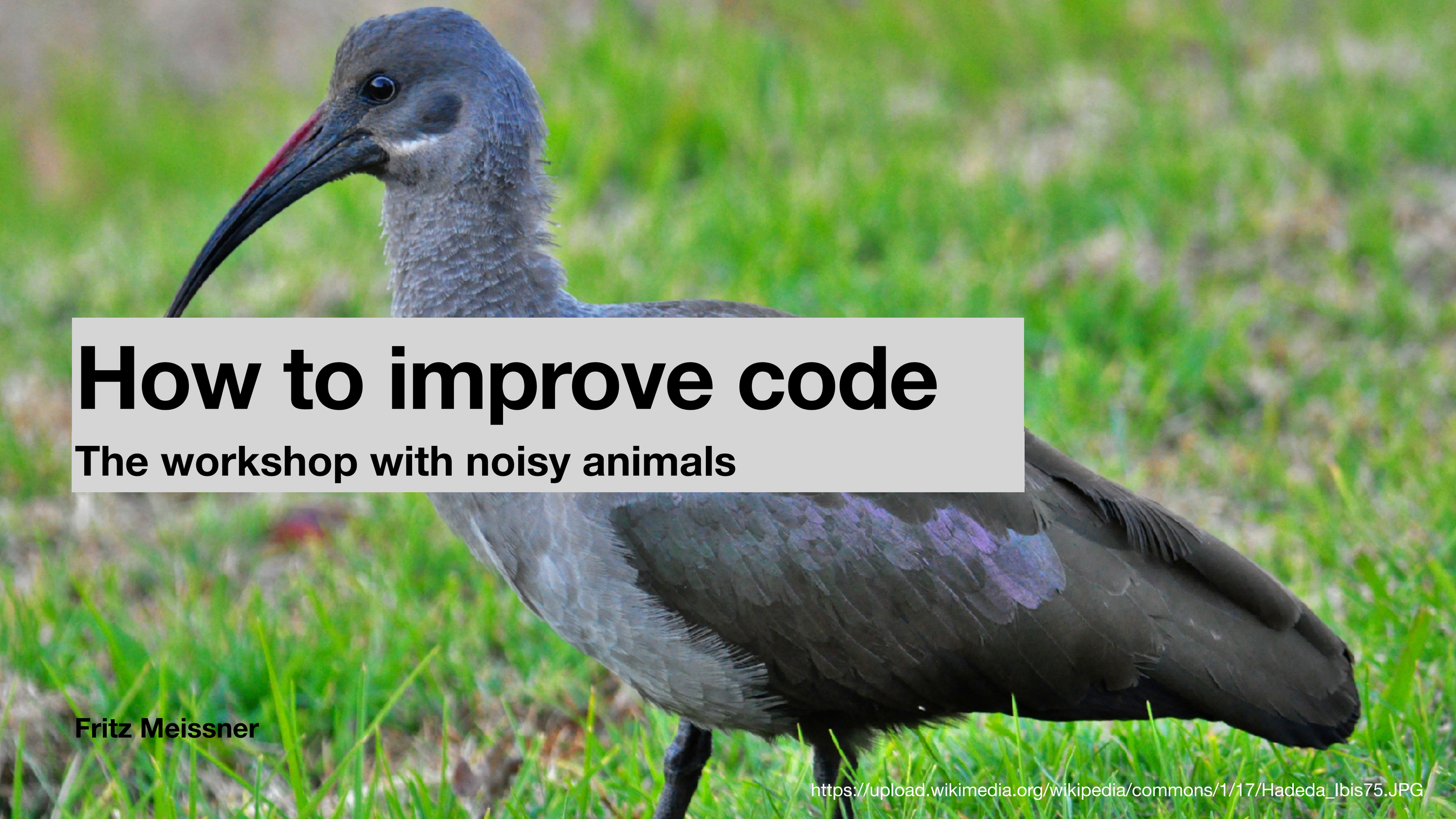
Icebreaker: which of these is you?

1. My boss said I should attend this workshop
2. I don't feel safe changing the code I work with
3. Some people make beautiful code; how do they do it?
4. I love refactoring and everyone else should too!
5. When people use the word "refactoring" it always seems like we're about to waste time



How to improve code

The workshop with noisy animals

Fritz Meissner

https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

What to expect

1. Theory: What is good/bad code? How does it help?
2. Theory - Practice - Review - Theory - Practice - Review
3. Where to go next
4. How to run your own workshop

https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

thoughtbot

What is good code?

Beyond "it works"

Two qualities beyond "it works"

1. Good code describes its purpose

Analogy

https://stockcake.com/i/builders-reaching-skyward_1675372_1223105

"The people in the stands look down on the men wearing a familiar orange with stripes. They stand ready for their challenge in pods of two, three, or four. The skies are blue with few clouds, the weather will offer no obstacles today."

Bad description!

"The people in the stands look down on the men wearing a familiar orange with stripes. They stand ready for their challenge in pods of two, three, or four. The skies are blue with few clouds, the weather will offer no obstacles today."

Better description

"Construction workers on the ground look at their section of a massive building site. Other workers wait on scaffolding for the material they need from cranes. In the distance more cranes indicate the presence of other building efforts."

