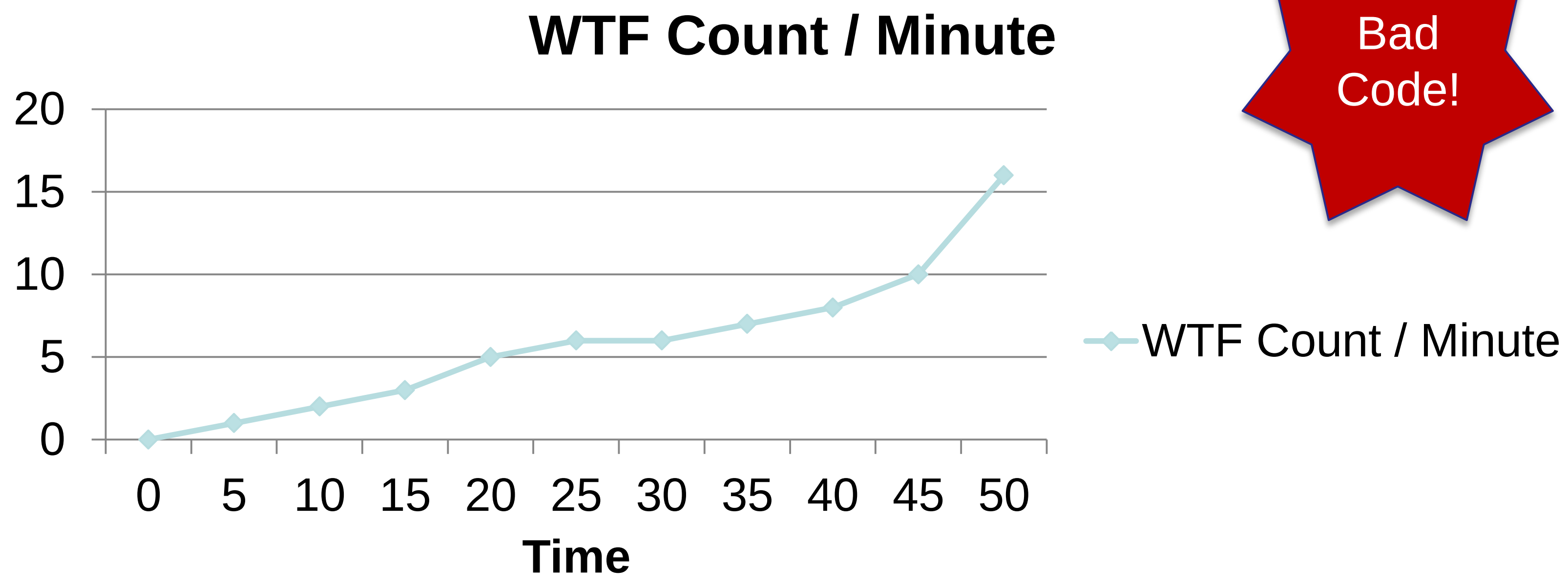
Characteristics of Good Code

- What is the best way to measure code quality during a code review?



Characteristics of Good Code

- What is the best way to measure code quality during a code review?



Dealing with Toxic Levels of Smelly Code

- One approach
  - Ridicule the offender to the point that they would not want to use any keyboard ever again (not recommended)
- Another approach
  - Build and edify your team (even if it is one person at a time)

