Testing Ruby Applications with RSpec

Getting Started

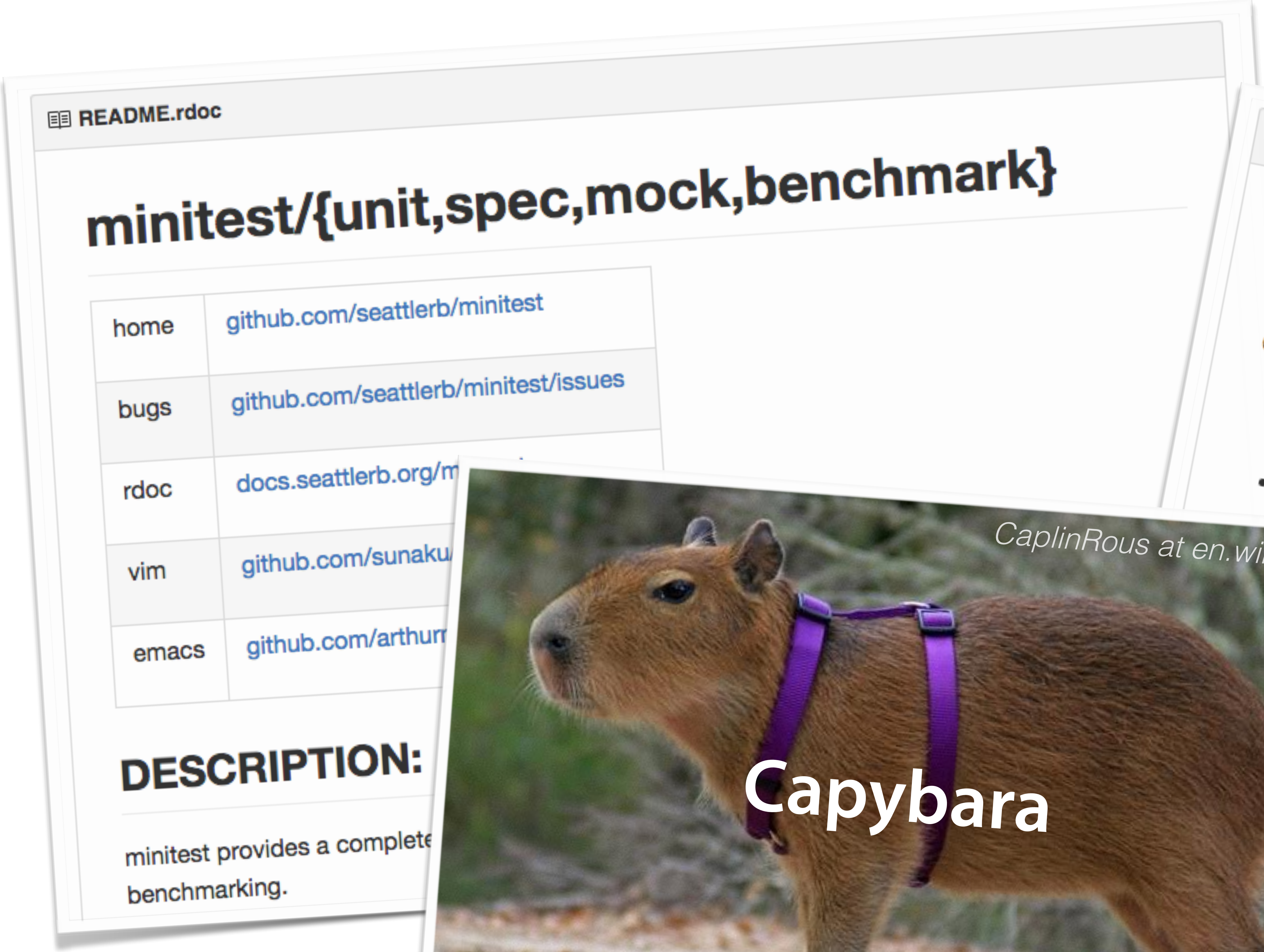
Xavier Shay

http://xaviershay.com | @xshay

RSpec Family

Library	Features
rspec-core	Runner and syntax
rspec-expectations	Express desired outcomes
rspec-mocks	Test doubles

