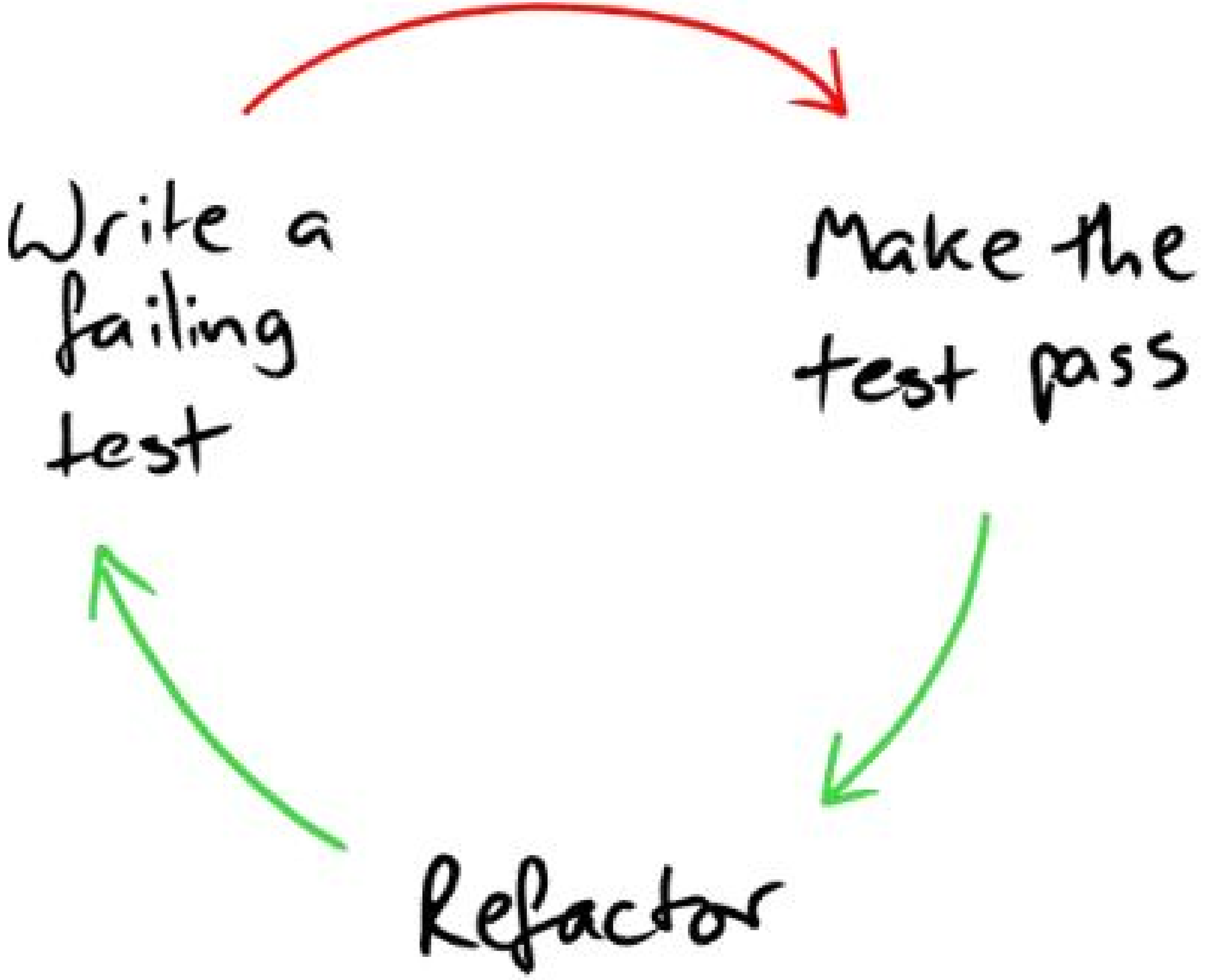

# Test-driven development

# How many tests do we have to write?


Write a failing test
Make the test pass
Refactor


https://www.flickr.
com/photos/matvey_andreyev/3656451273

# Start with the simplest failing test

IRON HACK

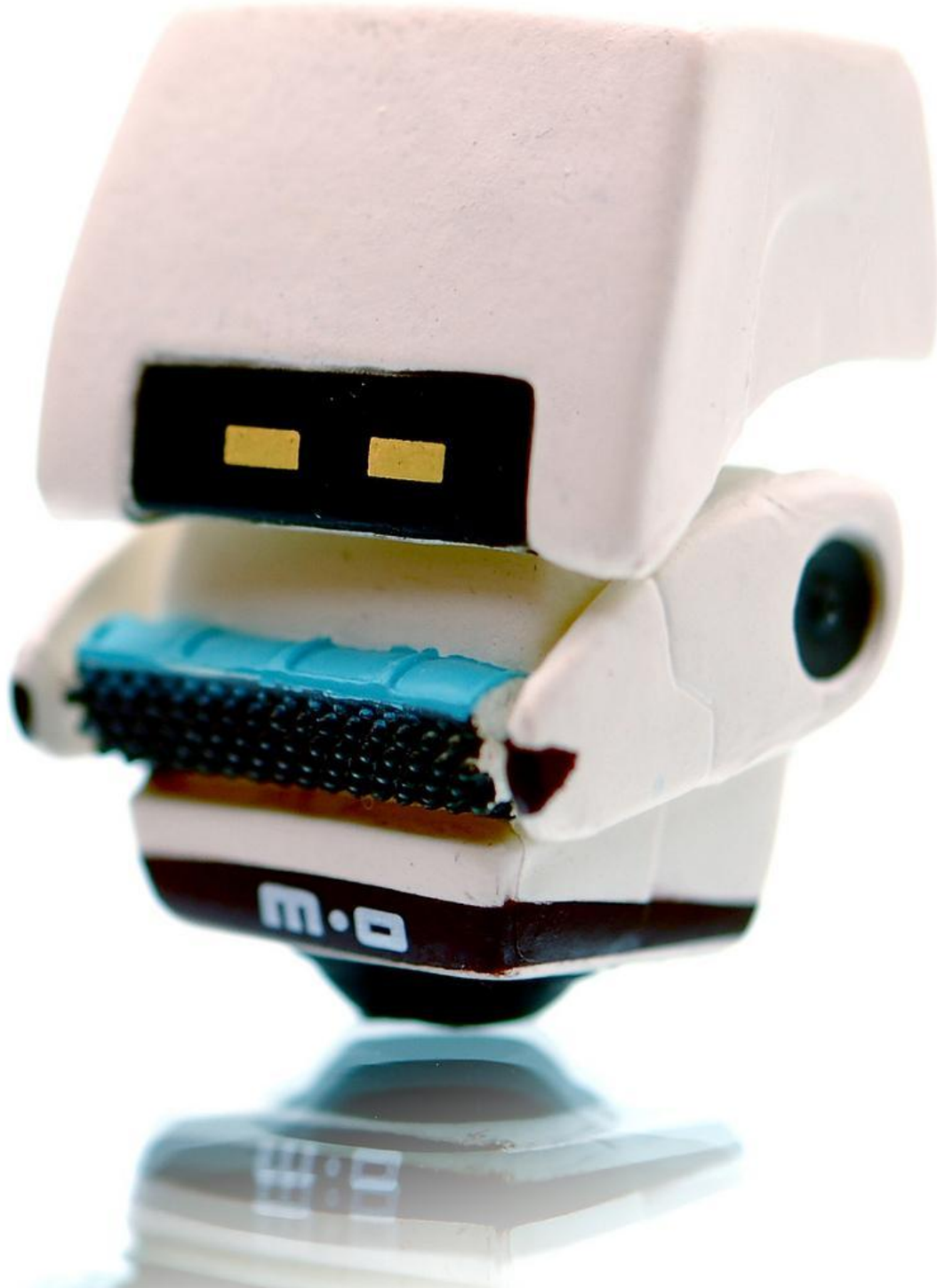
```ruby
it "should return 0 for an empty string" do
  expect(calculator.add("")).to eq(0)
end
```

# Write the simplest possible implementation that makes the test pass

https://www.flickr.com/photos/smanography/2954872088

```ruby
class StringCalculator
  def add(string_to_add)
    0
  end
end
```

# Refactor



https://www.flickr.com/photos/jdhancock/7037726211

# Refactoring

Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure.

https://www.flickr.
com/photos/billerr/1814657036

# Baby Steps

```ruby
it "should return 0 for an empty string" do
  expect(calculator.add("")).to eq(0)
end

it "should return 2 when only that number" do
  expect(calculator.add("2")).to eq(2)
end
```

```ruby
class StringCalculator
  def add(string_to_add)
    string_to_add.to_i
  end
end
```