Code should be a good description

- focus on the right details
- use helpful terms from the domain


https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

This code is a bad description

2. Good code is easy to change

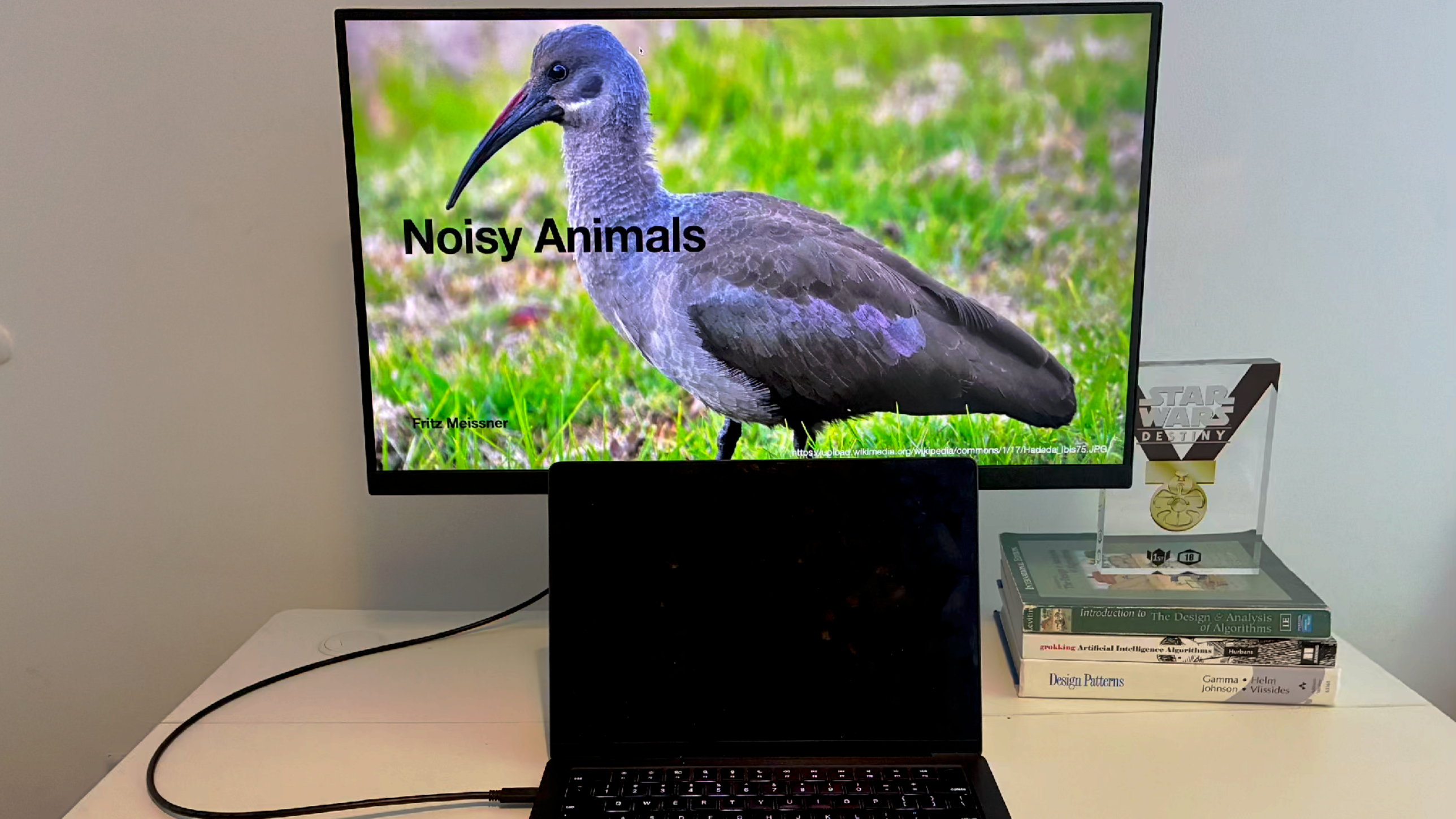

https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

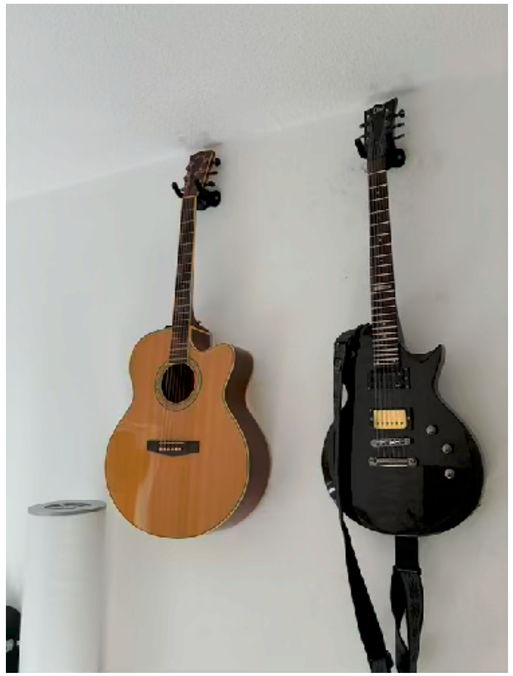
This code is hard to change

What can we do about it?

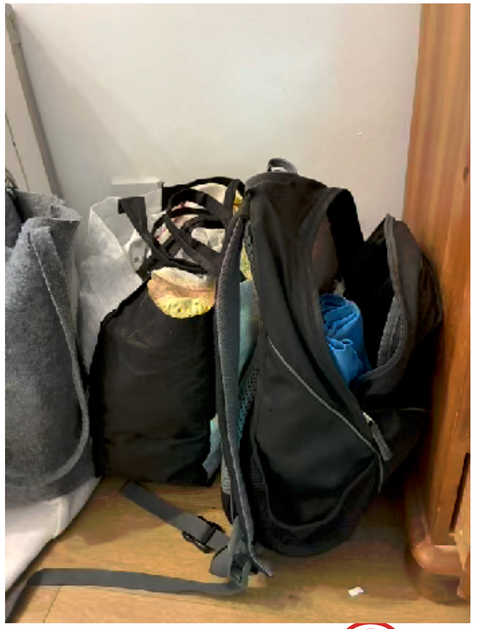
Analogy with tidying


EVERSION
ALASTAIR
REYNOLD
deuter


Noisy Animals
Fritz Meissner
https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG
STAR WARS
DESTINY
Introduction to The Design & Analysis of Algorithms
grokking Artificial Intelligence Algorithms
Design Patterns
Gamma • Helm
Johnson • Vlissides