RSpec Ecosystem


README.rdoc

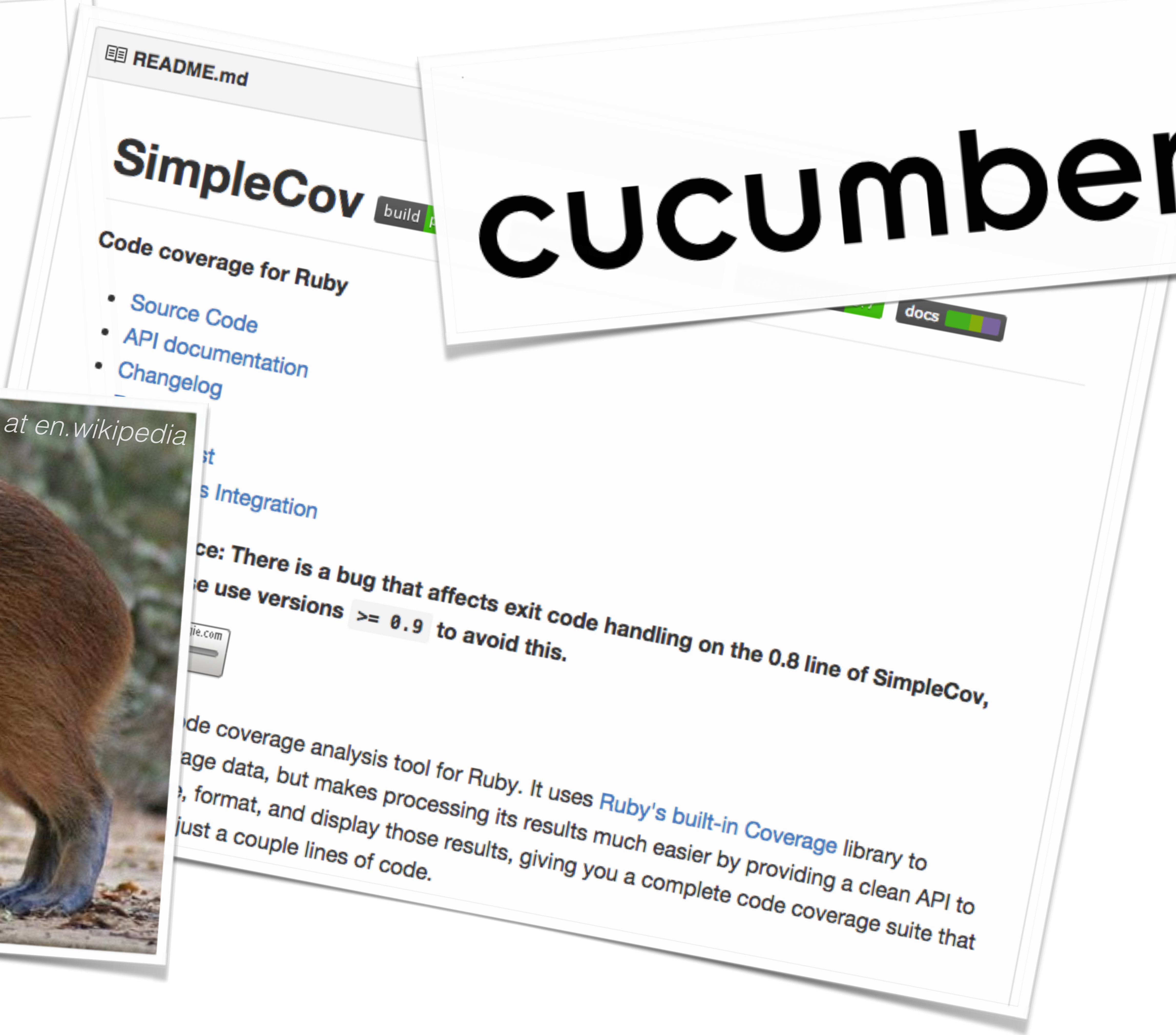
minitest/{unit,spec,mock,benchmark}

home	github.com/seattlerb/minitest
bugs	github.com/seattlerb/minitest/issues
rdoc	docs.seattlerb.org/m
vim	github.com/sunaku
emacs	github.com/arthur

DESCRIPTION:

minitest provides a complete

benchmarking.


README.md

SimpleCov build

Code coverage for Ruby

- Source Code
- API documentation
- Changelog

st

s Integration

ce: There is a bug that affects exit code handling on the 0.8 line of SimpleCov,
e use versions `>= 0.9` to avoid this.

de coverage analysis tool for Ruby. It uses Ruby's built-in Coverage library to
age data, but makes processing its results much easier by providing a clean API to
, format, and display those results, giving you a complete code coverage suite that
just a couple lines of code.

docs

cucumber

Testing Concepts

Why do we test?

When are different features appropriate?

Let’s go!

Setting Up

Gems, bundler, and documentation

RSpec Methods

Method	Meaning
it	Specify a property. “Example.”
describe	Group examples / properties.

