Refactoring, 2nd Ed.
A love story
Michael Hunger
Michael Hunger
Open Sourcerer
Neo4j
@mesirii
I know what you're here for!

Half-listening to Conference Talks

In Depth

@ThePracticalDev

Covers By: dev.to/rly
Which Refactoring do you like most?

- Extract Method
- Delete Unused Code

Ok, you won :)
Deleting Code

Oh how good it feels

@ThePracticalDev
List of Soundbites
Stolen from the 1st Ed.
Questions?!

1. Who has seen the Refactoring book?
2. Who has read it?
3. Who thought that it's just common sense?
4. Who refactors code on a daily basis?
5. Who mentors new developers?
“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

M. Fowler (1999)
“What did I do to deserve this?”

Resolving Broken Dependencies
This is Your Life Now

@ThePracticalDev
My Refactoring Story

Started 18 years ago
Refactoring: Improving the Design of Existing Code

Martin Fowler

Foreword by Erich Gamma
Contributions by Kent Beck, William Opdyke, Don Roberts

28. Juni 1999
Product details

**Hardcover:** 464 pages

**Publisher:** Addison-Wesley Professional; 1 edition (July 8, 1999)

**Language:** English

**ISBN-10:** 9780201485677

**ISBN-13:** 978-0201485677

**ASIN:** 0201485672

**Product Dimensions:** 7.5 x 1.2 x 9.2 inches

**Shipping Weight:** 2 pounds (View shipping rates and policies)

**Average Customer Review:** ★★★★★ 214 customer reviews

**Amazon Best Sellers Rank:** #21,435 in Books (See Top 100 in Books)

  - #8 in Books > Textbooks > Computer Science > **Object-Oriented Software Design**
  - #12 in Books > Computers & Technology > Programming > Software Design, Testing & Engineering > **Object-Oriented Design**
  - #64 in Books > Textbooks > Computer Science > **Programming Languages**
"A must-read"
I really liked that book too!
As did others ... 

... e.g. authors!
Refactoring for Software Design Smells: Managing Technical Debt

Principle-Based Refactoring
Steve Halladay

Refactoring in Large Software Projects
Performing complex restructurings successfully
Stefan Roock, Martin Lippert
Translated by Bettina von Stockfleth
My Student thesis at TU Dresden
Refactoring
Benefits and Disadvantages of an Amazing Technique

Michael Hunger

25th October 2000

bit.ly/refactoring-thesis
Everyone knows of and has experienced the decay of existing software. Refactoring intends to work against this decaying.

This paper discusses the basics of refactoring and its uses for real-world projects. It reflects the surprisingly long (in terms of computer science) history of refactoring and the evolution and development of tools which allow the user to apply refactorings in a more comfortable way. Some of them already automate the application of whole refactorings as the Refactoring Browser [Refactoring Browser] and others offer considerable possibilities for evolving. The tools are tested with a small set of classes but due to the fact that most of the tools offer only a small subset of refactorings the tests were quite incomplete.
Refactoring Tools for Java in 2000!

Getting it to Work
And Having No Idea How

O RLY?

@ThePracticalDev
2.3.2 JRefactory (Java)

Figure 2.1: JRefactory UML Interface, including Metrics and Package Selector
2.3.3 IntelliJ Renamer (Java)

Figure 2.2: IntelliJ Renamer, Search Results for a Method Search
Figure 2.3: The Xref-Speller Interface within Emacs
Fast Forward 18 years
A nice surprise
Michael,

I'm Martin Fowler's editor at Addison-Wesley. We are working on a revision of his Refactoring book. Martin suggested that I reach out to you about reviewing the manuscript. He was very impressed with feedback you've provided on other projects.

Thanks,

Greg

--

Gregory Doench
Executive Editor
Pearson Technology Group
Several Weeks and 435 Pages later
28. November 2018

Latest Memo:
Most people will be disappointed by the second edition

martinfowler.com/articles/refactoring-2nd-ed.html
Like the original, this edition explains **what** refactoring is; **why** you should refactor; **how** to recognize code that needs refactoring; and **how** to actually do it successfully, no matter what language you use.

