KU LEUVEN

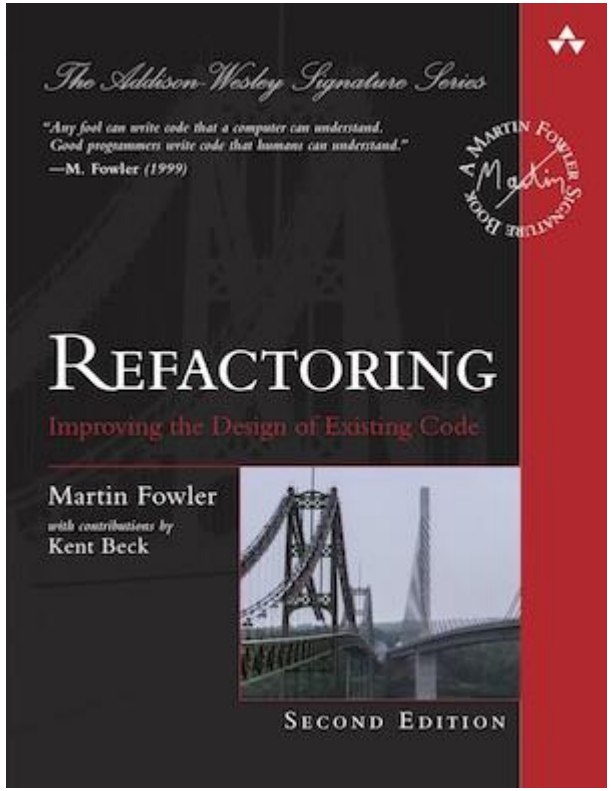
Programmeertechnieken/Programming Techniques

Part 5c
Refactoring

Koen Pelsmaekers

Campus Groep T, 2022-2023

The bible of Refactoring



"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
(Martin Fowler, 1999)

https://refactoring.com/

Refactoring, Improving the Design of Existing Code, Martin Fowler (+ Kent Beck), second edition, 2018.

What is “refactoring”?

- Fowler (noun):

  “A disciplined technique for restructuring an existing body of code, alterning its internal structure without changing its external behavior”

- Fowler (verb):

  “To restructure software by applying a series of refactorings without changing its observable behavior”

- Roberts:

  “A behaviour-preserving source-to-source program transformation”

- Beck:

  “A change to the system that leaves its behaviour unchanged, but enhances some non-functional quality – simplicity, flexibility, understandability, …”

Why do we need refactoring?

- To improve the software design
- To reduce
  - software decay/software aging
  - software complexity
  - software maintenance cost
- To increase
  - software understandability (readability), for instance by introducing design patterns
  - software productivity (at long term, not at short term)
- To facilitate future changes
  - useful in agile development
- …

When do we refactor?

- Consolidation phase in iterative development
  - adding new functionality – consolidate
  - especially when the functionality is difficult to integrate in the existing code base
- When you think it is necessary
  - not on a periodical base
- Apply the rule of three
  - first time: implement solution from scratch
  - second time: implement something similar by duplicating code
  - third time: do not reimplement or duplicate, but factorise!
- During normal code inspections/reviews
  - part of iterative development process

Some hints…

- It is no performance optimization
- Never refactor without unittests!
- Refactor in small steps
- Use IDE refactoring & code inspection support
- Identify *bad smells* in source code (see next slide)
- Some examples

  rename a class, extract a method, change the signature of a method, … or more complex operations as introduce inheritance instead of switch/instanceof
- Available for all major OO languages (Java, C++, C#, …), all major operating systems, all major IDE's

When to refactor? “Bad smells in code”

https://refactoring.com/catalog/

- Duplicated code
  - Extract Method
  - Pull up Method
  - Extract Class
- Long method (the longer the method the harder it is to see what it is doing)
  - “every comment”
  - Exctract Method
  - Introduce Parameter Object
  - Preserve Whole Object

When to refactor? (cont.)

- Data Clumps
  - “data items enjoy hanging around in groups together”
  - Extract Class
  - Introduce Parameter Object
  - Preserve Whole Object
- Switch/case or if Statements & Typecasts
  - Polymorphism & dynamic binding; replace “procedural code” by “object-oriented” code
- Data Class
  - “class with few methods (only getters/setters)
  - Encapsulate Field (to avoid public fields)
  - Encapsulate Collection
  - Move Method (try to move behavior into the class)
- Comments (to camouflage bad code)
  - Extract Method

When to refactor? (cont.)

- Large Class: too much responsibility?
  - “too many fields”
    - Extract Class
    - Extract Subclass
  - “too much code”
    - Extract Class
    - Extract Subclass
    - Extract Interface
- Long Parameter List
  - Replace Parameter with Method
  - Preserve Whole Object
  - Introduce Parameter Object