REFACTORING FOR MACHINE DIGNITY

@evanburchard

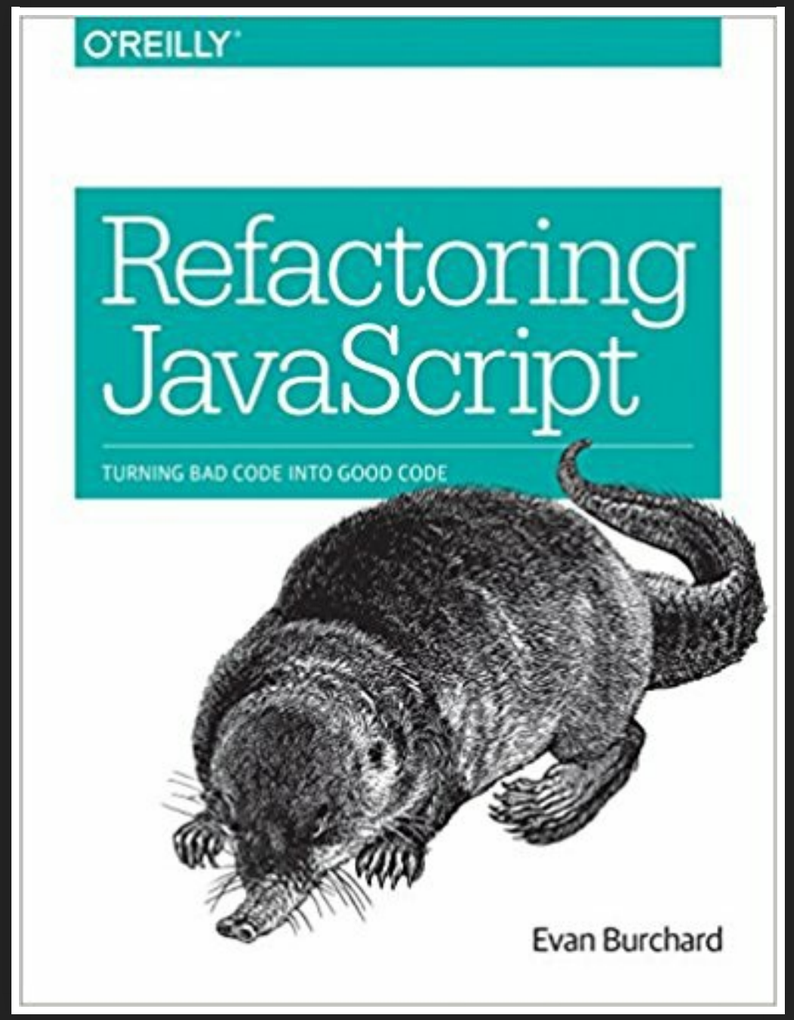
ME

- You're probably here to see Evan Phoenix
- Rubinius Author, Conference Organizer
- That's not me
- I'm Evan Burchard
- I like to write things

THE BOOK I WANTED WHEN I STARTED PROGRAMMING

@evanburchard

THE BOOK I WANTED WHEN I WROTE THE FIRST BOOK



@evanburchard

KEEPING RUBY WEIRD

@evanburchard

OVERVIEW: WHAT

- Refactoring, Safety, and Quality
- Prescriptive Approaches to Quality
- Quality Engines
- Machine Dignity

OVERVIEW: HOW

- A bit of talking
- Working with code

REFACTORING

DEFINITION

Ensuring consistent behavior while changing code.

REFACTORING

WILLIAM OPDYKE'S THESIS (1995)

"Two design constraints needed in refactoring are class invariants and exclusive components."

"In general, it is undecidable whether a predicate is a class invariant or whether a member variable is a component. Since refactorings must be behavior preserving, it is better to mistakenly decide that a predicate is not a class invariant (and so not be able to perform a legal refactoring) than it is to mistakenly decide that a predicate is a class invariant and eventually perform an illegal refactoring."

"Thus, a system for supporting refactorings will use conservative algorithms for deciding whether a predicate is a class invariant or a member variable is a component. The thesis will describe some conservative algorithms for both problems that are based on data-flow analysis."

@evanburchard

"Imagine that a circle is drawn around the parts of a program affected by a refactoring. The behavior as viewed from outside the circle does not change. For some refactorings, the circle surrounds most or all of the program... For other refactorings, the circle covers a much smaller area..."

