

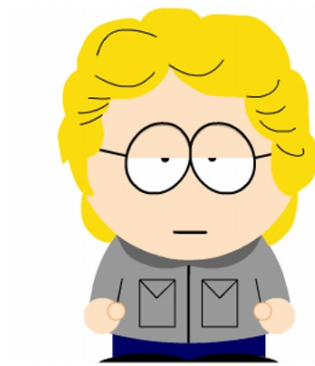
Smaller!

Craig Buchek



<http://boochtek.com/aatc2017>

About Me



Craig Buchek



BoochTek



This Agile Life

My Cat



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

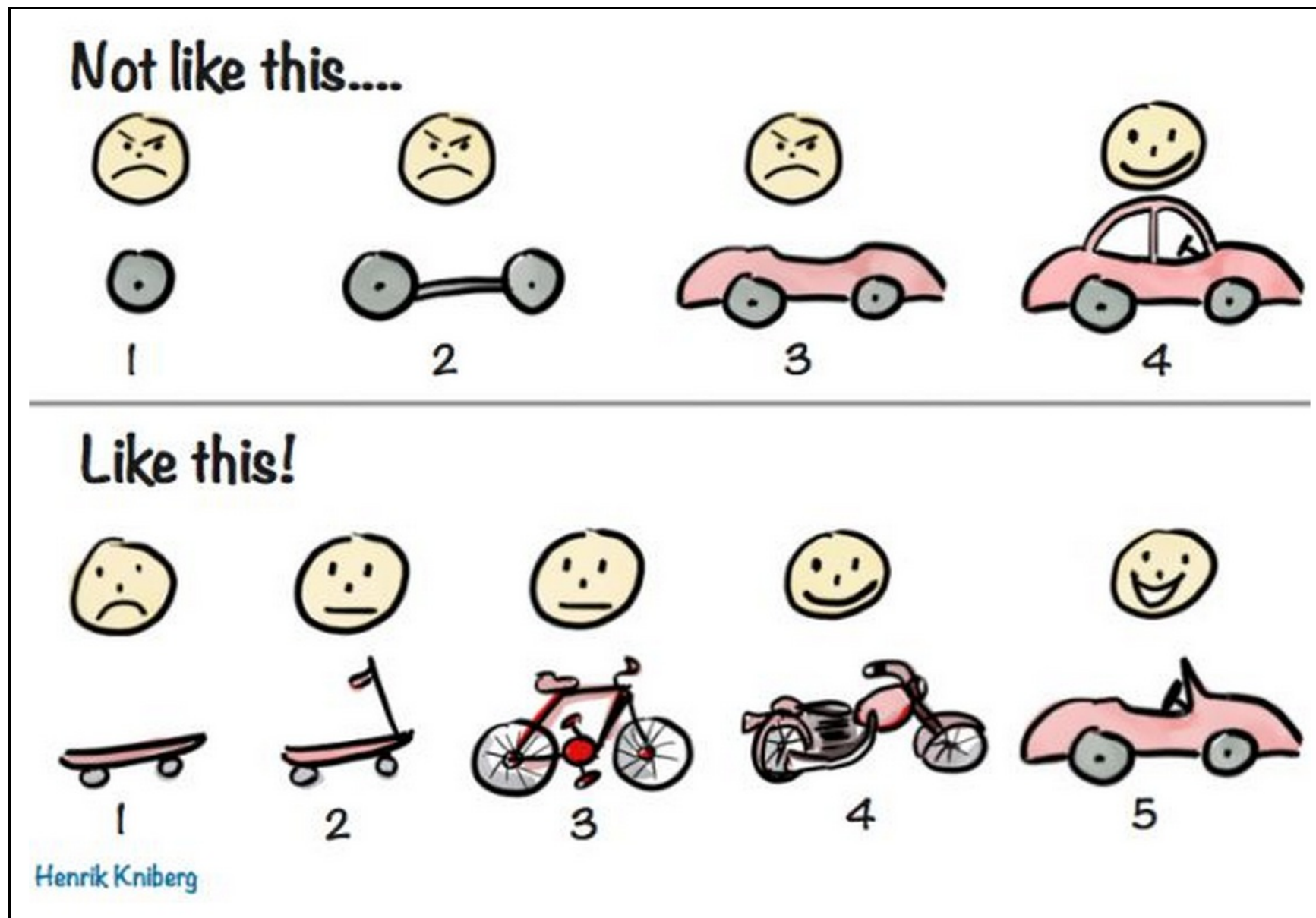
Smaller Stories - Why?