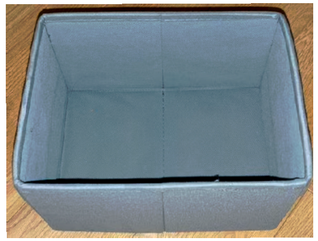

THE LEAN STARTUP ERIC RIES
EVERSION
ALASTAIR REYNOLDS
KIM STANLEY ROBINSON
ATLAS OF THE HEART

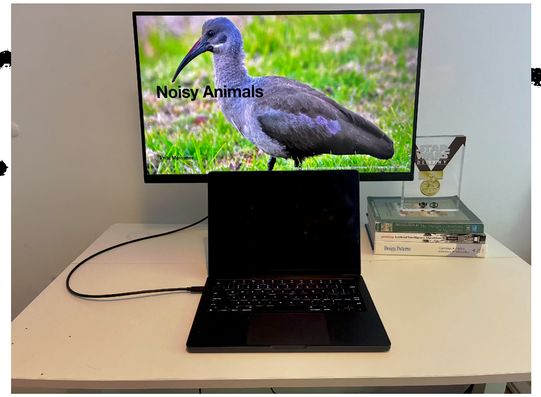


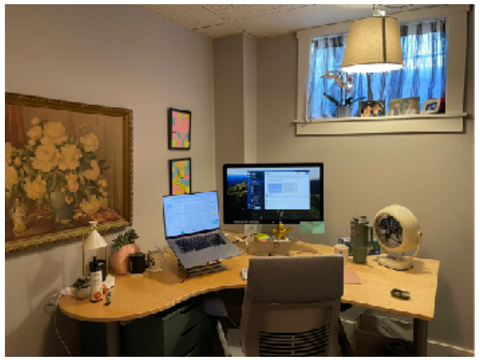
thoughtbot

Takes some tools and techniques

Tools and techniques are widely applicable



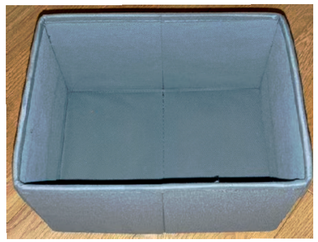




https://thoughtbot.com/blog/what-do-our-workspaces-look-like

thoughtbot

Tools and techniques are widely applicable



thoughtbot

Widely applicable code tidying

https://github.com/bablubambal/All_logo_and_pictures

thoughtbot

Widely applicable code tidying

Refactoring

https://github.com/bablubambal/All_logo_and_pictures

thoughtbot

Forget the fancy word for now, let's just fix bad code

What are we aiming for?

1. make the code describe its purpose better
2. make the code easier to change

What hides purpose and prevents change?

What don't you like?


https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

Some examples of problem code

- flag argument methods
- different syntax for the same logic
- domain concepts far apart in the code
- duplicate code
- complicated conditional

These problems are not unique!

- flag argument methods
- different syntax for the same logic
- domain concepts far apart in the code
- duplicate code
- complicated conditional

thoughtbot

Some examples of problem code

- flag argument methods
- different syntax for the same logic
- domain concepts far apart in the code
- duplicate code
- complicated conditional

Fix that conditional!

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif %w[cat dog leopard].include?(species)
    puts animal_noise
  end
end
```

thoughtbot

Fix that conditional!

- https://github.com/thoughtbot/noisy-animals-kata
- git clone/checkout;
- cd ruby; bundle
- Tests must pass without change! Run `rspec`
- Turn off AI!
- NO SUPPORT FOR OTHER ANIMALS


https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

Complicated conditional review

Before

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif %w[cat dog leopard].include?(species)
    puts animal_noise
  end
end
```

After

```
def make_noise(loud: true)
  if is_animal && loud
    2.times { puts animal_noise }
  elsif is_animal && !loud
    puts animal_noise
  elsif is_bird && loud
    make_bird_noise(true)
  elsif is_bird && !loud
    make_bird_noise(false)
  end
end
```

Is this better?

- better description?
- easier to change?

How (I) got there

Tidying loop

- identify a problem
- make a small change
- run the tests
- commit (or revert)

What to change: looking for consistency

- find something almost consistent
- eliminate difference
  - maintain logical equivalence

Almost consistent: names for boolean expressions

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif %w[cat dog leopard].include?(species)
    puts animal_noise
  end
end
```



Unnamed boolean expression

```ruby
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif %w[cat dog leopard].include?(species)
    puts animal_noise
  end
end
```

thoughtbot

More consistency: name expression with function

```
        2.times { puts animal_noise }
      end
      make_bird_noise(true) if is_bird
    elsif %w[cat dog leopard].include?(species)
    elsif is_animal
      puts animal_noise
    end
  end


