Refactoring in Ruby

Improve Your Code Incrementally

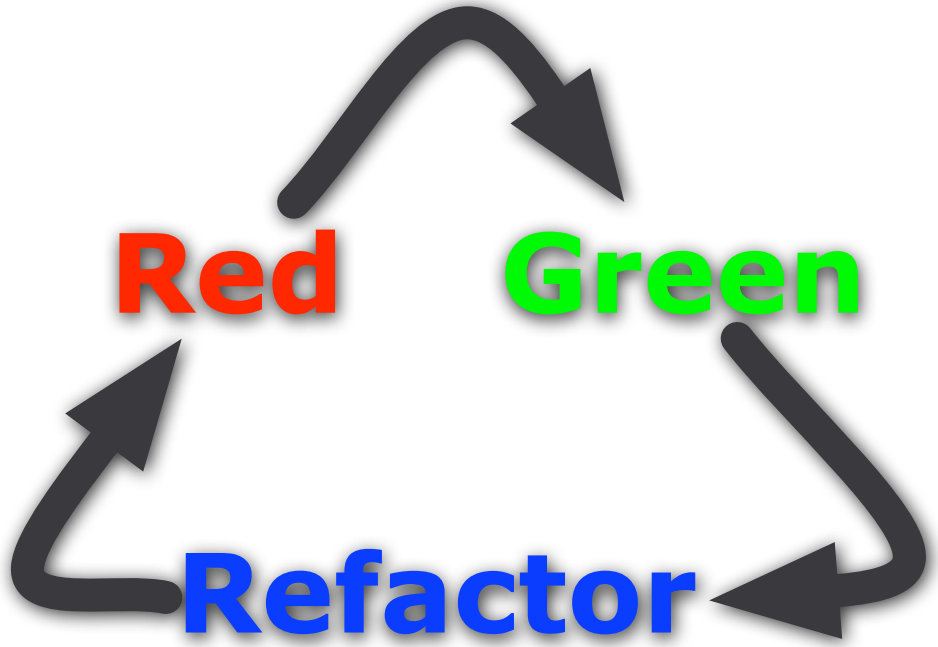
What is refactoring?

You have some software

You want to change it

What is your situation?

Write just
enough code for
the test to pass.



Write a
new test

Remove
duplication,
clean up, etc

Figure out where to start

Check your safety equipment

Tools for refactoring

- Unit testing
  - RSpec
  - Shoulda
  - Test::Unit
- Intergration Testing
  - Webrat
  - Cucumber
  - Selenium
  - Watir
  - Rails Intergration

