

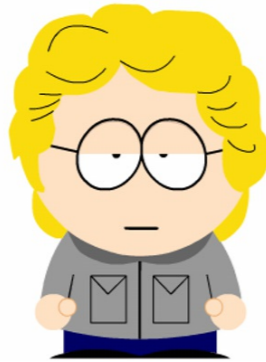
# Smaller!

**Craig Buchek**



<http://boochtek.com/aatc2017>

# About Me



Craig Buchek



BoochTek



This Agile Life

# Obligatory Cat Picture



# Smaller - Why?

- Focus on what's valuable
- Focus on task at hand
  - It's easier to focus on smaller things
  - We get bored when we work too long on a single thing
- Builds momentum
- Easier to modify
- Easier to throw away
- Smaller code usually runs faster
  - Does less
  - Fits in cache better

# Smaller - What?

- Stories
- Tests
- Methods
- Classes
- Commits
- Releases

# Smaller - How?

- Story splitting
- Better tests
- Refactoring
- Discipline

# Smaller Stories - Why?

- Smaller stories can be estimated more easily
- Smaller stories can be completed quicker
  - Delivers value quicker
  - Making quicker progress leads to more progress
- It's easier to realize you don't need a smaller story

# Smaller Stories - How?

- Thin vertical slices
- Story splitting
- Minimal Marketable Feature (MMF)
- One acceptance criteria per story
- INVEST
  - Independent
  - Negotiable
  - Valuable
  - Estimable
  - Small
  - Testable



# Vertically Sliced Increments

Not like this....

1 2 3 4

---

Like this!

1 2 3 4 5

Henrik Kniberg

The image illustrates two different ways to incrementally build a car. The top section, titled 'Not like this....', shows four stages where the car's head and body are disconnected. Stage 1 is a sad face above a single wheel. Stage 2 is a sad face above two wheels on an axle. Stage 3 is a sad face above a car body with two wheels. Stage 4 is a happy face above a complete car. The bottom section, titled 'Like this!', shows five stages where the car's form evolves smoothly. Stage 1 is a sad face above a skateboard. Stage 2 is a neutral face above a skateboard with a kickstand. Stage 3 is a neutral face above a bicycle. Stage 4 is a happy face above a motorcycle. Stage 5 is a very happy face above a complete car. The name 'Henrik Kniberg' is written in the bottom left corner of the diagram area.



# Thin Vertical Slices - Login

As a user,  
I want to log **in**,  
So that I can use the app

Given an existing user account  
When I log **in** with correct credentials  
Then I should be logged **in**

Given an existing user account  
When I log **in** with incorrect credentials  
Then I should see an error message  
And I should **not** be logged **in**

# Thin Vertical Slices - Point of Sale

As a customer,  
I want to buy something from the store,  
So that I can take it home **and** use it

When I take something to the cash register  
And the cashier rings it up  
Then the cash register should look up the price  
And tax should be added  
And I should be able to pay by cash, check, **or** credit card  
And a receipt should be printed  
And inventory should be updated

# Smaller Tests

- Arrange, Act, Assert
  - Do as little as possible in each step
- One assertion per test
- Better understanding of the problem
- Focus on the problem before thinking about solutions
- If you nest describe blocks:
  - Put your initial state on the outside
- Discipline

# Refactoring

Modifying code to improve its internal structure, without changing its external behavior, in order to make it easier to understand and cheaper to modify

# Refactoring

**Modifying code** to improve its internal structure, without changing its external behavior, in order to make it easier to understand and cheaper to modify

# Refactoring

Modifying code to **improve its internal structure**, without changing its external behavior, in order to make it easier to understand and cheaper to modify

# Refactoring

Modifying code to improve its internal structure, **without changing its external behavior**, in order to make it easier to understand and cheaper to modify