  def is_animal
    %w[cat dog leopard].include?(species)
  end
end
```

thoughtbot

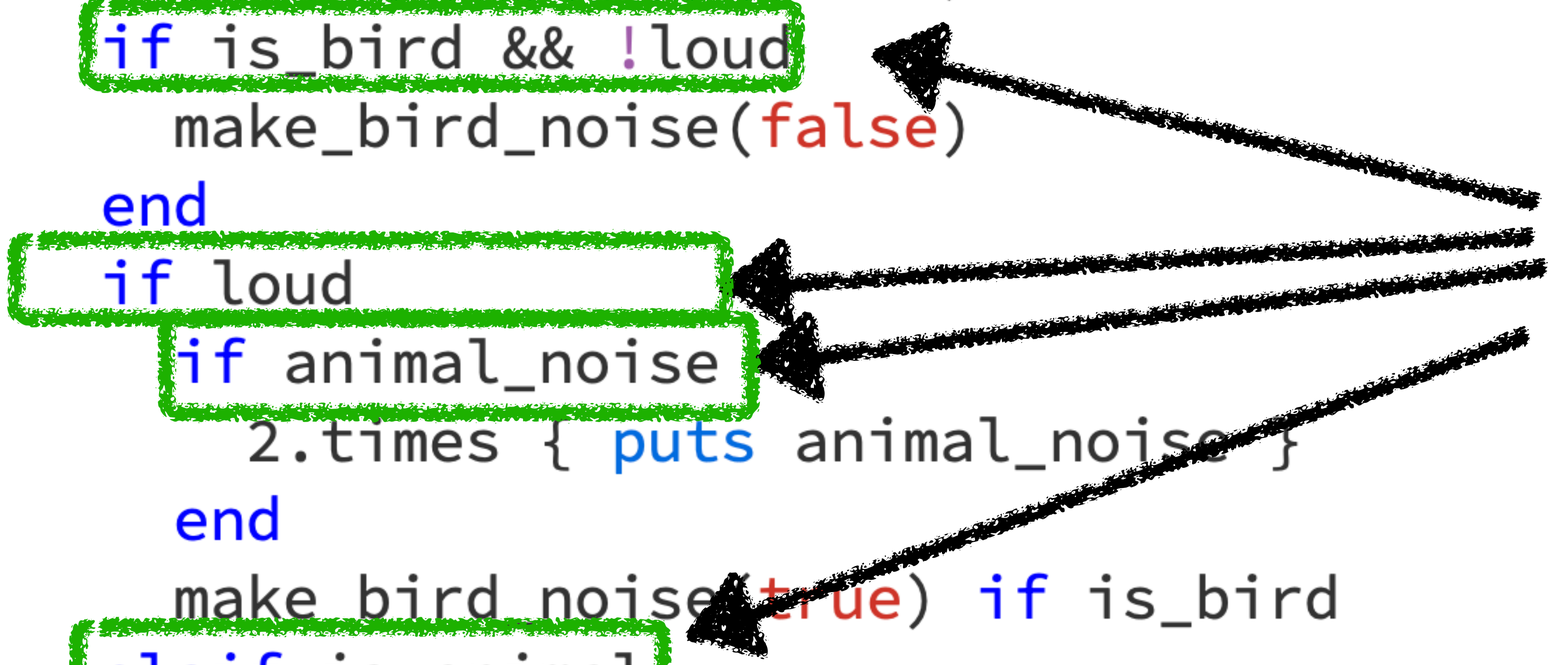
Result

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif is_animal
    puts animal_noise
  end
end
```

Almost consistent

```ruby
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif is_animal
    puts animal_noise
  end
end
```

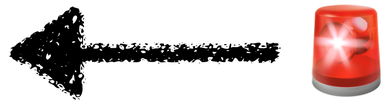
```ruby
if a_condition
  # do something
```



thoughtbot

Inconsistent if syntax

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    make_bird_noise(true) if is_bird
  elsif is_animal
    puts animal_noise
  end
end
```



thoughtbot

More consistency: if syntax

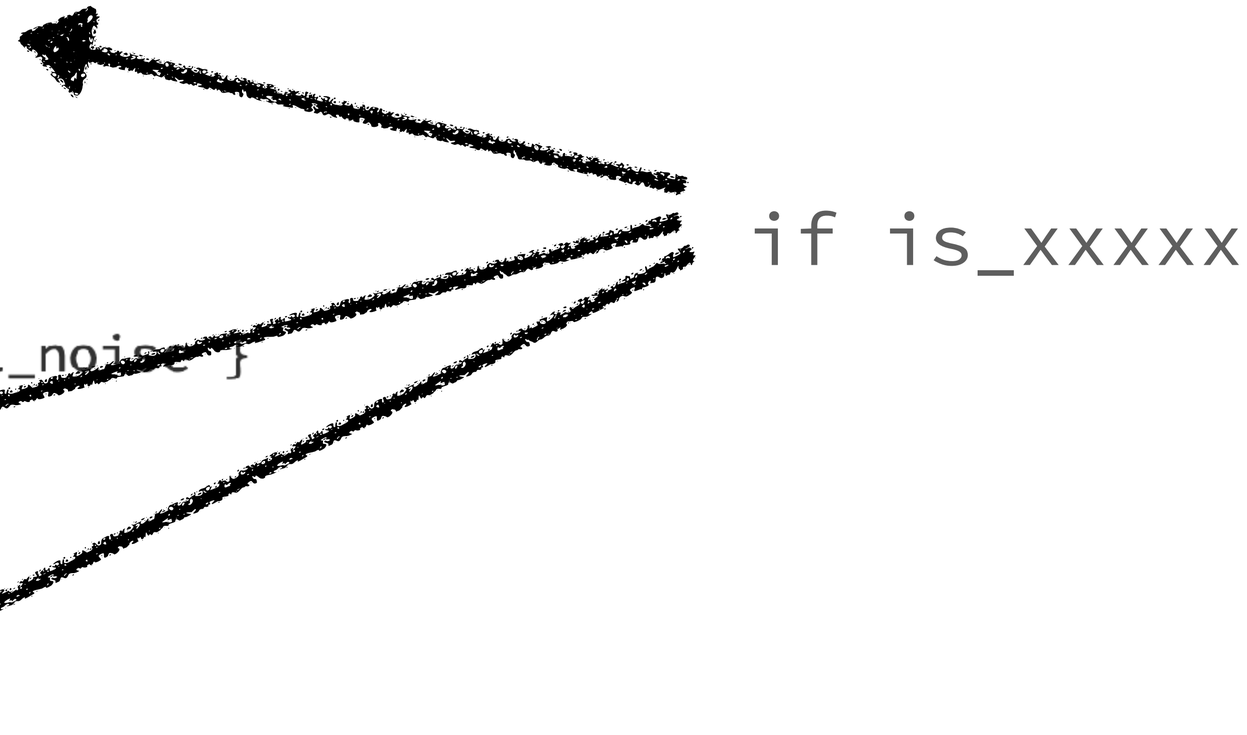
```
      if animal_noise
        2.times { puts animal_noise }
      end
      make_bird_noise(true) if is_bird
      if is_bird
        make_bird_noise(true)
      end
    elsif is_animal
      puts animal_noise
    end
```

Result

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    if is_bird
      make_bird_noise(true)
    end
  elsif is_animal
    puts animal_noise
  end
end
```

