Refactoring and Readability

Dario Campagna

Head of Research and Development

Why refactoring?

Clean code

We want code that's easy to understand, to evolve, to maintain.

No ugly code

We want to keep the code from becoming rigid, fragile, inseparable, opaque.

Sustain pace

We want to protect us against the long-term erosion of our capacity to deliver features.

Refactoring

Safely improve the design of existing code.

Safely

Take baby steps, keep test bar green.

Improve the design

Does not add new functionalities.

Existing code

It is not rewriting from scratch.

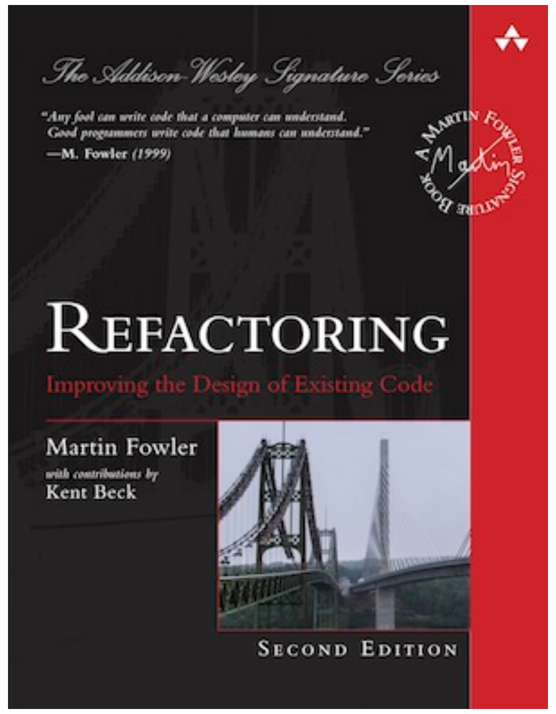
What to look for when refactoring?

Different ideas at different levels of abstraction.

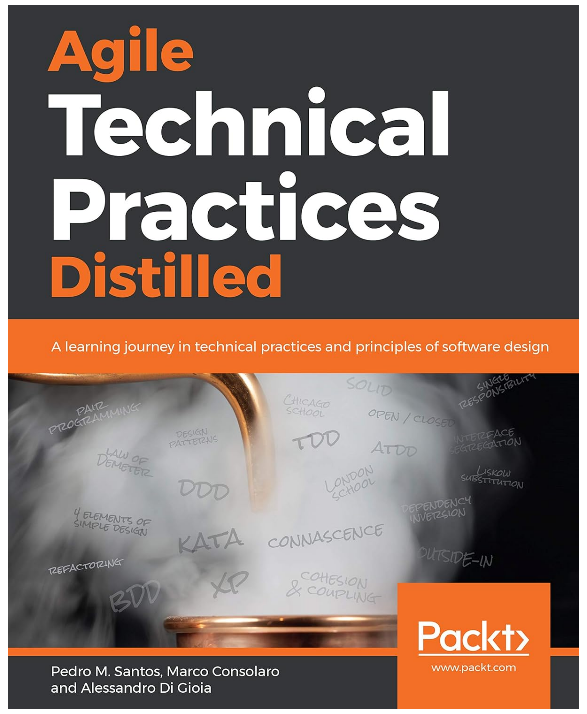
- Readability
- Code Smells
- Coupling and Cohesion
- S.O.L.I.D. Principles
- Simple Design

Some books about refactoring (and more)

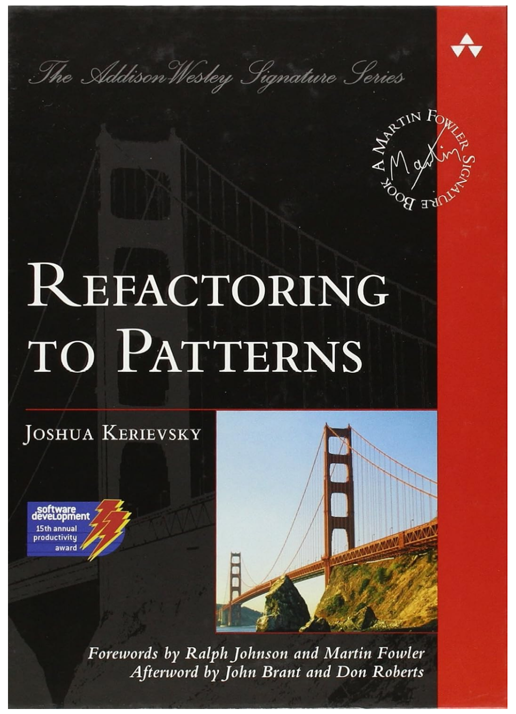
Look for them on http://www.biblio.units.it/


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." —M. Fowler (1999)
A Martin Fowler Signature Book
Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by Kent Beck
Second Edition

Refactoring by Martin Fowler


Agile Technical Practices Distilled
A learning journey in technical practices and principles of software design
Pedro M. Santos, Marco Consolaro and Alessandro Di Gioia
Packt
www.packt.com

Agile Technical Practices Distilled by Pedro Moreira Santos, Marco Consolaro, Alessandro Di Gioia


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Refactoring to Patterns
Joshua Kerievsky
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts

Refactoring to Patterns by Joshua Kerievsky

Readability

Small improvements in code readability can drastically improve code understandability

Ways to improve readability

Atomic refactors

Rename

Rename bad names, variables, arguments, instance variables, methods, classes.

Make abbreviations explicit.

Extract

Constants from magic numbers and strings.

Conditionals.

Extract a class (or methods or variables…), creating a new abstraction.

Inline

The inverse of extract – inline a method (or variable), deconstructing an abstraction.

Ways to improve readability

Atomic refactors

Move

Move a class (or methods or variables...) to some other place in the codebase.

Safe delete

Delete code and its usages in the code base.

Delete unnecessary comments.

Delete dead code.

Format

Format consistently and don’t force the reader to waste time due to inconsistent formatting.

Rename

The method name is accurate-but-vague.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Rename

The method name is accurate-but-vague.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Find

Rename

The method name is accurate-but-vague.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Find

Display

Rename

Now we have a precise name. Can we further improve readability?

```
private void findPriceAndDisplayAsText(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Extract

Two methods. Each with an intention-revealing name.

```
private String findPrice(String barcode) {
    return pricesByBarcode.get(barcode);
}

private void displayPrice(String priceAsText) {
    display.setText(priceAsText);
}
```

Tennis Refactoring Kata

Clean-up the code to a point where someone can read it and understand it with ease.

https://github.com/emilybache/Tennis-Refactoring-Kata

- Work on the class “TennisGame1”
- The test suite provided is fairly comprehensive, and fast to run.
- You should not need to change the tests, only run them often as you refactor.