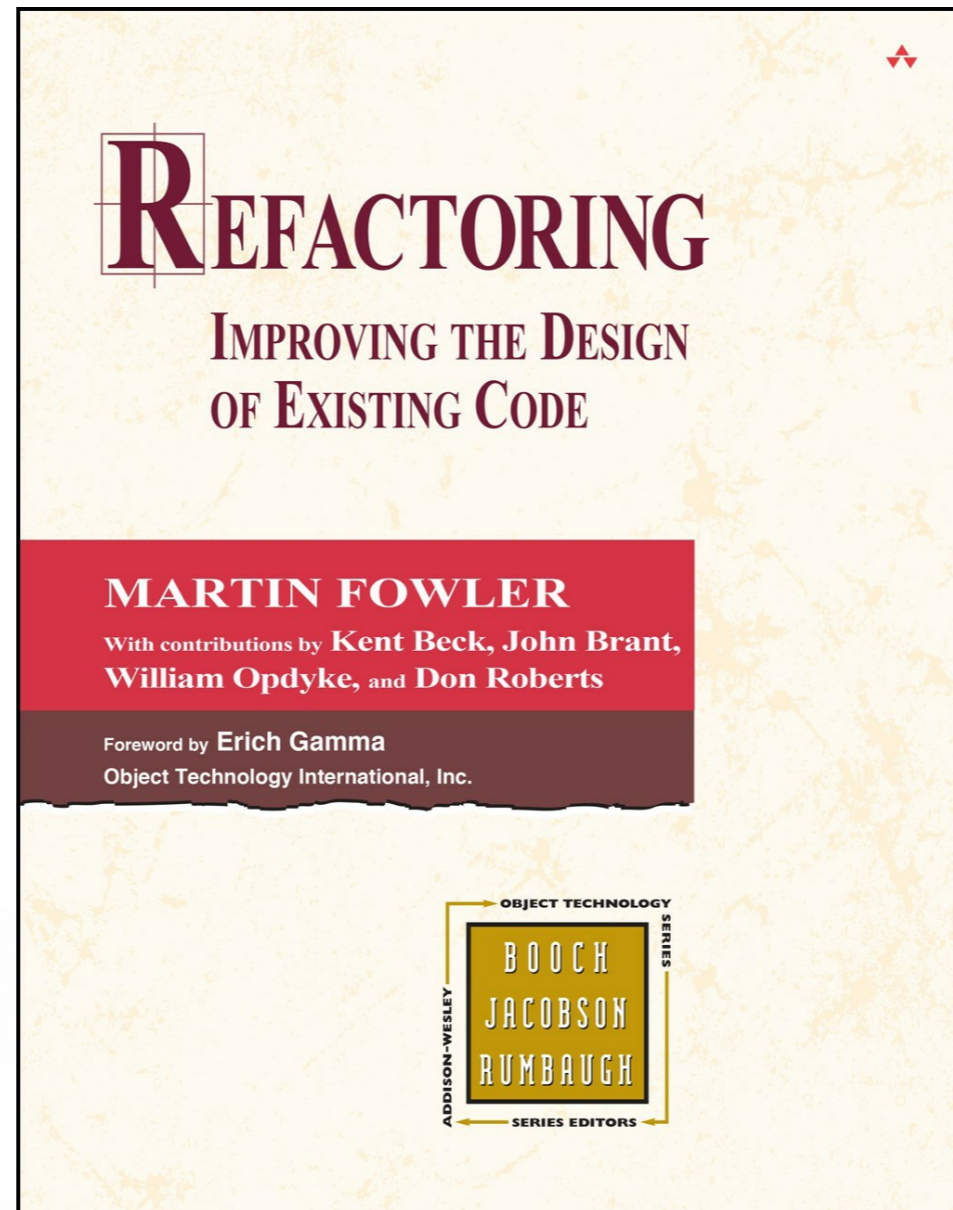
# Refactoring

Modifying code to improve its internal structure, without changing its external behavior, in order to make it **easier to understand** and cheaper to modify

# Refactoring

Modifying code to improve its internal structure, without changing its external behavior, in order to make it easier to understand and **cheaper to modify**

# Refactoring - Canonical Reference



# Why Refactor?

- Readability
- Simplification
  - DRY - Don't Repeat Yourself
  - SRP - Single Responsibility Principle
- Improved extensibility
- Maintainability
- Reduced bugs
- Improved performance