- Understand the process and general principles of refactoring
- Quickly apply useful refactorings to make a program easier to comprehend and change
- Recognize “bad smells” in code that signal opportunities to refactor
- Explore the refactorings, each with explanations, motivation, mechanics, and simple examples
- Build solid tests for your refactorings
- Recognize tradeoffs and obstacles to refactoring
Why a new book?

Variable Naming
The hardest part of coding

O RLY?
Creative Var. Name
Refactoring is as important as ever

Educate a new generation!
What changed?
No one knows anymore what video rentals are!
Javascript !?!?!
Javascript !?!?!
Book Structure (Basically the same)

1. Opening Narrative Example
   Theatre Invoicing
2. Principles
3. Code Smells
4. Testing
5. Catalogue
   a. most important ones first
   b. other refactorings
6. Dropped tangential topics and **big** refactorings
Refactoring Changes

"Split Temporary Variable" => "Split Variable"

"Extract Method" => "Extract Function"

Add Parameter, Remove Parameter, Rename Method

=> Change Function Declaration
Highlighted: Most important Refactorings

- Extract Function
- Inline Function
- Extract Variable
- Inline Variable
- Rename Variable
- Encapsulate Variable

- Migrate Function Declaration
- Change Function Declaration
- Introduce Parameter Object
- Combine Functions into Class
- Combine Functions into Transform
function printOwing(invoice) {
    printBanner();
    let outstanding = calculateOutstanding();

    // print details
    console.log(`name: ${invoice.customer}`);
    console.log(`amount: ${outstanding}`);
}

function printDetails(outstanding) {
    console.log(`name: ${invoice.customer}`);
    console.log(`amount: ${outstanding}`);
}
New Refactorings

All about Functions and Pipelines
New Refactorings

Combine Functions into Class
Combine Functions into Transform
Move Statements into Function
Move Statements to Callers
Remove Dead Code
Rename Field
Rename Variable
Replace Command with Function
Replace Derived Variable with Query
Replace Inline Code with Function Call
Replace Loop with Pipeline
Replace Query with Parameter
Replace Subclass with Delegate
Return Modified Value
Split Phase
Combine Functions into Class

```
function base(aReading) {...}
function taxableCharge(aReading) {...}
function calculateBaseCharge(aReading) {...}
```

class Reading {
    base() {...}
    taxableCharge() {...}
    calculateBaseCharge() {...}
}
function base(aReading) {...}
function taxableCharge(aReading) {...}

function enrichReading(argReading) {
  const aReading = _.cloneDeep(argReading);
  aReading.baseCharge = base(aReading);
  aReading.taxableCharge = taxableCharge(aReading);
  return aReading;
}
Move Statements to Callers

emitPhotoData(outStream, person.photo);

function emitPhotoData(outStream, photo) {
    outStream.write('<p>title: ${photo.title}</p>
    outStream.write('<p>location: ${photo.location}</p>
}

emtPhotoData(outStream, person.photo);
outStream.write('<p>location: ${person.photo.location}</p>

function emitPhotoData(outStream, photo) {
    outStream.write('<p>title: ${photo.title}</p>
}
const names = [];
for (const i of input) {
    if (i.job === "programmer")
        names.push(i.name);
}

const names = input
    .filter((i) => i.job === "programmer")
    .map((i) => i.name)
    ;
Web-Book Available on Safari

Book Description

Fully Revised and Updated—Includes New Refactorings and Code Examples

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”
—M. Fowler (1999)

For more than twenty years, experienced programmers worldwide have relied on Martin Fowler’s *Refactoring* to improve the design of existing code and to enhance software maintainability, as well as to make existing code easier to understand.