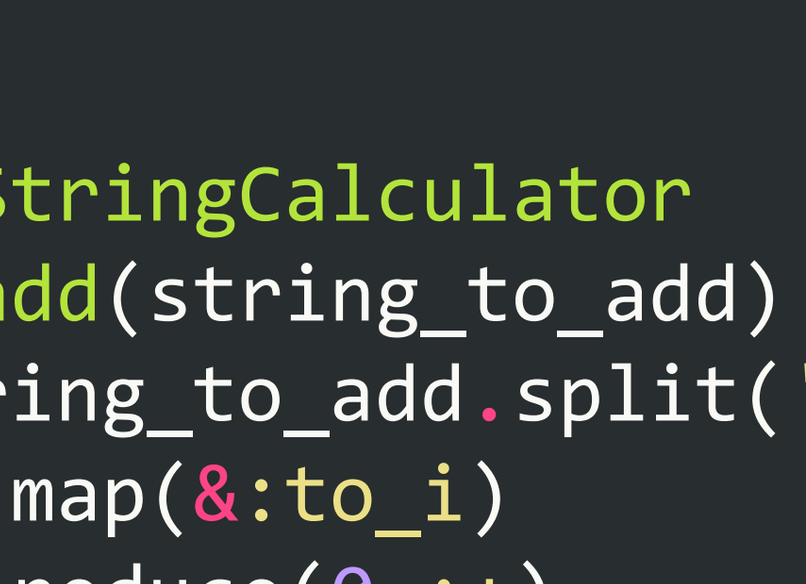
```ruby
it "should return 0 for an empty string" do
  expect(calculator.add("")).to eq(0)
end
it "should return 2 when only that number" do
  expect(calculator.add("2")).to eq(2)
end
it "should return 3 if adding 1 and 2" do
  expect(calculator.add("2,1")).to eq(3)
end
```

```ruby
class StringCalculator
  def add(string_to_add)
    total = 0
    string_to_add.split(",").each do |number_in_string|
      total += number_in_string.to_i
    end
    total
  end
end
```

# Fear of refactor

https://www.flickr.com/photos/foxspain/3614203847

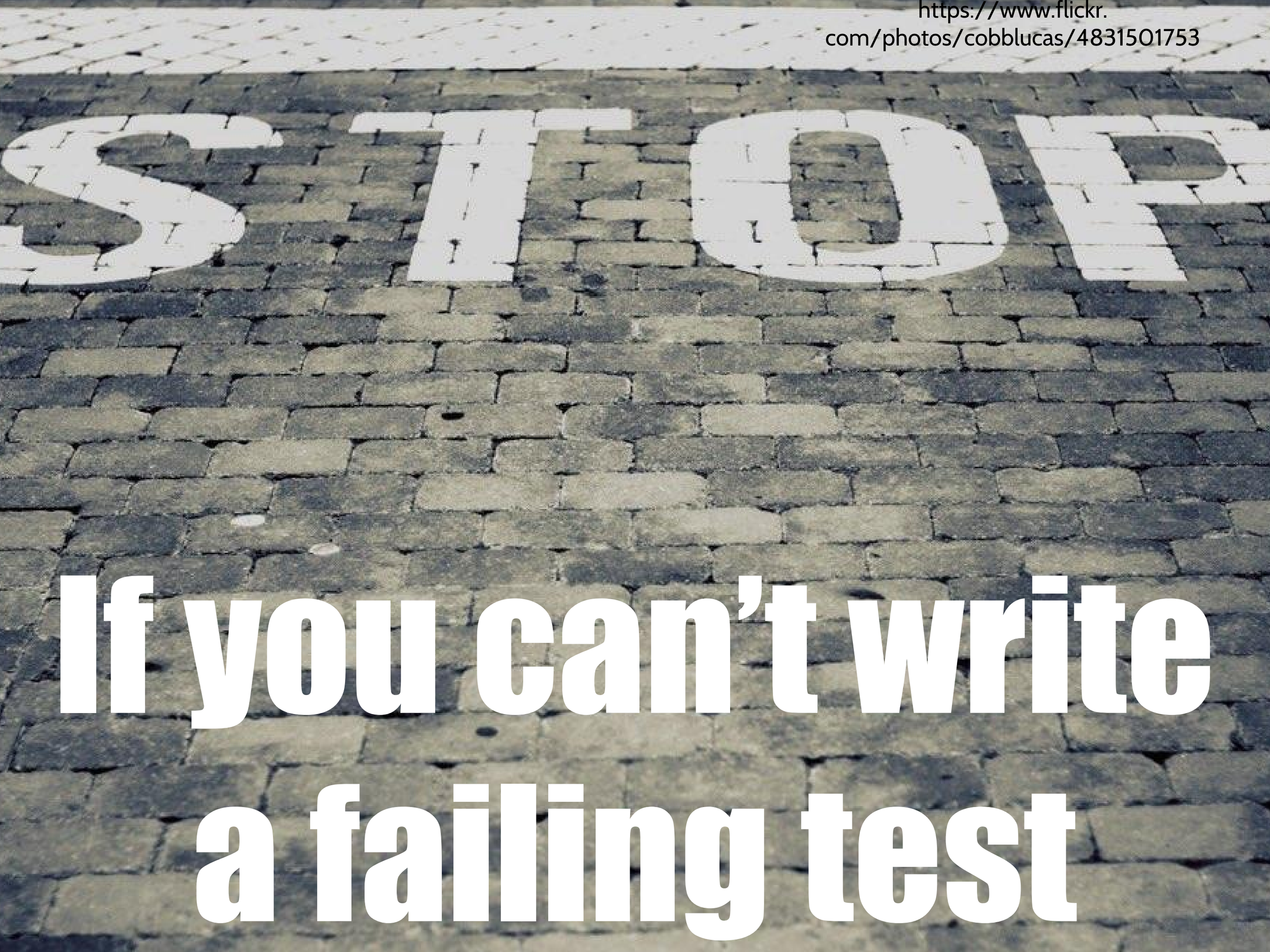
```ruby
class StringCalculator
  def add(string_to_add)
    string_to_add.split(',')
      .map(&:to_i)
      .reduce(0,:+)
  end
end
```