# When To Refactor

- Before making requested changes
  - To clarify what's going on
- After adding requested changes
  - Red, Green, Refactor
- When you realize you've got too much tech debt
- When you see something that's a problem

# What Do I Need to Refactor?

- Knowing what code needs refactoring
- Tests for the code in question
- Knowing what refactorings are available
- Automated refactoring tools (optional)

# Refactorings Have Names

- Extract Method
- Move Method
- Extract Variable
- Inline Temp
- Replace Parameter with Method

Catalog of refactorings: <https://refactoring.com/catalog/>

# How Do I Know What To Refactor?

- Code "smells"
- Heuristics
  - Sandi Metz's rules
- Your own *gut feelings* from experience
- Test "smells"



# Code Smells

- Duplicated code
- God class
- Feature envy
- Too many parameters
- Long method
- Comment

# Code Smells

- Duplicated code
- God class
- Feature envy
- Too many parameters
- Long method
- Comment

# Code Smells

- Duplicated code
- God class
- Feature envy
- Too many parameters
- Long method
- Comment

# Code Smells

- Duplicated code
- God class
- Feature envy
- Too many parameters
- Long method
- **Comment**

# Test Smells

- Too many collaborators
- Tests mirror code too closely
- Fragile tests
- Slow tests
- If your tests are hard to write, your code is probably too complex

# Sandi Metz's Rules

1. Classes should be no longer than 100 lines of code
2. Methods should be no longer than 5 lines of code
3. Methods should take no more than 4 parameters
  - Hash options count as parameters
4. Break the rules only if you can convince your pair

# When Am I Done Refactoring?

- When code is as clear as possible
- You're likely **not** overdoing it
- When every method is 1 line long
  - Preferably with no ifs
- When you meet the Sandi Metz metrics
- When you meet the Four Rules of Simple Design
  - Passes the tests
  - Reveals intention
  - No duplication
  - Fewest elements

# Smaller Methods

- Should do one thing
- Refactoring
  - Extract Method
  - Replace Parameter with Method
  - Decompose Conditional
  - Replace Conditional with Polymorphism
  - Introduce Null Object
- Use more declarative terms
  - Don't pretend to tell when you're asking
    - Don't use get or compute methods
- Don't have methods that pre-compute values
  - Just ask for the value when you need it
    - Memoize if necessary





# Smaller Methods - Booleans

- Original

```
def deletable?  
  if sequential_approvers_enabled?  
    !answered?  
  else  
    true  
  end  
end
```

# Smaller Methods - Booleans

- Amos's partial refactoring

```
def deletable?  
  !(sequential_approvers_enabled? && answered?)  
end
```

OR

```
def deletable?  
  !sequential_approvers_enabled? || !answered?  
end
```

# Smaller Methods - Booleans

- Amos's suggested refactoring

```
def deletable?  
  sequential_approvers_disabled? || unanswered?  
end  
  
def unanswered?  
  !answered?  
end  
  
def sequential_approvers_disabled?  
  !sequential_approvers_enabled?  
end
```

# Smaller Classes

- Single Responsibility Principle (SRP)
  - A class should have only one reason to change
- Remove duplication (DRY)
- Refactoring
  - Extract Class
  - Extract Superclass / Subclass
  - Extract Interface

# Smaller Commits - Why?

- Rolling back an atomic unit
- git bisect

# Smaller Releases - Why?

- Less to go wrong
- Practice makes perfect

# Smaller Releases - How?

- Build confidence with customers and management
- Release bug fixes "out of band"
- Show that smaller changes are less risky
- Keep decreasing time between releases

# What Else?

- Meetings!
- Feedback loops
- Pair-switching
  - Every 2 hours
  - Every hour?
- Others?



# Exceptions to the Rule

- Time spent with the customer
- Commit messages
- Others?

# Workshop

<https://github.com/boochtek/aatc2017>

# Thanks

# Feedback

- Twitter: [@CraigBuchek](https://twitter.com/CraigBuchek)
- GitHub: [booch](https://github.com/booch)
- Email: [craig@boochtek.com](mailto:craig@boochtek.com)
- Slides: <http://boochtek.com/aatc2017>
  - [Remark](#) presentation software