On the trail of Code Smells

Budapest, 29. April 2021
Lapos Zsófia

„IF it stinks change it.” – Martin Fowler

Content

- Definition
- Bad smells
- Refactoring
- Conclusion

Definition

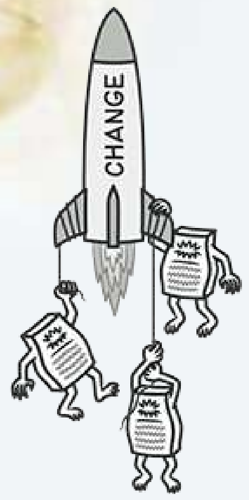
What are code smells?????

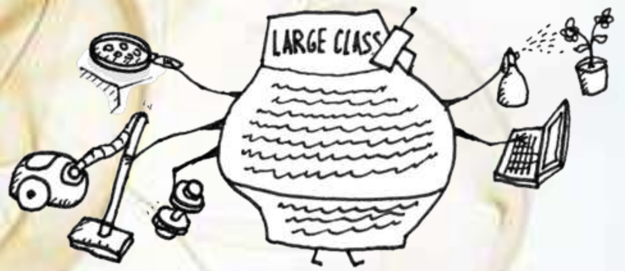
The term was popularized by Kent Beck in the late 90s and its usage increased after appearing in the well known Martin Fowler's book ***Refactoring*.**

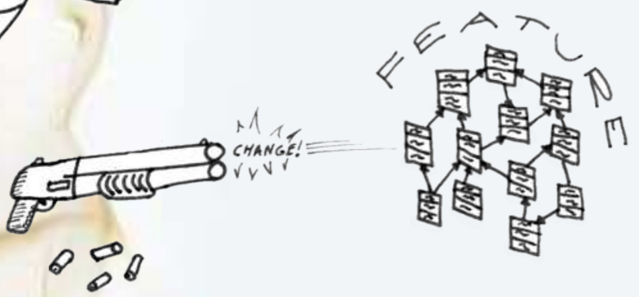
Code Smells are indicators that something **may** be wrong in a piece of code. They are not a problem just because they are a smell, a deeper analysis is needed to determine if there is a problem or not.

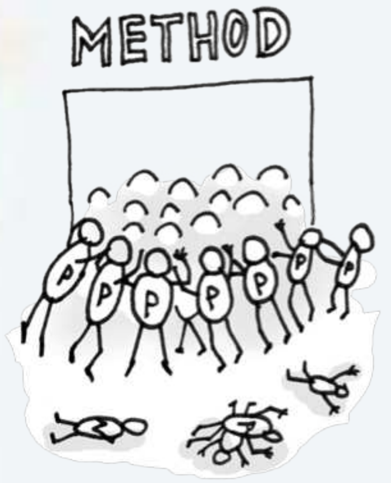
Bad Smells: Classification

- Class/Method organisation
  - Large Class, Long Method, Long Parameter List, Lazy Class, Data Class, …
- Lack of loose Coupling and Cohesion
  - Inappropriate Intimacy, Feature Envy, Data Clumps, Shotgun Surgery, …
- Too much or too little delegation
  - Message Chains, Middle Man, …
- Non Object-Oriented control or date structures
  - Switch Statements, Primitive Obsession, …
- Other confusing Smells:
  - Comments, Code Duplication, Dead Code …