Clever

# RUBY HEROES

```ruby
it "should return 0 for an empty string" do
  expect(calculator.add("")).to eq(0)
end
it "should return 2 when only that number" do
  expect(calculator.add("2")).to eq(2)
end
it "should return 3 if adding 1 and 2" do
  expect(calculator.add("2,1")).to eq(3)
end
```




https://www.flickr.com/photos/cobblucas/4831501753

# Exercise

By now you should all know FizzBuzz rules. Can you TDD a FizzBuzz implementation?

The class should have a method that receives a number and returns the right message for that number.

# Some advice from our experts

# What to test?

- Business logic of your application. What it does.
- We don’t test UI
- We don’t test external systems
- We don’t test 3rd party libraries
- Separation of concerns is key to TDD
- Legacy code before you refactor it

# Avoid testing implementation details

https://www.flickr.com/photos/gsi-r/14663918878

```ruby
RSpec.describe "string calculator" do
  let(:calculator) { StringCalculator.new }

  it "should return 0 for an empty string" do
    expect(calculator.add("")).to eq(0)
  end
  it "should return 2 when only that number" do
    expect(calculator.add("2")).to eq(2)
  end
  it "should return 3 if adding 1 and 2" do
    expect(calculator.add("2,1")).to eq(3)
  end
end
```

```ruby
class StringCalculator
  def add(string_to_add)
    StringSplitter.new(string_to_add).split
      .map(&:to_i)
      .reduce(0,:+)
  end
end
```

Behaviour

```ruby
class StringSplitter
  def initialize(string_to_split)
    @string_to_split = string_to_split
  end

  def split
    @string_to_split.split(',')
  end
end
```

Implementation detail

```ruby
RSpec.describe "string calculator" do
  let(:calculator) { StringCalculator.new }

  it "should return 0 for an empty string" do
    expect(calculator.add("")).to eq(0)
  end
  it "should return 2 when only that number" do
    expect(calculator.add("2")).to eq(2)
  end
  it "should return 3 if adding 1 and 2" do
    expect(calculator.add("2,1")).to eq(3)
  end
end
```

https://www.flickr.com/photos/designandtechnologydepartment/5139208012

# Integration testing

```ruby
it "should return json from our external service" do
  expect(external_service.read_data.to_json).not_to be_nil
end
```

