Java Refactoring: Best Practices

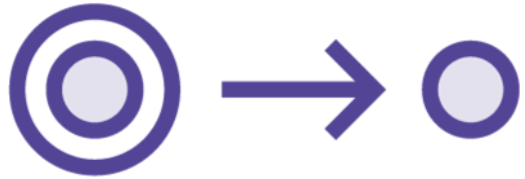
INTRODUCTION TO REFACTORING

Andrejs Doronins
TEST AUTOMATION ENGINEER

Main Programming Activities

Reading

Writing



Changing

Refactoring

Why?

What?

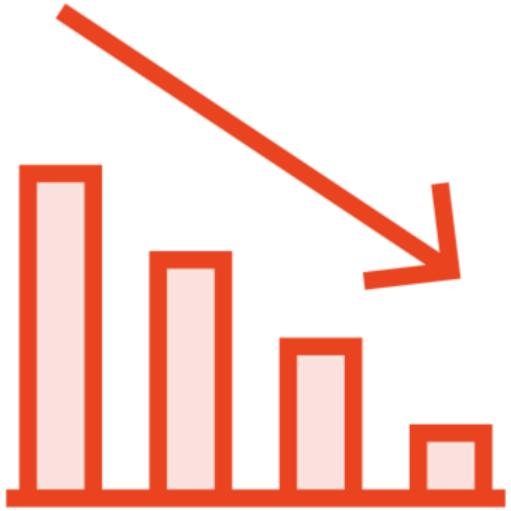
When not?

When?

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”

Martin Fowler

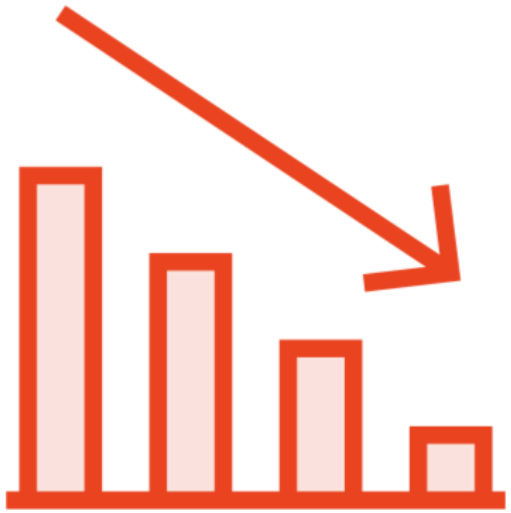
Without Refactoring



Productivity decreases:

- Duplicated code accumulates
- Logic becomes more complex
- Code is difficult to understand

Reasons for Technical Debt



Not enough experience to write code well

Lazy coding

Tight deadlines

Junior

Mid-level

Senior

Low

Code Quality

High

Can you deliver this new feature yesterday?

Right...

Boy Scout Rule:
Leave the ~~campground~~ code
cleaner than you found it.

Reasons to Remove Technical Debt

Easier and faster to change existing code

Don’t Remove Technical Debt if:

Current code doesn’t work

Deadlines must be met

It results in gold-plating

“Business is well served by continuous refactoring, yet the practice of refactoring must coexist harmoniously with business priorities.”

Joshua Kerievsky

Code Smell

A surface indication that usually corresponds to a deeper problem in the system

Don’t fix what’s not broken!


Learn a Code Smell
Understand why it is bad
Identify it in Code
Fix it

Standard Refactoring Process

Prerequisites

Java and OOP

Any IDE

Non-exhaustive List of Code Smells

Long parameter list	Long method	Primitive obsession
Data Clumps	Large class	Refused Bequest
Divergent Change	Shotgun surgery	Feature Envy
Message chains	Bad comments	Dead code
...	...	...

Mika Mäntylä

Casper Lassenius

Code
Smell
Taxonomy

Code Smells Taxonomy

Bloaters

Object-Oriented Abusers

Change Preventers

Couplers

Dispensables