Tour of Refactoring Guru

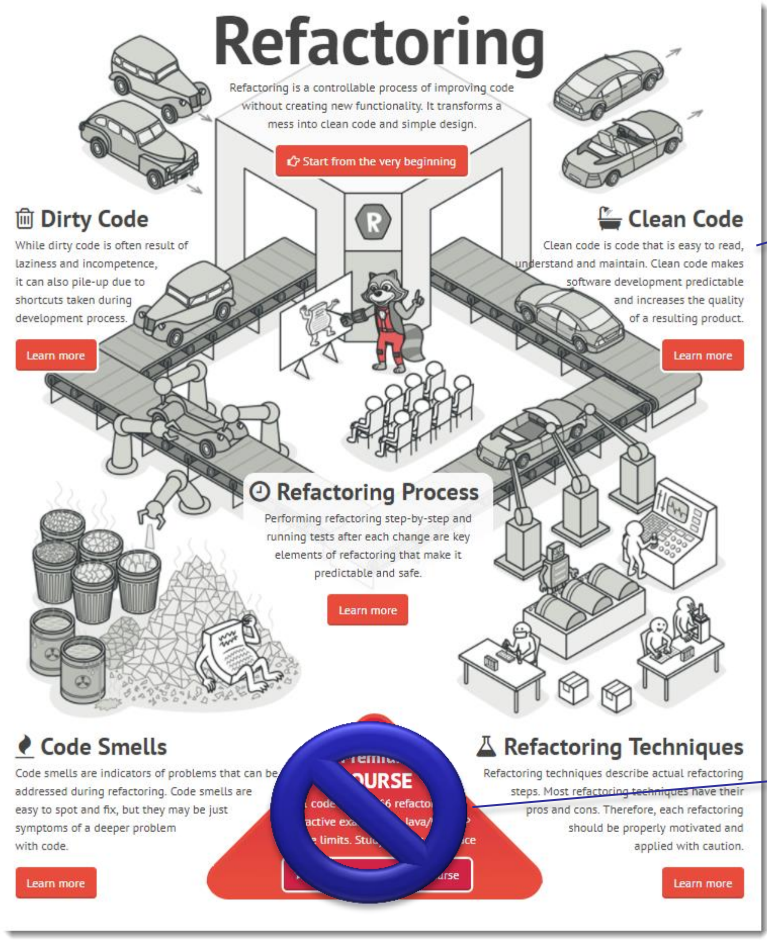
- It is free (you can also purchase)
- It is informative
- It is useful
- It can be shared with participants during code reviews
- From the site:
  - Refactoring.Guru is a shiny website where you can find tons of information on code-smells refactoring, design patterns, SOLID principles and other smart programming topics.

Tour of Refactoring Guru

How to ID bad code?

How to Fix bad code?

It’s free!



Refactoring

Refactoring is a controllable process of improving code without creating new functionality. It transforms a mess into clean code and simple design.

Start from the very beginning

Dirty Code

While dirty code is often result of laziness and incompetence, it can also pile-up due to shortcuts taken during development process.

Learn more

Clean Code

Clean code is code that is easy to read, understand and maintain. Clean code makes software development predictable and increases the quality of a resulting product.

Learn more

Refactoring Process

Performing refactoring step-by-step and running tests after each change are key elements of refactoring that make it predictable and safe.

Learn more

Code Smells

Code smells are indicators of problems that can be addressed during refactoring. Code smells are easy to spot and fix, but they may be just symptoms of a deeper problem with code.

Learn more

Premiu
OURSE
code 66 refacto
active exa Java/
limits. Stu ce
urse

Refactoring Techniques

Refactoring techniques describe actual refactoring steps. Most refactoring techniques have their pros and cons. Therefore, each refactoring should be properly motivated and applied with caution.

