Refactoring Fundamentals: Introducing Refactoring

Steve Smith

Ardalis.com

Twitter: @ardalis

pluralsight
hardcore developer training

In This Course

- What is Refactoring?
- Why do it?
- What's the process?
- What are some tools that can assist with it?
- What is a *Code Smell*?
- What are some examples of Code Smells?
- What are some common refactorings?
- How does one apply them correctly?

Related Courses

Design Patterns Library

A reference library for design patterns of all types

+1 0 Tweet 0 Like 0 Send Share

SOLID Principles of Object Oriented Design

This course introduces foundational principles of creating well-crafted code and is appropriate for anyone hoping to improve as a developer

Test First Development - Part 1

Test first development techniques and practices with C#, Visual Studio, and NUnit

+1 0 Tweet 0 Like 0 Send Share 14

What is Refactoring?

Two Related Definitions

A refactoring

A change made *to make it easier to under* *t changing its observable b*

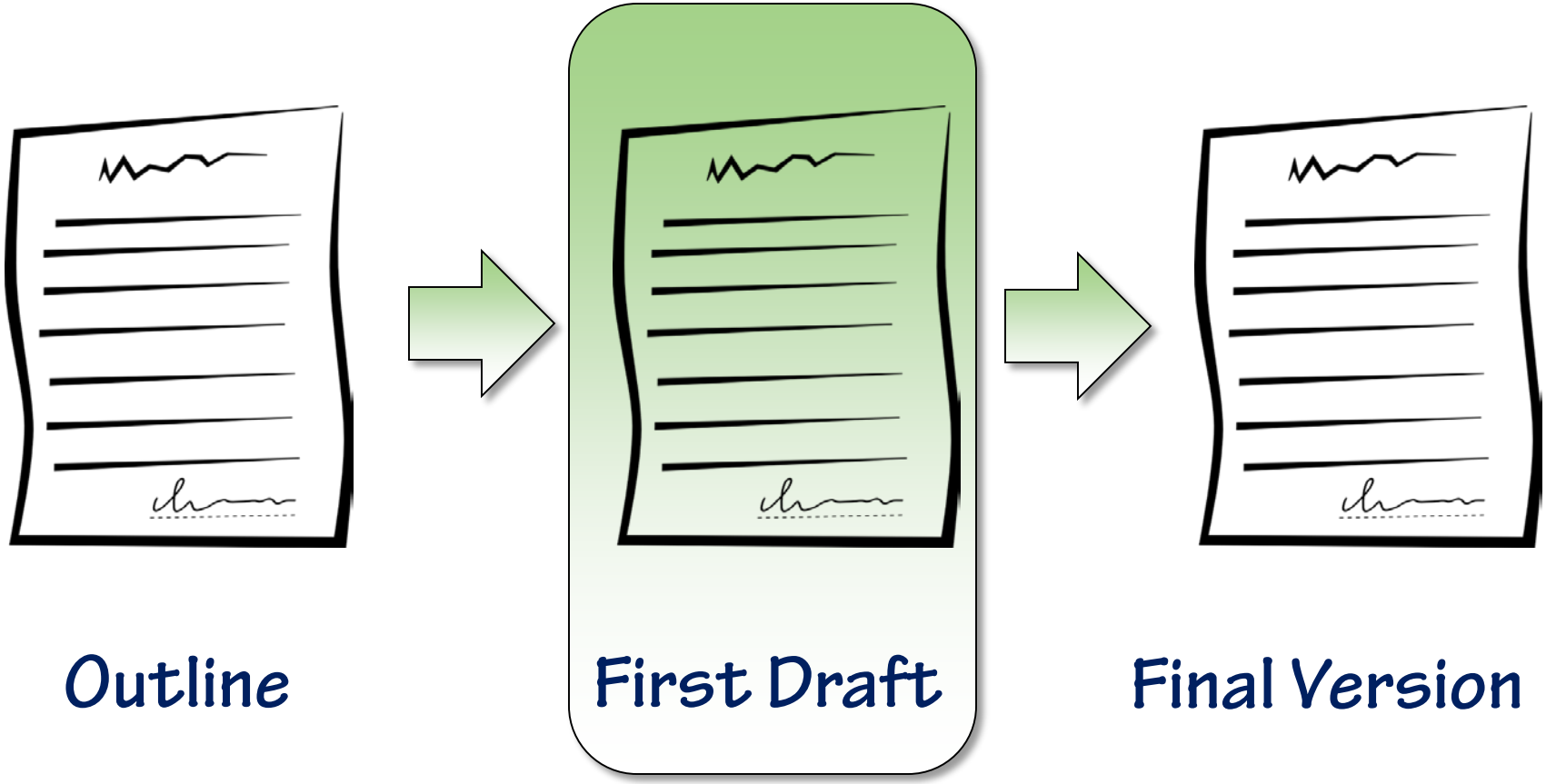
"Refactoring" is derived from the term "factoring"