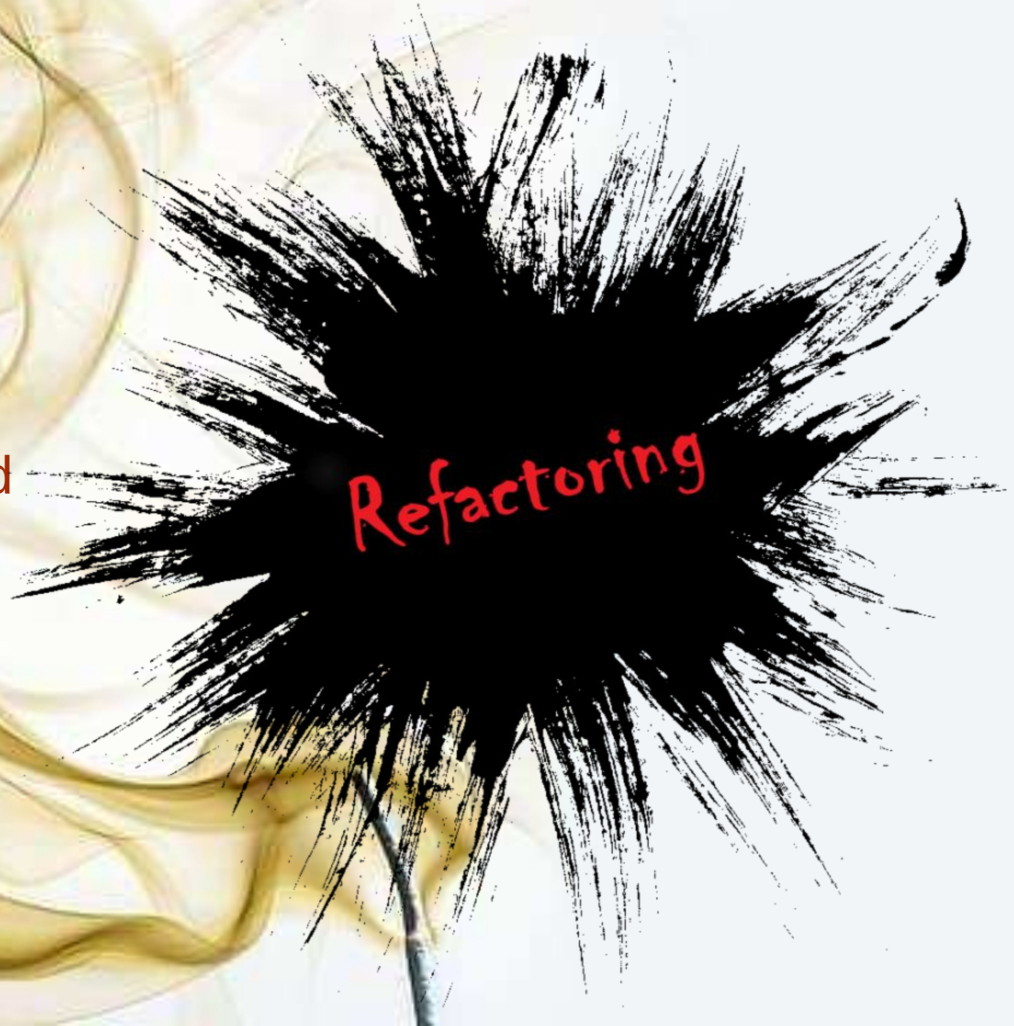


Cartoons borrowed from: https://sourcemaking.com/refactoring/smells

- ✓ They are not bugs
- ✓ They are not technically incorrect
- ✓ They do not block a working program

- ➢ Indicators of weaknesses in design
- ➢ Indicators of slow development
- ➢ Indicators of high risk of introducing new bugs and errors

It’s time to refactor!

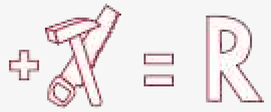


Refactoring

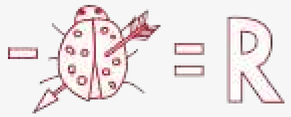
Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet **improves** its internal structure.

When to refactor ?

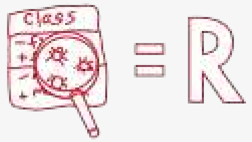
- when adding a feature



- when fixing a bug



- during a code review



Cartoons borrowed from: https://sourcemaking.com/refactoring/smells

The main Refactoring Techniques

- Red-Green-Refactor → used in TDD
- Refactoring by Abstraction
- Composing Method
- Simplifying Methods
- Moving Features Between Objects
- Preparatory Refactoring → This is done when a developer notices the need for refactoring while adding a new feature

Encapsulated Field

Generalize Type

Pull Up Field/Method

Push Down Field/Method

Inline Class/Func./Variable

Extractions(Class, Method…)