Learn more

Code Smell Categories

- Bloaters
- OO Abusers
- Change Preventers
- Dispensables
- Couplers

Let’s dive into a couple…

- Bloaters
- OO Abusers
- Change Preventers
- Dispensables
- Couplers

Before we dive in…

- A word about
  - **Unit Testing**
  - **Integration Testing**

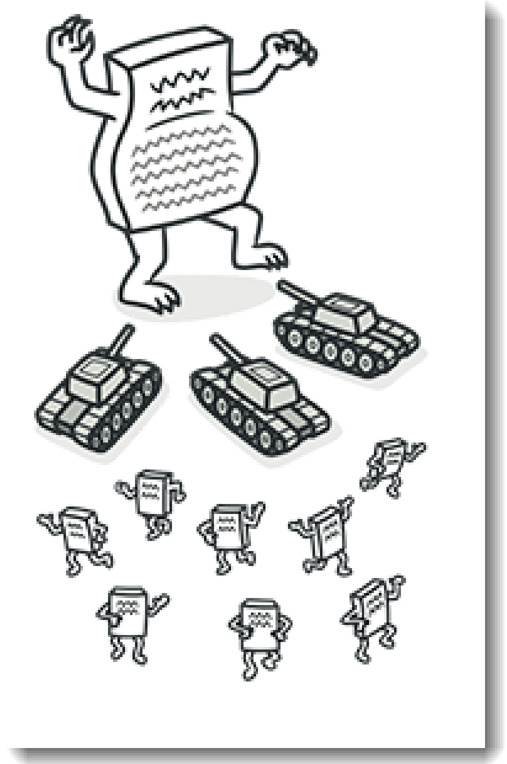
"Good design is testable. Code that isn't testable is badly designed."

Michael Fathers. **Working Effectively With Legacy Code**.

Bloaters…

These are classes, methods or areas of the code that have gotten **HUGE** over time. They come in various categories

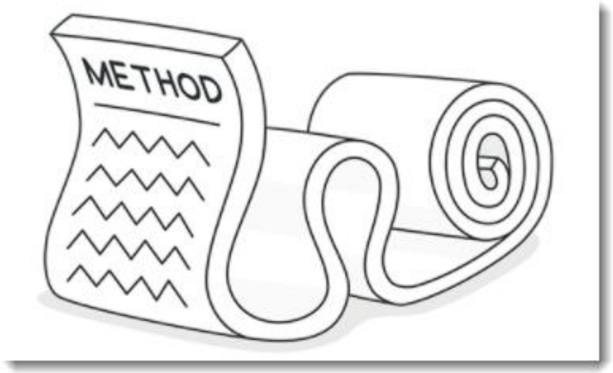
- Long Method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps



Bloaters

Long Method

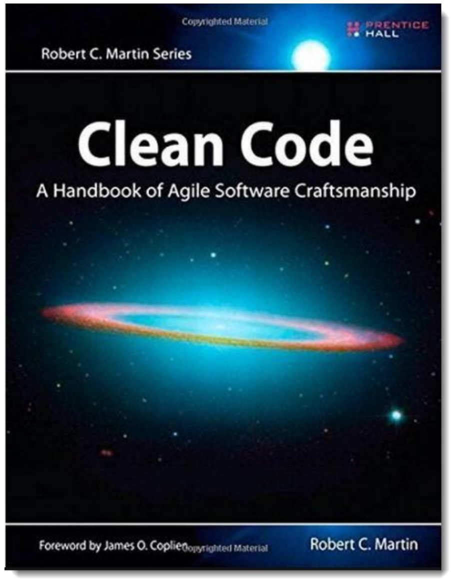
- **Symptoms**
  - Too many lines of code
  - [10 – 25] > you should ask ?s
- **Reasons**
  - Ppl just keep on piling stuff in there!
  - Never is anything taken out!
  - No unit testing to speak of
- **Treatment**
  - Use “**Extract Method**” pattern to reduce the length of the body. If you comment something, consider putting that logic in another method
  - Other patterns: “**Replace Temp with Query**”, “**Introduce Parameter Object**”