@evanburchard

REFACTORING

WHAT'S CHANGED SINCE THEN

- our languages
- our tools
- our notion of "safety"

LANGUAGES

- Dynamic
- Multi-paradigmic (Not Always OOP)

TOOLS

- Testing & Methodologies
- Continuous Integration
- Continuous Delivery
- Distributed Version Control

SAFETY

GIVEN THE TECHNOLOGY AND PROCESS DEVELOPMENTS...

in general, we're making changes more quickly, and as Rails developers, tend to make changes based on "Confidence" rather than "Constraints."

ENSURING CONSISTENT BEHAVIOR WHILE CHANGING CODE.

ESTABLISHING TOOLS AND PROCESSES THAT MONITOR AND CREATE PROXIES FOR OUR CONFIDENCE TO SUPPORT CONSISTENT BEHAVIOR WHILE CHANGING CODE.

A FEW PROXIES FOR CONFIDENCE:

- Code Reviews
- Fast and Frequently Run Tests
- Multi-level Testing
- Atomic Commits
- Test Coverage
- Other Code Metrics
- Style Guides & Linters

QUALITY

(aka, the *other* reason we change code)

PRESCRIPTIVE QUALITY

A FEW IDEAS

- DRY
- SOLID
- Follows Patterns {A, B, C...}
- Avoids Antipatterns {A, B, C...}
- Clean Code

LET'S MAKE OUR OWN PRINCIPLES OF QUALITY

I did this exercise with my dog. He used his name.

@evanburchard

@evanburcha

"PIP" SOFTWARE DEV PRINCIPLES

- Push code every day
- Interrogative (?) and Destructive (!) method suffixes
- Pupper-oriented design

@evanburchard

PUPPER-ORIENTED DESIGN MIGHT NOT BE FOR YOU

@evanburchard

LET'S MAKE OUR OWN PRINCIPLES OF QUALITY

You can use your name or just a word you like if your name is long/a secret.

Let's take 3 minutes.

WHAT DID WE COME UP WITH?

ANYONE HAVE TROUBLE WITH PARTICULAR LETTERS?

WE'LL DO THAT AGAIN AT THE END.

SOME "ENGINES" FOR QUALITY

- Livable
- Maintainable
- Not Smelly (has transcended prescriptive)
- Considered Harmful
- Rails Doctrine
- Follows Patterns

THESE COME WITH MORE GENERAL MOTIVATING QUESTIONS. ALSO...

- Less brittle
- Less context dependent
- Less immediately applicable
- Both engines and prescriptive principles are useful

REFACTORING FOR MACHINE DIGNITY

(OUR ENGINE)

@evanburchard

REFACTORING

"Establishing tools and processes that monitor and create proxies for our confidence to support consistent behavior while changing code."

-Me

MACHINE

"An object that runs a program."

-Eric Steinhart

DIGNITY

"The idea that a being has an innate right to be valued, respected, and to receive ethical treatment."

-Wikipedius

DIGNITY FOR *MACHINES* INSTEAD OF *HUMANS*?

@evanburchard

NO

That's just a convenient place to begin our interogation.

Also, it's where most of us spend a lot of our energy.

TWO QUESTIONS WHEN THINKING ABOUT THE DIGNITY OF THE MACHINE:

1. Would you want to act as the machine?
2. To what degree is the machine able to flourish?

WE NEED CONTEXT FOR THESE

OUR LENSES

HOW CAN THIS CODE...

1. ... be more useful to me? (Personal)
2. ... be more useful to others? (Shared)
3. ... impress or inspire someone? (Aesthetic)
4. ... express a considerate worldview? (Civil)

OUR LENSES

In some ways, these are an extension of the old adage of "make it work, make it readable, make it fast," but in each of these contexts, the questions to motivate changes in our code can remain the same:

1. Would you want to act as the machine?
2. To what degree is the machine able to flourish?

Which themselves come from the question: How can we grant our machines dignity in their work?

CIRCLES AND PRECONDITIONS

	Act As Machine	Flourishing
Personal		
Shared		
Aesthetic		
Civil		

LET'S USE THEM

FIRST, IN ADDITION TO OUR LENSES, WE NEED A TEST FRAMEWORK

```
def assert(code_as_string)
  raise 'NOOOOOOO' unless eval(code_as_string)
end
```

@evanburchard

LENS 1: PERSONAL

```
def s(a,b);a+b;end;puts s(2,3)
assert('s(2, 3) == 5')
```

@evanburchard

LENS 2: SHARED

Who are you working with?