To refactor (v

To restructure software by applying one or more refactorings (without changing observable behavior)

Adapted from *Refactoring*, Fowler

Refactoring is Editing



Refactoring is Editing

Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte.

I have made [this letter] longer than usual because I have not had time to make it shorter.

Blaise Pascal, 1657

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Martin Fowler, Refactoring

http://quoteinvestigator.com/2012/04/28/shorter-letter/

Refactoring is Preventive Maintenance

Over time, software *rots* or *decays*

- Duplication
- Excess Coupling
- Quick Fixes
- Hacks
- Technical Debt


http://flickr.com/photos/lwr/47250008

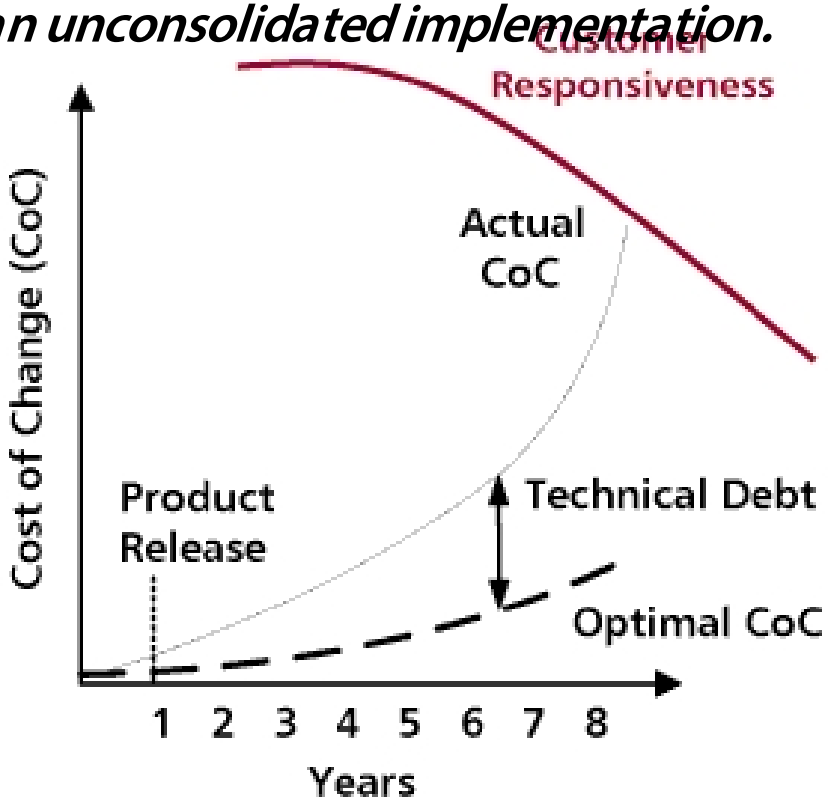
Why Should You Refactor?

- Improves Design
- Improves Readability
- Reveals Defects
- Helps You Program Faster

Technical Debt

Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite [refactoring]… The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation.