Almost consistent

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    if is_bird
      make_bird_noise(true)
    end
  elsif is_animal
    puts animal_noise
  end
end
```



thoughtbot

Almost consistent

```ruby
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    if is_bird
      make_bird_noise(true)
    end
  elsif is_animal
    puts animal_noise
  end
end
```

thoughtbot

animal_noise vs is_animal

```ruby
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if animal_noise
      2.times { puts animal_noise }
    end
    if is_bird
      make_bird_noise(true)
    end
  elsif is_animal
    puts animal_noise
  end
end
```

thoughtbot

animal_noise vs is_animal

```ruby
def is_animal
  %w[cat dog leopard].include?(species)
end

def animal_noise
  {
    'cat' => 'meow',
    'dog' => 'woof',
    'leopard' => 'growl'
  }[species]
end
```

is_animal == animal_noise

thoughtbot

More consistency: use is_animal function

```
if loud
   if animal_noise
   if is_animal
     2.times { puts animal_noise }
   end
   if is_bird
```

Result

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud
    if is_animal
      2.times { puts animal_noise }
    end
    if is_bird
      make_bird_noise(true)
    end
  elsif is_animal
    puts animal_noise
  end
end
```

Reduce nesting

```
if loud
  if is_animal
    2.times { puts animal_noise }
  end
  if is_bird
    make_bird_noise(true)
  end
```

Reduce nesting

```
if loud && is_animal
  2.times { puts animal_noise }
elsif loud && is_bird
  make_bird_noise(true)
```

thoughtbot

Result

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif is_animal
    puts animal_noise
  end
end
```

Almost consistent

```
def make_noise(loud: true)
  if is_bird && !loud                is_bird, loud
    make_bird_noise(false)
  end
  if loud && is_animal                loud, is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird               loud, is_bird
    make_bird_noise(true)
  elsif is_animal
    puts animal_noise
  end
end
```

thoughtbot

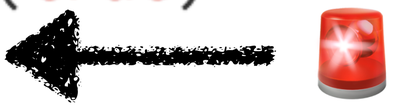
Almost consistent

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif is_animal
    puts animal_noise
  end
end
```

is_bird, loud

loud, is_animal

loud, is_bird



🚨 **loud????**

thoughtbot

More consistent: show implicit boolean

```
if loud && is_animal
  2.times { puts animal_noise }
elsif loud && is_bird
  make_bird_noise(true)
elsif is_animal
elsif !loud && is_animal
  puts animal_noise
end
```

thoughtbot

Result

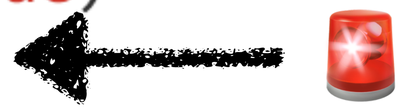
```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  end
end
```

Almost consistent

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud && is_animal                loud, is_xxxx
    2.times { puts animal_noise }
  elsif loud && is_bird               loud, is_xxxx
    make_bird_noise(true)
  elsif !loud && is_animal            loud, is_xxxx
    puts animal_noise
  end
end
```

Almost consistent

```
def make_noise(loud: true)
  if is_bird && !loud
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  end
end
```



🚨 **is_xxxx first**

```
                                  loud, is_xxxx
                                  loud, is_xxxx
                                  loud, is_xxxx
```

thoughtbot

More consistent: swap booleans

```
  def make_noise(loud: true)
    if is_bird && !loud
    if !loud && is_bird
      make_bird_noise(false)
    end
    if loud && is_animal
      2.times { puts animal_noise }
    elsif loud && is_bird
      make_bird_noise(true)
    elsif !loud && is_animal
      puts animal_noise
    end
  end
```

thoughtbot

Result

```
def make_noise(loud: true)
  if !loud && is_bird
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  end
end
```

loud, is_xxxx

loud, is_xxxx

loud, is_xxxx

loud, is_xxxx

Almost consistent

```
def make_noise(loud: true)
  if !loud && is_bird
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  end
end
```

thoughtbot

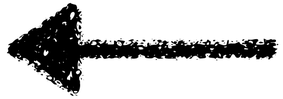
Almost consistent