- 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



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

Smaller Methods

- Should do one thing
- Use more declarative terms
- Don't have methods that pre-compute values
 - Just ask for the value when you need it
 - Memoize if necessary
 - Don't pretend to tell when you're asking
 - Don't use get or compute methods

Smaller Classes

- Single Responsibility Principle (SRP)
 - A class should have only one reason to change

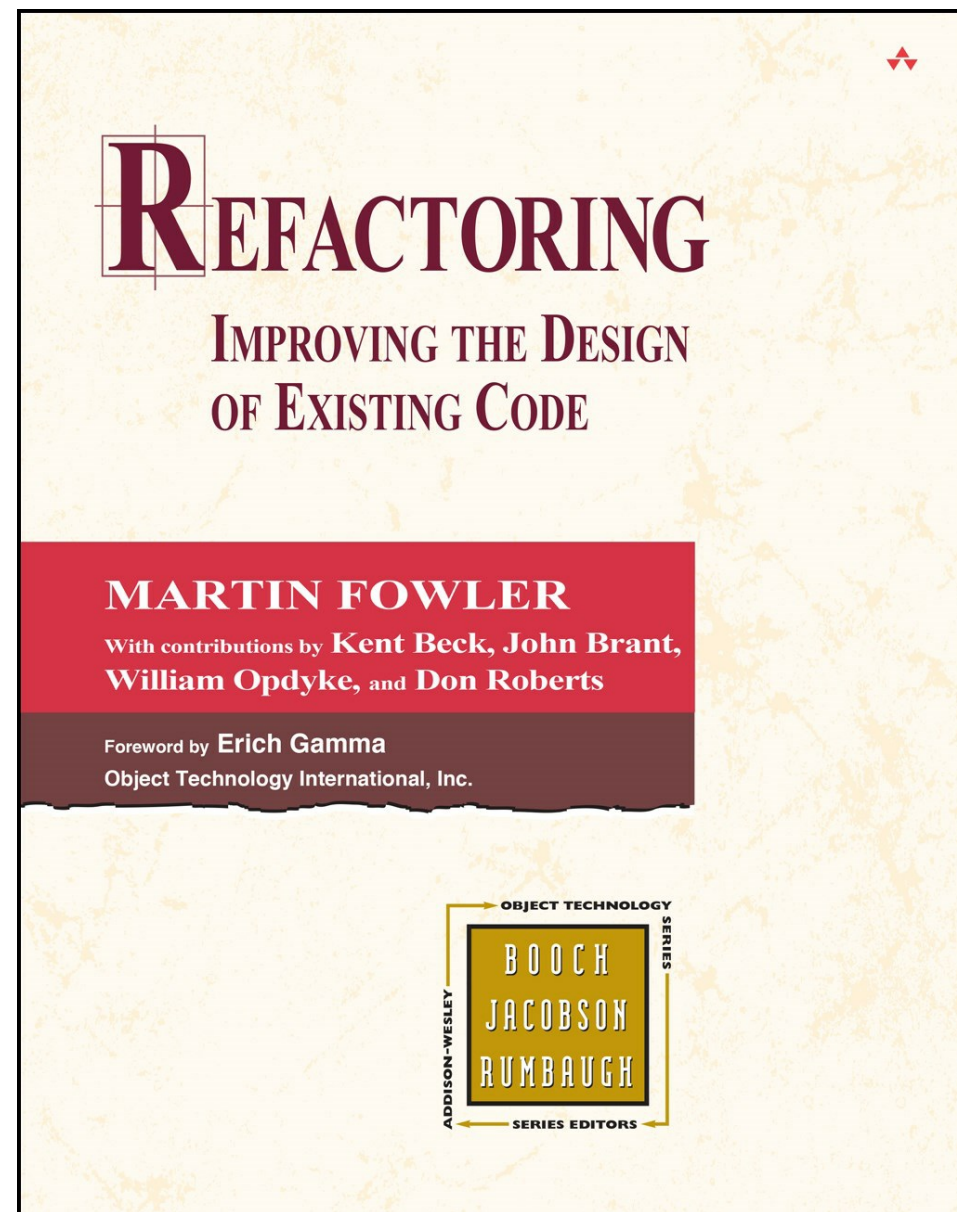
What Is Refactoring?

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

History of Refactoring

- Term first used by Ralph Johnson in 1990
- Popularized by Extreme Programming in 1999
- Martin Fowler's book written in 1999

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 from experience
- Test "smells"

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. Rails views should receive only 1 object from the controller
5. Rails controller actions should know only 2 other class names
6. Break the rules only if you can convince your pair

When Am I Done Refactoring?

- When code is as clear as possible
 - Never?
- You're likely **not** overdoing it
- When every method is 1 line long
 - 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 - 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 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

Exceptions to the Rule

- Commit messages
- Others?

Workshop

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

Thanks

Feedback

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