LENS 2: SHARED

```
puts 3
puts 3
puts 8
puts 3
puts 3
puts 3
puts 3
puts 3
puts 3
puts 2
puts 3
puts 1
puts 3
puts 3
```

@evanburchard

LENS 2: SHARED

```
def remove_prefix(string)
  if(string[0..6] == 'prefix')
    return string[7..-1]
  else
    return string
  end
end
```

LENS 3: AESTHETIC

Who are you trying to impress?

- Interview?
- Code Review?
- Stack Overflow Participants?
- Blog Audience?
- Obfuscated C Judges?
- JS XK Judges?
- "Easier to reason about"? (Be careful here)

LENS 3: ! GOLF

```
def factorial(number)
  if(number > 1)
    factorial(number - 1) * number
  else
    1
  end
end
```

@evanburchard

LENS 4: CIVIL

```
def blow_up_the_moon
  # We probably shouldn't blow up the moon...
  raise 'moon will blow up when you run this,
  comment this line to run'
  # comment out the line above this one to
  # run the code below
  # keep in mind that this will blow up the moon
  # and the moon provides the Earth with protection
  # against asteroids

  code_that_actually_blows_up_the_moon
end
```

@evanburchard

QUALITY AT ALL LEVELS

@evanburchard

CONTEXTS FOR LENS CONFLICTS AND INTERPLAY

- Writing code
- Code Reviews
- Interviews
- Competition
- Open Source
- All the lenses are moving targets

QUALITY DOESN'T STOP AT PERSONAL CODE

- Adapt Your Code for Others
- Learn About Various Aesthetics
- Decide on things you won't do

@evanburchard

YOU CONTROL WHAT CIVIL CODE IS OPERATIONALIZED

We operationalize the company's worldview, but in arguing for higher quality, we often stop at the aesthetic.

Often, their worldview is simply "we should be growing" with some other stuff tacked on.

As DHH said, this can be uninspiring.

Destructive is worse than uninspiring, and is common.

THE AGILE MANIFESTO CEDES CONTROL OF THE CIVIL ASPECTS

It argues well for a quality process, but kicks off as:

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

This assumed primacy of the customer downplays the our role.

@evanburchard

YOU HAVE OTHER OPTIONS

- Adopt the "Ethical Design Manifesto," "Inclusive Design," "GNU Manifesto" or make up your own guiding principles
- Avoid (or quit) implementations, projects, teams, and companies that don't support your values.
- Support team members who are wronged.
- Offer a sliding scale for clients who support your values.
- Support and argue for your company supporting open source devlopers (with money).
- Understand where the money comes from and goes.

THIS IS BECOMING MORE IMPORTANT

Machine Learning makes our role very obvious
because it gives so much autonomy.

Web apps have the same capacity.

REGARDLESS OF THE CONFLICTS

PLEASE DON'T BLOW UP THE MOON

@evanburchard

BUT REALLY

Consider using quality, and possibly machine dignity,
as motivations for doing good work.

TECHNIQUES TO BE COMFORTABLE WITH

- Naming and renaming things
- Extracting
- Inlining
- Declarative over Imperative

SOME RUBY DESIGN CONSIDERATIONS

- DSLs
- Passing nil
- Variables out of nowhere
- Deep Inheritance
- Impractical Grep via Metaprogramming

But metaprogramming for reflection is great

LET'S REVISE OUR PRINCIPLES OF QUALITY

KEEP A FEW THINGS IN MIND

- Prescription vs. Engine
- Personal Lens
- Shared Lens
- Aesthetic Lens
- Civil Lens
- Would you want to perform as the machine?
- How can you promote flourishing?

Take 2 minutes

ANY DIFFERENT?

THANKS!

AND AGAIN, PLEASE DON'T BLOW UP THE MOON

@evanburchard

RESOURCES

github.com/EvanBurchard/refactoring_for_machine_dignity

BONUS:

CHARACTERIZATION TESTS

Assert that the code doesn't do anything of value:

```
expect(result).to eq(nil)
```

Let the test tell you, NO I AM NOT nil, I'm "Whatever"

Then fill in your test with that. Now, setup and suite running time are your only testing problems.