Jump over to VS

- Take a moment to discuss the idea of “Clean Code” with an example

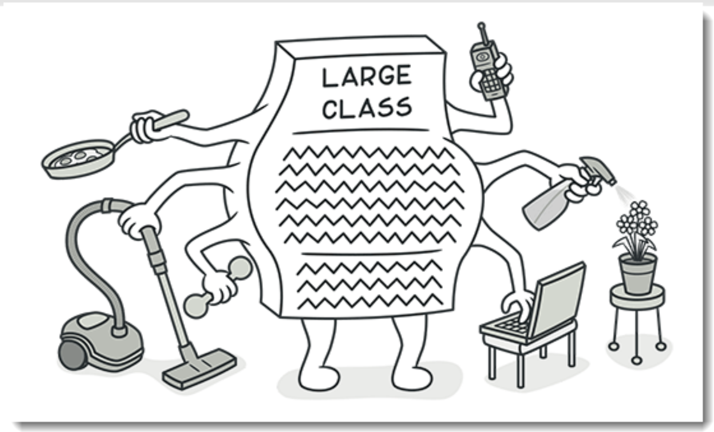
- By Robert C. Martin

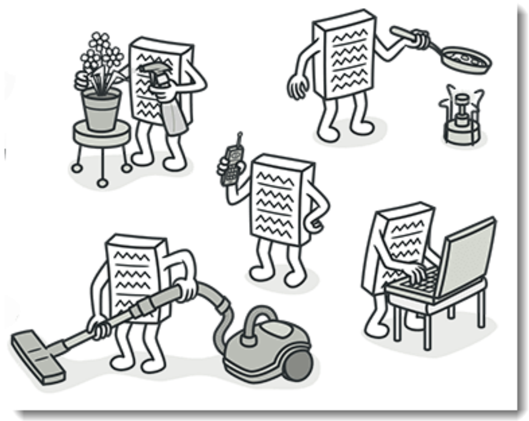


Bloaters

Large Class

- **Symptoms**
  - Class has many
    - Fields
    - Methods
    - LoC
- **Reasons**
  - Even small classes… tend get large over time
  - Similar to "Long Method" things are never taken out
- **Treatment**
  - Use "**Extract Class**" pattern to reduce the length of the class.
  - Other patterns: "**Extract Subclass**", "**Extract Interface**"

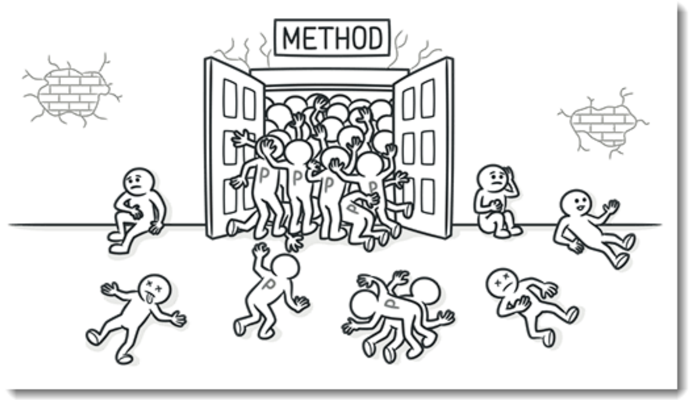


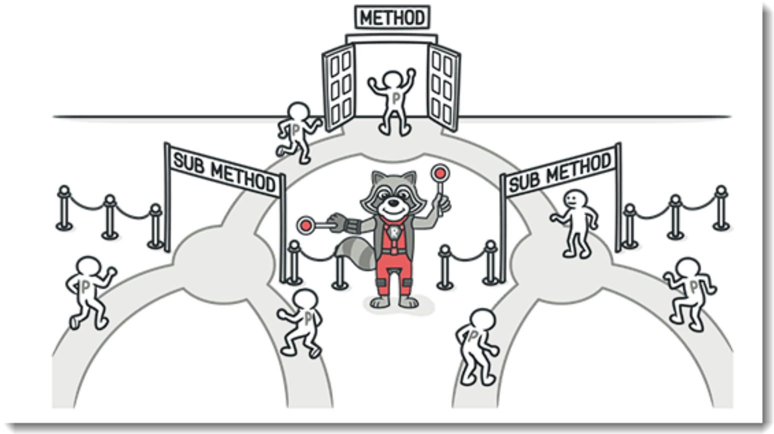


Bloaters

Long Parameter List

- **Symptoms**
  - More than 3 or 4 parameters for a method
- **Reasons**
  - Method is responsible for multiple logic paths
  - Logic for creating dependencies is with caller
- **Treatment**
  - Use “Replace Parameter with Method Call”
  - Try “Preserve Whole Object”
  - Try “Introduce Parameter Object”