```
def make_noise(loud: true)
  if !loud && is_bird
    make_bird_noise(false)
  end
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  end
end
```

thoughtbot

More consistent: combine conditionals

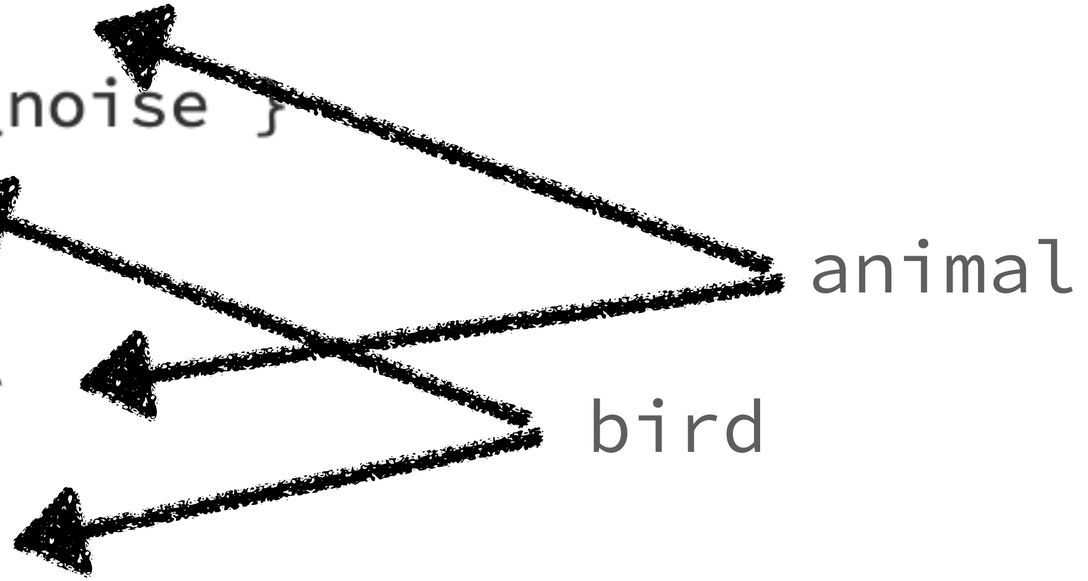
```
def make_noise(loud: true)
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  elsif !loud && is_bird
    make_bird_noise(false)
  end
end
```



Bonus: readability

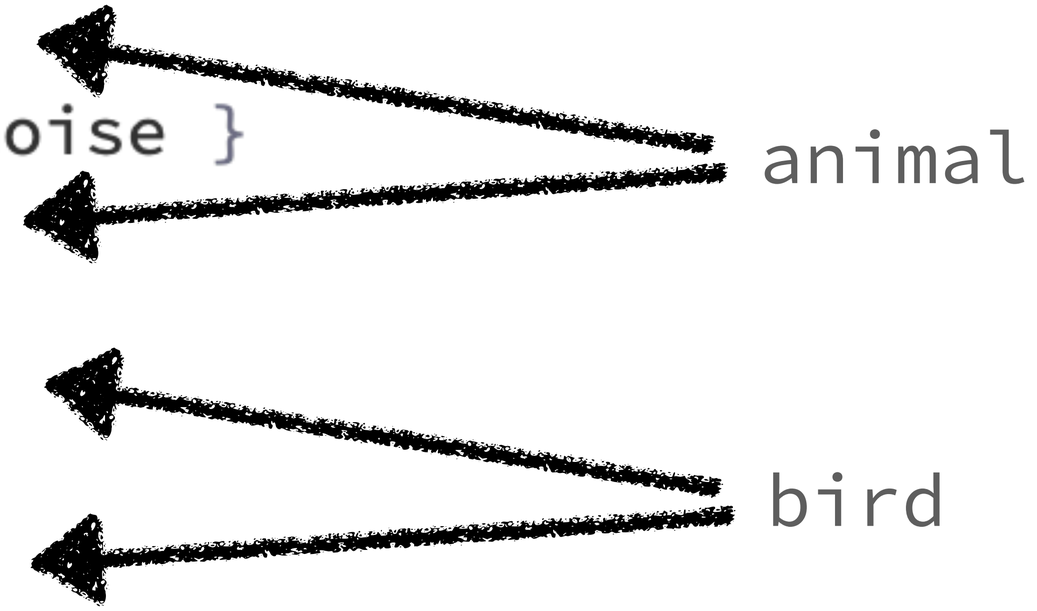
Bring animal/bird concepts together

```
def make_noise(loud: true)
  if loud && is_animal
    2.times { puts animal_noise }
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_animal
    puts animal_noise
  elsif !loud && is_bird
    make_bird_noise(false)
  end
end
```



Bring animal/bird concepts together

```
def make_noise(loud: true)
  if loud && is_animal
    2.times { puts animal_noise }
  elsif !loud && is_animal
    puts animal_noise
  elsif loud && is_bird
    make_bird_noise(true)
  elsif !loud && is_bird
    make_bird_noise(false)
  end
end
```



thoughtbot

Order of && affects readability

```
def make_noise(loud: true)
  if loud && is_animal
  if is_animal && loud
    2.times { puts animal_noise }
  elsif !loud && is_animal
  elsif is_animal && !loud
    puts animal_noise
  elsif loud && is_bird
  elsif is_bird && loud
    make_bird_noise(true)
  elsif !loud && is_bird
  elsif is_bird && !loud
    make_bird_noise(false)
  end
end
```

We got there!

```
def make_noise(loud: true)
  if is_animal && loud
    2.times { puts animal_noise }
  elsif is_animal && !loud
    puts animal_noise
  elsif is_bird && loud
    make_bird_noise(true)
  elsif is_bird && !loud
    make_bird_noise(false)
  end
end
```

Fixed conditional!

- git checkout origin/after-conditional-clean

Compare with your notes

- what problems did you see at the start?
- what do you see now?

Code smells

"we have learned to look for certain structures in the code that suggest—sometimes, scream for—the possibility of refactoring."

Martin Fowler and Kent Beck

Familiar smell: complex conditional

"Much of the power of programs comes from their ability to implement conditional logic—but, sadly, much of the complexity of programs lies in these conditionals"

Martin Fowler and Kent Beck

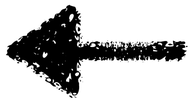
Next code smell: flag arguments

"[Flag argument] functions complicate the process of understanding what function calls are available and how to call them"

Martin Fowler and Kent Beck

Flag arguments

```
def make_noise(loud: true)
  if is_animal && loud
    2.times { puts animal_noise }
  elsif is_animal && !loud
    puts animal_noise
  elsif is_bird && loud
    make_bird_noise(true)
  elsif is_bird && !loud
    make_bird_noise(false)
  end
end
```



thoughtbot

Flag arguments are bad

- reader expects one behaviour per function
- flag arguments = 2+ behaviours per function
- boolean flags are worse: what does `true` mean?

Fix that smell! Break `make_bird_noise` into two


https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

Flag argument method removed