Fight of the Century

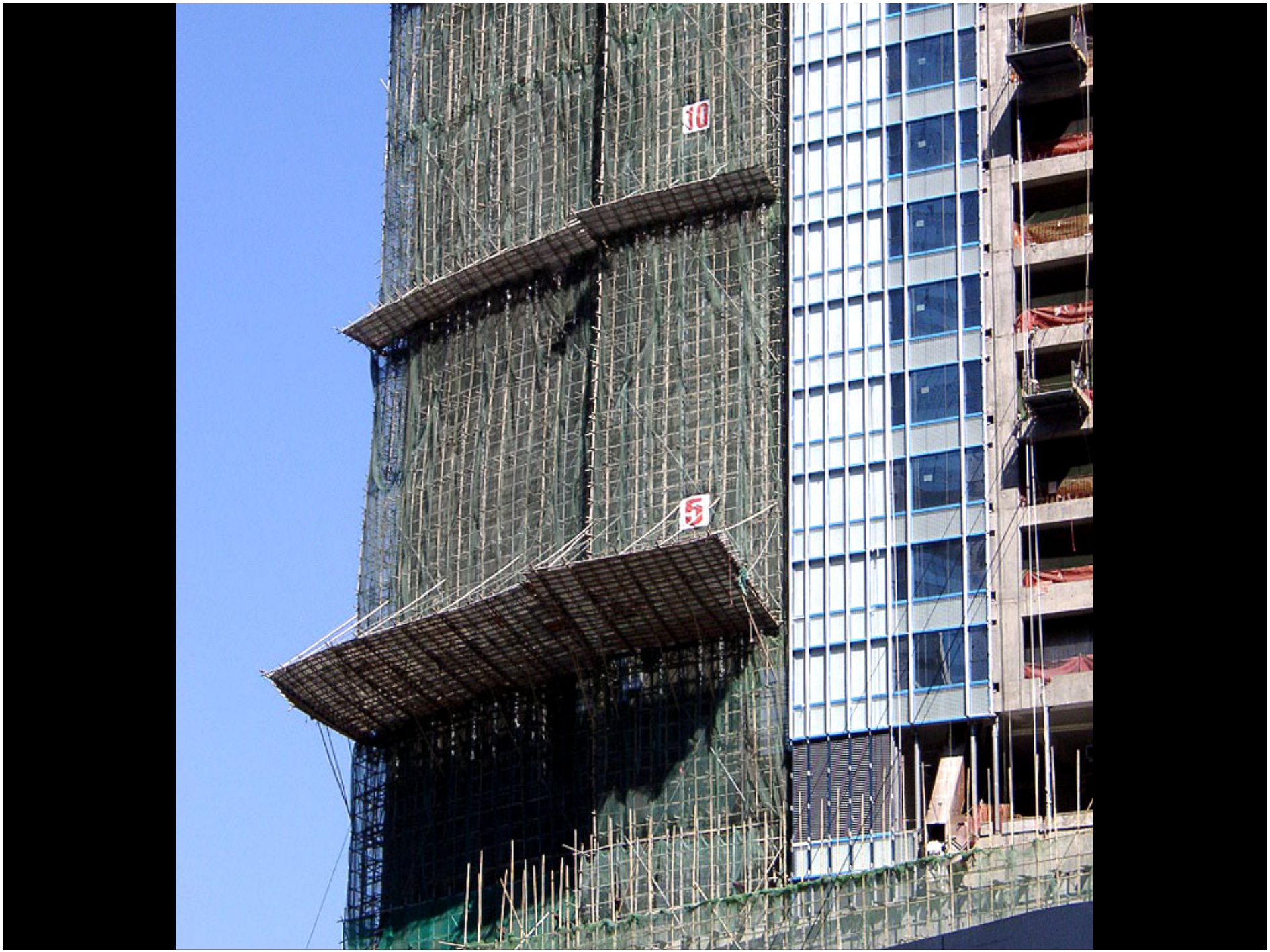
vs

shoulda

RSpec

Tools for refactoring

- Unit testing
  - RSpec
  - Shoulda
  - Test::Unit
- Intergration Testing
  - Webrat
  - Cucumber
  - Selenium
  - Watir
  - Rails Intergration


10
5

Some tests will break

Begin

Types of refactoring

- Rename Method
- Extract Method
- Inline Method
- Extract Parameter
- Inline Parameter
- Move Method
- Extract Superclass
- Extract Module
- Replace Array/Hash with Object

refactoring.com

Rename Method

```
def exit()
  f = FileList[*...
  f.each { |file|
    f.delete!
  }
end
```

```
def cleanup()
  f = FileList[*...
  f.each { |file|
    f.delete!
  }
end
```

Rename Method

WHY?

intention revealing names

misleading names

Extract Method

```
def cleanup()
  @process_one.quit

  f = FileList[*...
  f.each { |file|
    f.delete!
  }

  log.(“shutdown at...
end
```

```
def cleanup()
  @process_one.quit
  delete_files
  log.(“shutdown at...
end

def delete_files
  f = FileList[*...
  f.each { |file|
    f.delete!
  }
```

Extract Method

HINTS

Blank lines in your code

Different levels of abstraction

Extract Method

WHY?

Intention not immediately clear

Method is too large

Duplication across methods

Inline Method

```
def cleanup()
  quit_process
  delete_files
  log_it
end

def log_it
  log.(“shutdown at...
end
```

```
def cleanup()
  quit_process
  delete_files
  log.(“shutdown at...
end
```

Inline Method

WHY?

- Method is unnecessary
- An extra method reduces clarity
- Different levels of abstraction
- Method is only used in one place

Extract Parameter

```
def post_create
  post :create,
       :game => { :name => "Bob" }
end

def post_create(name)
  post :create,
       :game => { :name => name }
end

def post_create(name=”Bob”)
  post :create,
       :game => { :name => name }
end
```

Extract Parameter

WHY?

You want to reuse the method

It is not general enough

Inline Parameter

```
def post_create(name)
  post :create,
       :game => { :name => name }
end

def post_create(name="Bob")
  post :create,
       :game => { :name => name }
end

def post_create
  post :create,
       :game => { :name => "Bob" }
end
```

Inline Parameter

WHY?

Parameter never varies

Method is over generalized

Move Method

```
class HourlyEmployee
  ...
  def update_time_card(hours)
    if hours > 8
      @time_card.regular_hours += 8
      @time_card.overtime_hours += hours - 8
    else
      @time_card.regular_hours += hours
    end
  end
  ...
end
```

Move Method

```
class HourlyEmployee
  ...
  def update_time_card(hours)
    @time_card.update(hours)
  end
  ...
end
```

Move Method

WHY?

Behavior is in the wrong place

Tell, Don’t Ask

Extract Superclass

```
class HourlyEmployee
  def pay ...
  def calculate_pay ...
  def clock_in ...
  def clock_out ...
end
```

```
class Employee
  def pay
end

class HourlyEmployee <
                Employee
  def calculate_pay ...
  def clock_in ...
  def clock_out ...
end
```

Extract ~~Superclass~~ Module

```
class HighSchoolTranscript
  def send
  def delivered?
  def create
end
```

Extract ~~Superclass~~ Module

```
module TranscriptDelivery
  def send ...
  def delivered? ...
end

class HighSchoolTranscript
  include TranscriptDevlivery

  def create ...
end
```

Extract ~~Superclass~~ Module

WHY?

Common abstracted behavior

Easy to extend your system

Open Close Principle

Replace Hash with Object

```
h = {}
h[“Reds”] = 14
h[“Cardinals”] = 35
h[“Pirates”] = 9
```

what’s the average?

```
h = HomeRuns.new
h.add_team(“Reds”, 14)
h.add_team(“Cardinals”, 35)
h.add_team(“Pirates”, 9)
h.average
```

Replace Hash with Object

```
h = HomeRuns.new
h.add_team("Reds", 14)
h.add_team("Cardinals", 35)
h.add_team("Pirates", 9)
h.average
```

```
class HomeRuns
  def initialize
    @teams = {}
  end
  def average ...
  def add_team(team, score)
end
```

Replace Hash with Object

WHY?

Provide for better readability

Add behavior to data structure

... many, many more

- refactoring.com
- Book: Refactoring, Martin Fowler (AW)
- Book: Refactoring, Ruby Edition (August 09)
- Book: Small Talk Best Practice Patterns, Kent Beck