Ward Cunningham



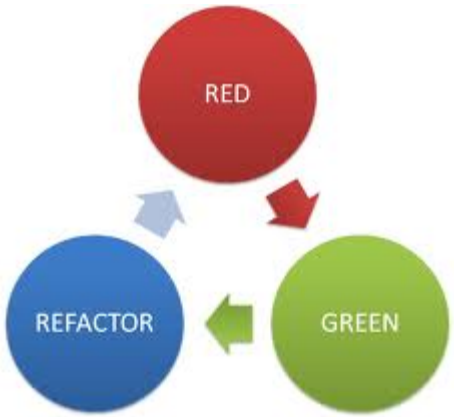
- Once on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt, estimating is nearly impossible
- Only three strategies:
  1. Do nothing, it gets worse
  2. Replace, high cost/risk
  3. Incremental refactoring, commitment to invest

Image: *http://bit.ly/cL4MJ3*

When Should You Refactor?

http://en.wikipedia.org/wiki/File:Restaurant.jpg

When Should You Refactor?



- **Following Test Driven Development?**
  - Red – Green – Refactor
- **Following Pain Driven Development?**
  - If it's causing you pain, refactor it!
- **Refactor to make adding a new function easier**
- **Refactor as part of the process of fixing bugs**
- **Refactor as part of code reviews**
  - Formal, scheduled reviews can be interactive
  - Pair Programming can provide continuous code reviewing

When Shouldn't You Refactor

- **Massive Technical Debt**
  - Declare Technical Bankruptcy
  - Rewrite the Application
- **Current Code Doesn't Work**
- **Imminent Deadline**
  - Incur conscious technical debt

Other than when you are very close to a deadline, however, you should not put off refactoring because you haven't got time. Experience with several projects has shown that a bout of refactoring results in increased productivity. Not having enough time usually is a sign that you need to do some refactoring.

Martin Fowler, Refactoring

Refactoring Principles

Keep It Simple

Keep It DRY

Make It Expressive

Kent Beck's Rules of Simple Design (in priority order):

1. Run all the tests
2. Contain no duplicate code
3. Express all the ideas the authors wants to express
4. Minimize classes and methods

Reduce Overall Code

Separate Concerns

Appropriate Level of Abstraction

Boy Scout Rule

Leave your code better than you found it.

The Refactoring Process

- **(check in or back up the current code!)**
- **Verify Existing Behavior**
- **If no unit tests exist, write *characterization tests***
  - Write a test you know will fail
  - Use the output of the failing test to determine existing behavior
  - Update the test to assert the existing behavior
  - Run the test again; it should pass
- **Apply Refactoring**
- **Confirm existing behavior has been preserved**
  - If not, roll back and try again using smaller steps

Refactoring Tips

- Keep Refactorings Small
- One At A Time
- Make a Checklist
- Make a “Later” List
- Check In Frequently
- Add Test Cases
- Review the Results

Refactoring Tools

Commercial Tools

- Visual Studio (Microsoft)
- JustCode (Telerik)
- ReSharper (JetBrains)
- CodeRush (DevExpress)
- Visual Assist X (Whole Tomato)

Where I work; assume bias

http://www.agile-code.com/blog/list-of-visual-studio-code-refactoring-

Premature Optimization

Premature optimization is the root of all [software] evil.

Donald Knuth

- **Avoid over-engineering**
  - Follow YAGNI
- **Don't spend more time on refactoring than on delivering value**
  - Avoid Gold-Plating your code

Refactoring and Tests

- Treat Tests Like Production Code
- Keep Them DRY, but Expressive

Demo

A Simple Refactoring

Summary

What is refactoring?

Why should we refactor?

When should we do it? When shouldn't we?

How do we do it?

What tools can we use to make it easier?

References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Test-First Development http://bit.ly/XliJHQ

Design Patterns Library http://bit.ly/SJmAX1

Books

Refactoring http://amzn.to/110tscA

Refactoring to Patterns http://amzn.to/Vq5Rj2

Working Effectively with Legacy Code http://amzn.to/VFFYbn

Code Complete http://amzn.to/Vq5YLv

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

pluralsight
hardcore developer training