```ruby
def make_noise(loud: true)
  if is_animal && loud
    2.times { puts animal_noise }
  elsif is_animal && !loud
    puts animal_noise
  elsif is_bird && loud
    2.times { puts bird_noise }
  elsif is_bird && !loud
    puts bird_noise
    raise 'no such thing' if species == 'hadedah'
  end
end
```

Next steps

- address speculative generality: animal vs bird
- hadedah vs everything else!
- make it easy to add a mouse


https://upload.wikimedia.org/wikipedia/commons/1/17/Hadeda_Ibis75.JPG

Examples after tidying

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    raise 'no such thing!' if species == 'hadedah' && !loud

    noise = {
      'cat' => 'meow',
      'dog' => 'woof',
      'leopard' => 'growl',
      'owl' => 'hoot',
      'eagle' => 'caw',
      'hadedah' => 'squawk'
    }[species]

    n = loud ? 2 : 1
    n.times { puts noise }
  end
end
```

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    {
      'dog' => [-> { puts 'woof' }, -> {puts "woof\nwoof"}],
      'leopard' => [-> { puts 'growl' }, -> {puts "growl\ngrowl"}],
      'cat' => [-> { puts 'meow' }, -> {puts "meow\nmeow"}],
      'eagle' => [-> { puts 'caw' }, -> {puts "caw\ncaw"}],
      'owl' => [-> { puts 'hoot' }, -> {puts "hoot\nhoot"}],
      'hadedah' => [-> { puts 'squawk' }, -> {raise "no such thing!"}]
    }[species] => [low_volume, high_volume]
    loud ? high_volume.call : low_volume.call
  end
end
```

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    {
      'dog' => [-> { puts 'woof' }, -> {puts "woof\nwoof"}],
      'leopard' => [-> { puts 'growl' }, -> {puts "growl\ngrowl"}],
      'cat' => [-> { puts 'meow' }, -> {puts "meow\nmeow"}],
      'eagle' => [-> { puts 'caw' }, -> {puts "caw\ncaw"}],
      'owl' => [-> { puts 'hoot' }, -> {puts "hoot\nhoot"}],
      'hadedah' => [-> {raise "no such thing!"},-> { puts "squawk\nsquawk" }],
      'mouse' => [-> { puts '' }, -> { puts '' }]
    }[species] => [low_volume, high_volume]
    loud ? high_volume.call : low_volume.call
  end
end
```

thoughtbot

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    voice = {
      'cat' => Voice.new('meow'),
      'dog' => Voice.new('woof'),
      'leopard' => Voice.new('growl'),
      'owl' => Voice.new('hoot'),
      'eagle' => Voice.new('caw'),
      'hadedah' => NoQuietVoice.new('squawk')
    }[species]

    loud ? voice.loud : voice.quiet
  end
end
```

```ruby
class Voice
  attr_accessor :noise

  def initialize(noise)
    @noise = noise
  end

  def quiet = puts noise

  def loud = 2.times { puts noise }
end

class NoQuietVoice < Voice
  def initialize(noise)
    super(noise)
  end

  def quiet = raise 'no such thing!'
end
```

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    loud ? voice.loud : voice.quiet
  end

  def voice
    @voice ||= {
      'dog' => Voice['woof'],
      'leopard' => Voice['growl'],
      'cat' => Voice['meow'],
      'eagle' => Voice['caw'],
      'owl' => Voice['hoot'],
      'hadedah' => WithoutQuiet.new(Voice['squawk'])
    }[species]
  end
end
```

```ruby
Voice = Data.define(:noise) do
  def quiet = puts noise

  def loud = 2.times { puts noise }
end

class WithoutQuiet < SimpleDelegator
  def quiet = raise "no such thing!"
end
```

thoughtbot

```ruby
class NoisyAnimal
  attr_reader :species

  def initialize(species)
    @species = species
  end

  def make_noise(loud: true)
    loud ? voice.loud : voice.quiet
  end

  def voice
    @voice ||= {
      'dog' => Voice['woof'],
      'leopard' => Voice['growl'],
      'cat' => Voice['meow'],
      'eagle' => Voice['caw'],
      'owl' => Voice['hoot'],
      'hadedah' => WithoutQuiet.new(Voice['squawk']),
      'mouse' => SingleVolume.new(Voice[''])
    }[species]
  end
end
```

```ruby
Voice = Data.define(:noise) do
  def quiet = puts noise

  def loud = 2.times { puts noise }
end

class WithoutQuiet < SimpleDelegator
  def quiet = raise "no such thing!"
end
```

```ruby
class SingleVolume < SimpleDelegator
  def loud = quiet
end
```

thoughtbot

Where to next?