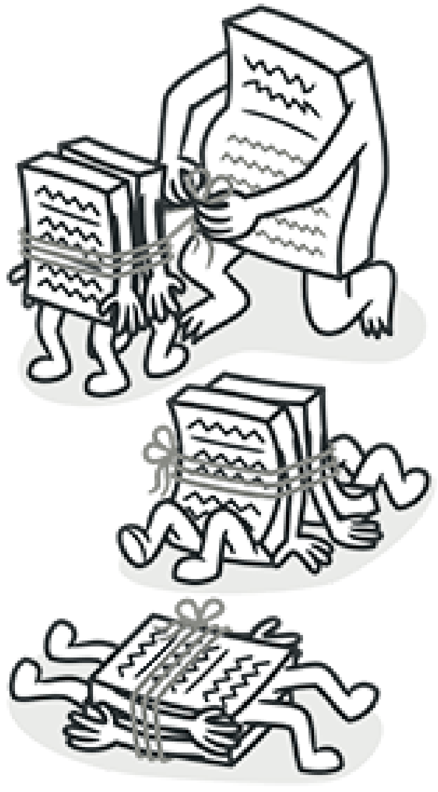




Couplers…

This kind of smell occurs when there is excessive coupling between classes. They come in various categories

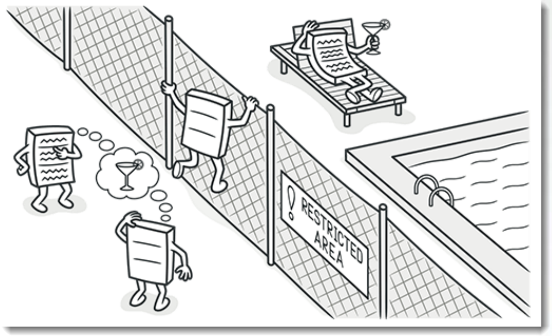
- Feature Envy
- Inappropriate Intimacy
- Message Chains
- Middle Man



Couplers

Feature Envy

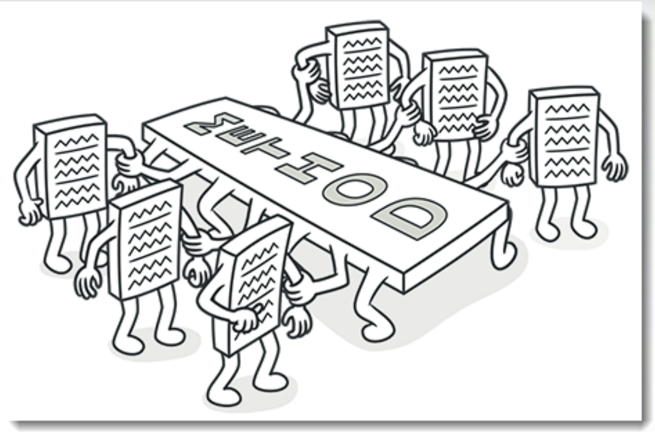
- **Symptoms**
  - A method accesses data of another object more than its own data
- **Reasons**
  - Fields in object "A" are moved to a Data Class
- **Treatment**
  - Some times you'll realize that the envious method should be in the other class. For that case use: **Move Method**
  - For some cases you may also consider using **Extract Method**



Couplers

Inappropriate Intimacy

- **Symptoms**
  - One class uses the internal fields and methods of another class.
- **Reasons**
  - Cohesion decreases within a class instead of increase
  - Single responsibility is not guarded
- **Treatment**
  - The simplest is to “**Move Method**” or “**Move Field**” to the class where the data / behavior is needed
  - Other options include: “**Extract Class**” or “**Inheritance**”



- Switch to Refactoring Guru for walk-thru of some scenarios

- https://www.refactoring.guru

- Bonus: Practice Makes Perfect
- https://www.codewars.com

Thanks for Watching!

- Questions?
- Comments?
- Do you use other tools?
- How do you deal with smelly code?
- Please, share your thoughts