This eagerly awaited new edition has been fully updated to reflect crucial changes in the programming landscape. *Refactoring, Second Edition*, features an updated catalog of refactorings and includes JavaScript code examples, as well as new functional examples that demonstrate refactoring without classes.
My take on the new book
Back to the essence

Feigning knowledge of a word you've heard a few times

Expert
Pretending to Know About Stuff

O RLY?
@ThePracticalDev
Refactoring
Benefits and Disadvantages of an Amazing Technique

Michael Hunger

25th October 2000
Topics

1. What, When, How
2. Benefits
3. Costs / Issues
4. History
5. Catalogue
6. Tools
7. Example
What?
Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing the observable behavior of the software.
Refactor (verb): to restructure software by applying a series of refactorings without changing the observable behavior of the software.
When?
Code Smells!

If it stinks, change it
- Grandma Beck
Code Smells

Mysterious Name
Duplicated Code
Long Function
Long Parameter List
Global Data
Mutable Data
Divergent Change
Shotgun Surgery
Feature Envy
Data Clumps
Primitive Obsession
Repeated Switches
Lazy Element
Speculative Generality
Temporary Field
Message Chains
Middle Man
Insider Trading
Large Class
Alternative Classes with Different Interfaces
Data Class
Refused Bequest
Comments
Loops
Adding features is hard?
Working Effectively with Legacy Code

Michael Feathers
Full of complexity and bugs
Code as a Crime Scene

Software Design X-Rays

Adam Tornhill
Code is not habitable
Timeless Way of Building

Christopher Alexander
How?
Refactoring is as much about safety testing, small steps, commit as it is about code changes.
Refactoring changes the programs in small steps. If you make a mistake, it is easy to find the bug.
Two Hats
Adding Function

Refactoring
Add Function

1. Red
2. Green
3. Refactor
Refactor

1. Refactor
2. Green
3. Commit
Before you start refactoring, check that you have a solid suite of tests. These tests must be self-checking.

Martin Fowler
Tests, are your **safety net**

Listen to the flight attendant this **one** time
Make sure all tests are fully automatic and that they check their own results.

Martin Fowler
Fast Tests
Test Ordering

R.I.P. JunitMax
Many small commits
Can squash later
Run your tests frequently. Localize tests whenever you compile - every test at least every day.

Martin Fowler
A suite of tests is a powerful bug detector that decapitates the time it takes to find bugs.

Martin Fowler
Unit tests are like the second pair of eyes of a pair programmer.

Kent Beck
When you get a bug report, start by writing a unit test that exposes the bug.

Martin Fowler
Think of the boundary conditions under which things might go wrong and concentrate your tests there.

Martin Fowler
Don’t forget to test that exceptions are raised when things are expected to go wrong.

Martin Fowler
Kinds of Refactoring
Continuous Clean-Up
Continuous Clean-Up

- Rename
- Replace comment with function
- Make intent clear
- Intention in your IDE
- Not about polishing
DRY

Three strikes and you refactor.
Boy Scout Rule

*Leave it better than you found it*
Use >> Reuse

Reuse within your module not across.
Planned Refactoring
Planned Refactoring

- Before adding a feature
- Addressing technical debt
- Improve testability (seams)
- Improve understandability
- For learning about a new code-base
(BIG) Refactoring / Rewrite
Sunken Cost Fallacy
Deliberate Discovery

Dan North
Benefits
Forces you to test
Don’t let the fear that testing can’t catch all bugs stop you from writing the tests that will catch most bugs.

Martin Fowler
It is better to write and run incomplete tests than not to run complete tests.

Martin Fowler
Code Ownership

Modify your code ownership policies to smooth refactoring.
Essential
Hating Other People’s Code

It's only a clever hack if you're the one who wrote it

Fights Code Rot and Decay

O RLY?