Remove Middle Man

Replace Control Flag with Break

Replace Conditional with Polymorphism

Introduce Assertion

Replace Inline Code with Function Call

Move Field/Function/Method

Remove Dead Code

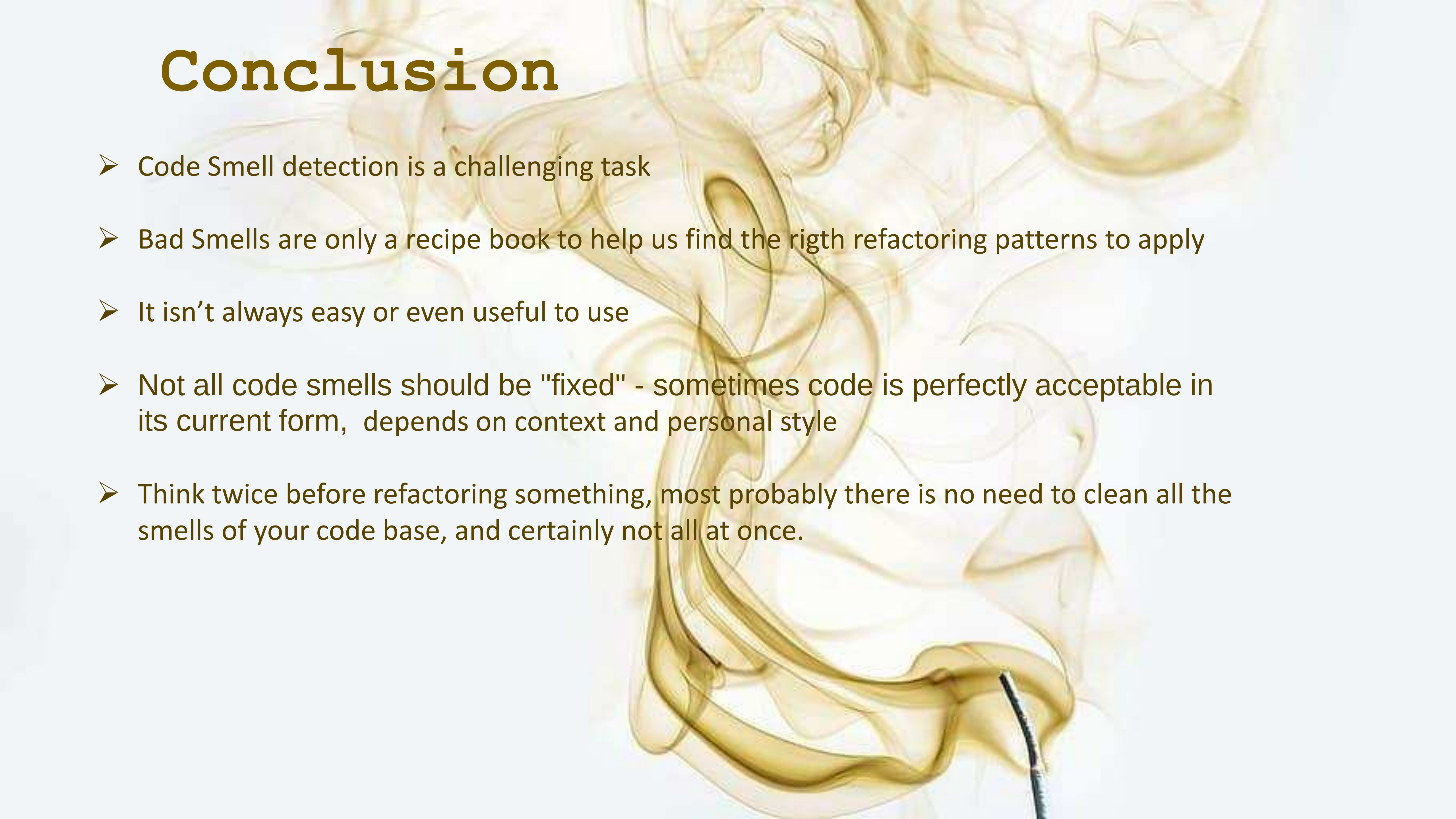
…and others…

My experience with Sonar

- I think that most of the Code Smells in our Sonar System are not real code smells like the ones we learned about here

- They are mainly Clear Code cases:

    - Unused variables
    - Unused imports
    - Commented code
    - Expected { after „if” condition
    - Unreachable code
    - Missing semicolon
    - Not immediatly returned value
    - …..

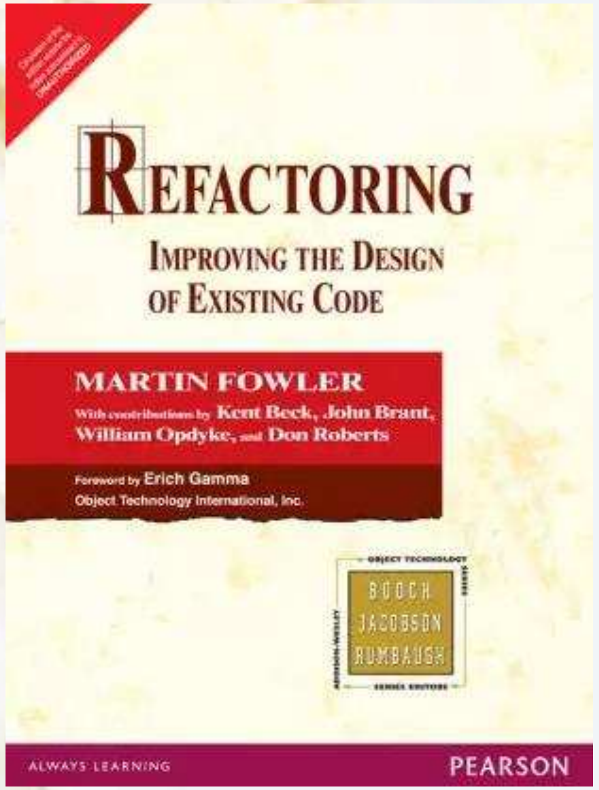


Conclusion

- Code Smell detection is a challenging task
- Bad Smells are only a recipe book to help us find the rigth refactoring patterns to apply
- It isn’t always easy or even useful to use
- Not all code smells should be "fixed" - sometimes code is perfectly acceptable in its current form, depends on context and personal style
- Think twice before refactoring something, most probably there is no need to clean all the smells of your code base, and certainly not all at once.

References:

M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts: **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, 1999.
A quick intro to code smells
https://dev.to/jmir17/a-quick-intro-to-code-smells-3eie
Bad Code Smells - Course „Software Maintenance and Evolution”, given by Prof. Kim Mens at UCL, Belgium
https://www.slideshare.net/kim.mens/bad-code-smells
Refactoring Catalog - Martin Fowler
https://refactoring.com/catalog/


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH
ALWAYS LEARNING
PEARSON

Thank you for your attention!