https://www.flickr.
com/photos/denniswong/3309276218

# Test Smells


# Fragile Test

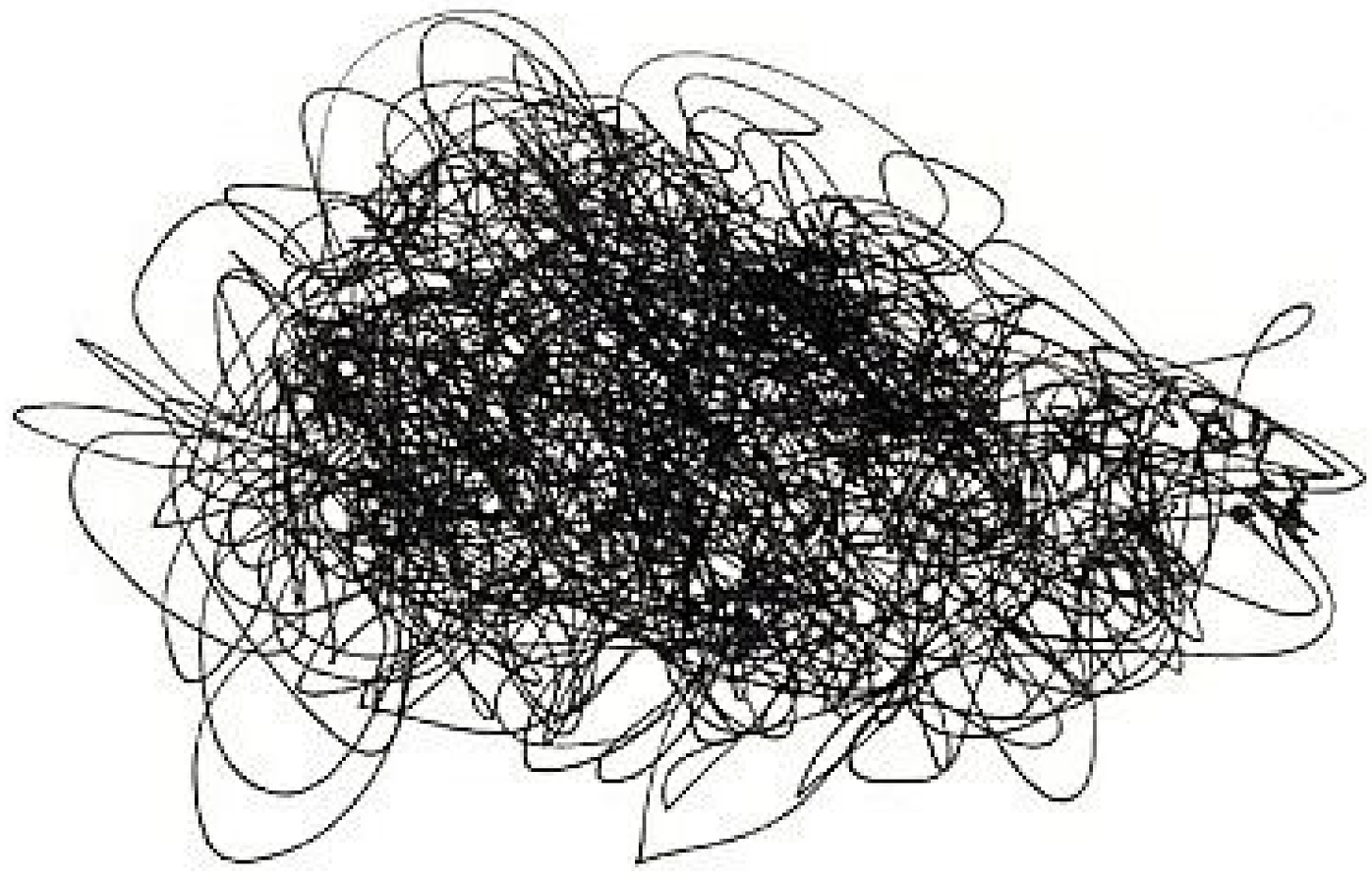
```ruby
class StringCalculator
  def add(string_to_add)
    StringSplitter.new(string_to_add).split
      .map(&:to_i)
      .reduce(0,:+)
  end
end
```

```ruby
class StringSplitter
  def initialize(string_to_split)
    @string_to_split = string_to_split
  end

  def split
    @string_to_split.split(',')
  end
end
```

```ruby
RSpec.describe "string splitter" do
  it "splits a simple string separated by comma" do
    expect(StringSplitter.new("a,b").split).to eq(["a", "b"])
  end
end
```

```ruby
class StringCalculator
  def add(string_to_add)
    StringSplitter.new.split(string_to_add)
      .map(&:to_i)
      .reduce(0,:+)
  end
end

class StringSplitter
  def split(string_to_split)
    string_to_split.split(',')
  end
end
```

# Erratic Test



```ruby
RSpec.describe "people" do
  it "saves the person correctly" do
    @person = Person.new.save
    expect(@person).to be_true
  end

  it "finds the last person saved correctly" do
    expect(Person.last_person_saved).to eq(@person)
  end
end
```

# Slow tests

https://www.flickr.com/photos/barbourians/6662357209