Tidying loop is easily optimised

identify a problem → code smell knowledge

make a small change → refactoring techniques

run the tests → better tests

run the tests → how to run tests easily

commit or revert → git technique

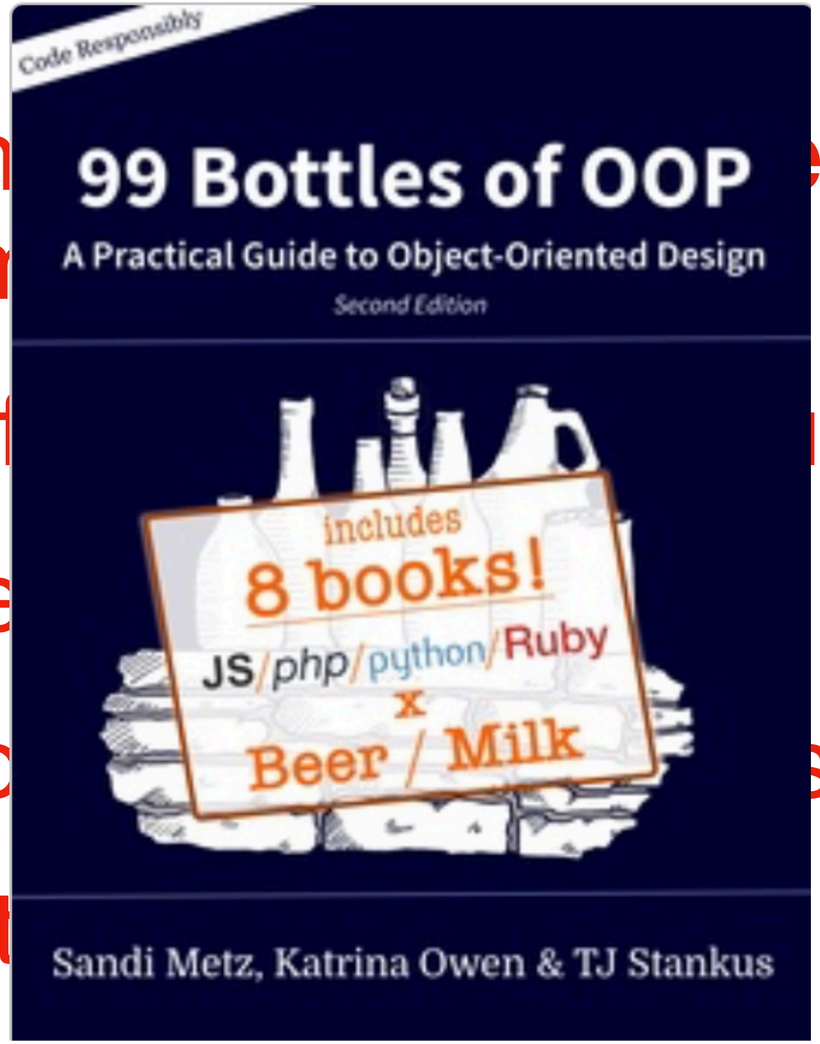
Options for improved tidying

identify a problem →

make a small change →

run the tests →

commit or revert →



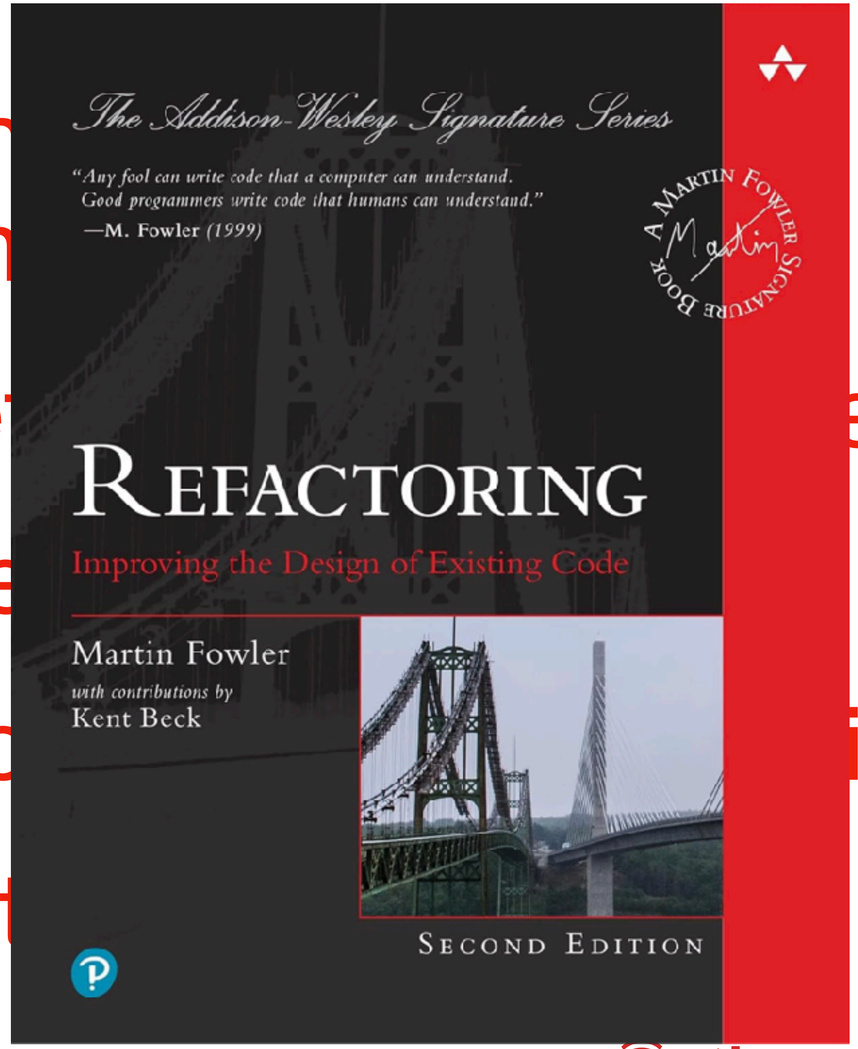
Options for improved tidying

identify a problem

make a small change

run the tests

commit or revert



thoughtbot

Catchphrases to understand

- tidy first
- make the change easy, then make the easy change
- red/green refactor
- flocking rules
- open for extension, closed for modification

Running your own workshop

How to run your own Noisy Animals workshop

- https://github.com/thoughtbot/noisy-animals-kata
- simple workshop exercises follow
- kata = repeat exercise

thoughtbot

Noisy Animals vs problem domain

- Bad code hides domain, good code reveals it
- ask domain questions
- try tidy
- ask domain questions again

Noisy Animals vs change

- Bad code is difficult to change, but we can make it easier
- ask where to add mouse
- clean up
- ask where to add mouse again

Noisy Animals vs tests

- More test runs = more speed and accuracy
- try as few test runs as possible
- or try as often as you can
- challenge: test with every single line change
  - (requires new techniques)

Noisy Animals vs code smells

- Fixing problems reveals other problems
- look for problems
- fix a code smell
- look for problems again
  - (you'll see more)

@iftheshoefritz

iftheshoefritz.com

Thank you!

thoughtbot

Questions?