@ThePracticalDev
Improves Understanding
Costs
APIs

Don’t publish interfaces prematurely. Downstream dependencies.
Side Effects

e.g. Performance
How do I tell my Boss?
Where is my Spare Time?
The Time Turner Manual

Time invested

O RLY²

Hermione Granger
Refactoring & Design

Buzzword-first Design
The Definitive Guide

ORLY?
@ThePracticalDev
Refactoring & Design

SDD: Spaghetti Driven Design
A tasty guide to code

O RLY?
The top 10 Refactorings
Not another list

Refactoring Catalogue

O RLY?
Know-It-All
Refactoring Catalogue (my favorites)

Extract Function
Extract Variable
Push Statements into Function
Push members up
Extract Interface / Superclass
Migrate Function Declaration
Combine Functions into Transform
Encapsulate Collection
Replace Temp with Query
Separate Query from Modifier
Parameterize Function
Move Function
Split Loop
Replace Loop with Pipeline (Stream)
Replace Magic Literal
Replace Derived Variable with Query
Replace Typecode with Subclass
Replace Superclass with Delegate
Replace Control Flag with Break (Return)
Introduce Special Case (Null Object)
Replace Nested Conditionals with Guard Clauses
Extract Variable

```
return order.quantity * order.itemPrice -
    Math.max(0, order.quantity - 500) * order.itemPrice * 0.05 +
    Math.min(order.quantity * order.itemPrice * 0.1, 100);
```

```
const basePrice = order.quantity * order.itemPrice;
const quantityDiscount = Math.max(0, order.quantity - 500) * order.itemPrice * 0.05;
const shipping = Math.min(basePrice * 0.1, 100);
return basePrice - quantityDiscount + shipping;
```

Alias: *Introduce Explaining Variable* (in First Edition)

Inverse of: *Inline Variable*
let averageAge = 0;
let totalSalary = 0;
for (const p of people) {
    averageAge += p.age;
    totalSalary += p.salary;
}
averageAge = averageAge / people.length;

let totalSalary = 0;
for (const p of people) {
    totalSalary += p.salary;
}

let averageAge = 0;
for (const p of people) {
    averageAge += p.age;
}
averageAge = averageAge / people.length;
Replace Derived Variable with Query

```javascript
get discountedTotal() {return this._discountedTotal;}
set discount(aNumber) {
    const old = this._discount;
    this._discount = aNumber;
    this._discountedTotal += old - aNumber;
}

get discountedTotal() {return this._baseTotal - this._discount;}
set discount(aNumber) {this._discount = aNumber;}
```
Replace Type Code with Subclasses

```javascript
function createEmployee(name, type) {
    switch (type) {
        case "engineer": return new Engineer(name);
        case "salesman": return new Salesman(name);
        case "manager": return new Manager(name);
    }
}
```


Inverse of: *Remove Subclass*
Replace Superclass with Delegate

class List {
    ...
}
class Stack extends List {
    ...
}

class Stack {
    constructor() {
        this._storage = new List();
    }
}
class List {
    ...
}

Alias: Replace Inheritance with Delegation (in First Edition)
Crappy Developer Tools

Whatever works

As long as it's free
Tools today make Refactoring a Joy
Smart Understanding of code & intent

Useful suggestions

Intentions & Quick fixes
Use and Learn the Shortcuts

Pragmatic Programmer
Intentions & Quick Fixes
IntelliJ & all JetBrains IDEs
Help
Productivity Guide

The Extract Variable refactoring helps you simplify complicated statements in your code. For example, in the code fragment below, you can select an expression in the code:

```java
myEditorPane.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
```
and press `Ctrl`+`V` (refactor | Extract | Variable...). This will result in the following:

```java
new JEditorPane();
```
What did you NOT use yet
Eclipse
Editor developers focus on the editor part
Language developers focus on the language support

Eclipse
Language
Server
Netbeans
VS Code & Visual Studio
3 Space Indentation

The only true indent

ORLY®

Satan
Refactoring & Patterns
Refactoring & Patterns

- patterns encapsulate good solutions in a context
- represent roles of components
- avoid code-smells
- they can be the goal of a design improvement
- refactorings are steps to those designs
Resources

dzone.com/refcardz/refactoring-patterns

refactoring.guru

martinfowler.com/tags/refactoring.html

refactoring.com/catalog
Ask your questions?

Thank you for your time!

Follow @mesirii