Your First Spec

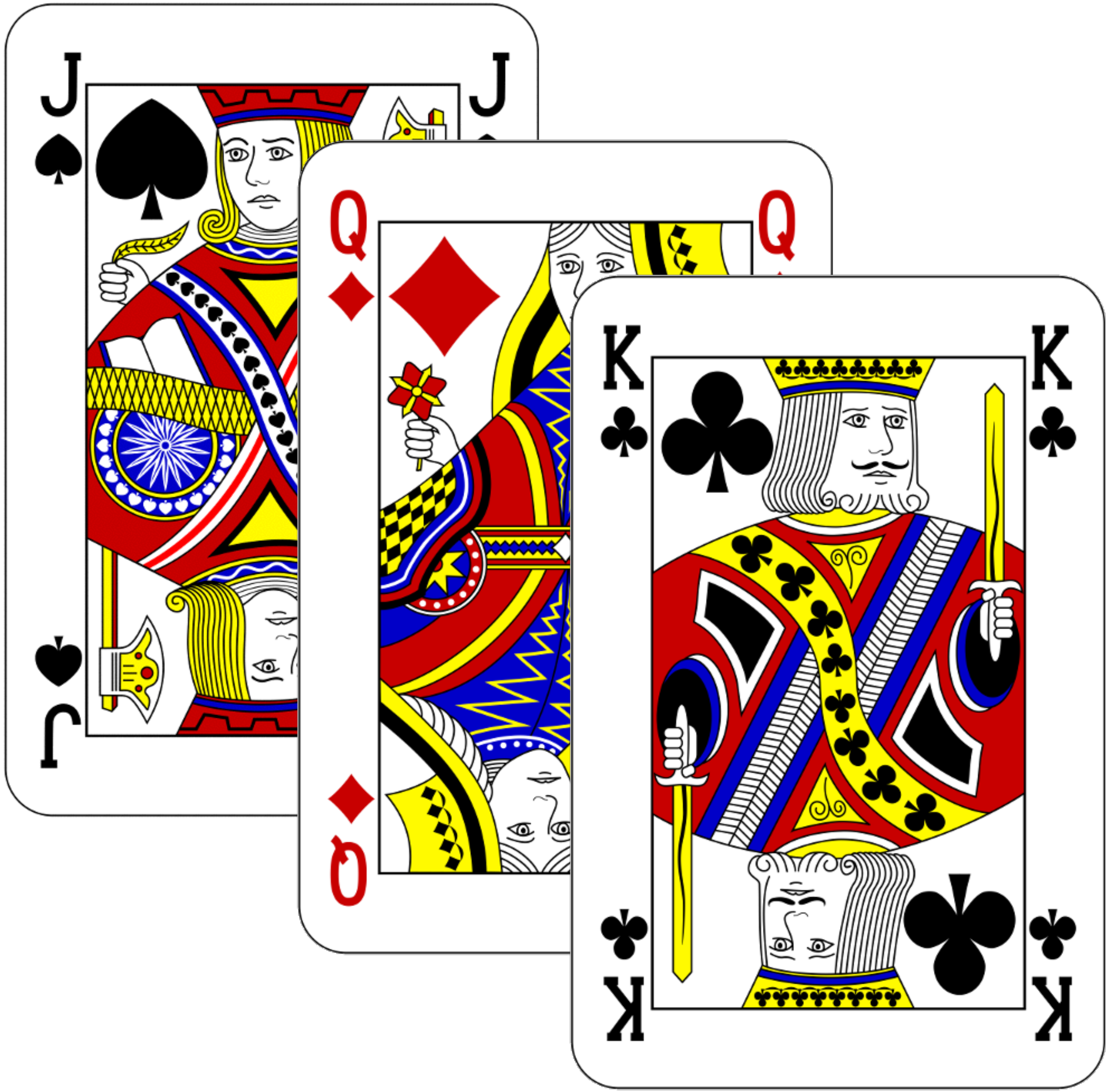
describe, it, and the rspec command

Pokeravond, Thomas van de Weerd

RSpec Motivation

Automatically verify correctness

Document desired behavior


J
J
Q
Q
K
K

Behavior

not

Implementation

Why spec?

Why?

CONFIDENCE

Guidelines

Test for confidence, not proof

One branch/method per spec

Recap

Basic Building Blocks

Describe, it, how to use them.

Principles

Behavior not implementation, confidence not proof.

RSpec CLI

Formatters and color.

Get Organized

File layout and common configuration

Local > Project > Global

Recap

Three Conventions

lib and spec, _spec.rb, spec_helper.rb

Configuration

Local, project, global.

Compact Specifications

Helper methods, shared examples, and let.

```
Card.new(suit: :spades, rank: 4)
Card.new(suit: :spades, rank: 4)
```

1. Constructor supports a rank.

```
Card.new(suit: :spades, rank: 4)
```

1. Constructor supports a rank.
2. Constructor is called new.

```
Card.new(suit: :spades, rank: 4)
```

1. Constructor supports a rank.
2. Constructor is called new.
3. Uses named parameters.

```
Card.new(suit: :spades, rank: 4)
```

1. Constructor supports a rank.
2. Constructor is called new.
3. Uses named parameters.
4. Class is named Card.

Spec should only change when *behavior* does subject

equivalent to

let(:subject)

Recap

Helper methods

def card, def subject

Shared examples

shared_examples_for, it_behaves_like

RSpec convenience

let, subject

Concepts

Reduce dependency

Card.new(suit: :spades, rank:4) → card(rank: 4)

Spec behavior, not implementation

Testing with Set

Deep Dive

describe and it

Terminology

Keyword	Concept
describe	Example Group
it	Example

Example Group Mental Model

Classes and methods!

describe and it are just fancy syntax

Acceptance Tests

Does our app work?

example

alias for

it

Types of tests

Card	CLI
Specific	General
Locates Problem	Problem Unclear
Fast	Slow

Unit Test

Do our objects do the right thing, and are they convenient to work with?

Integration Test

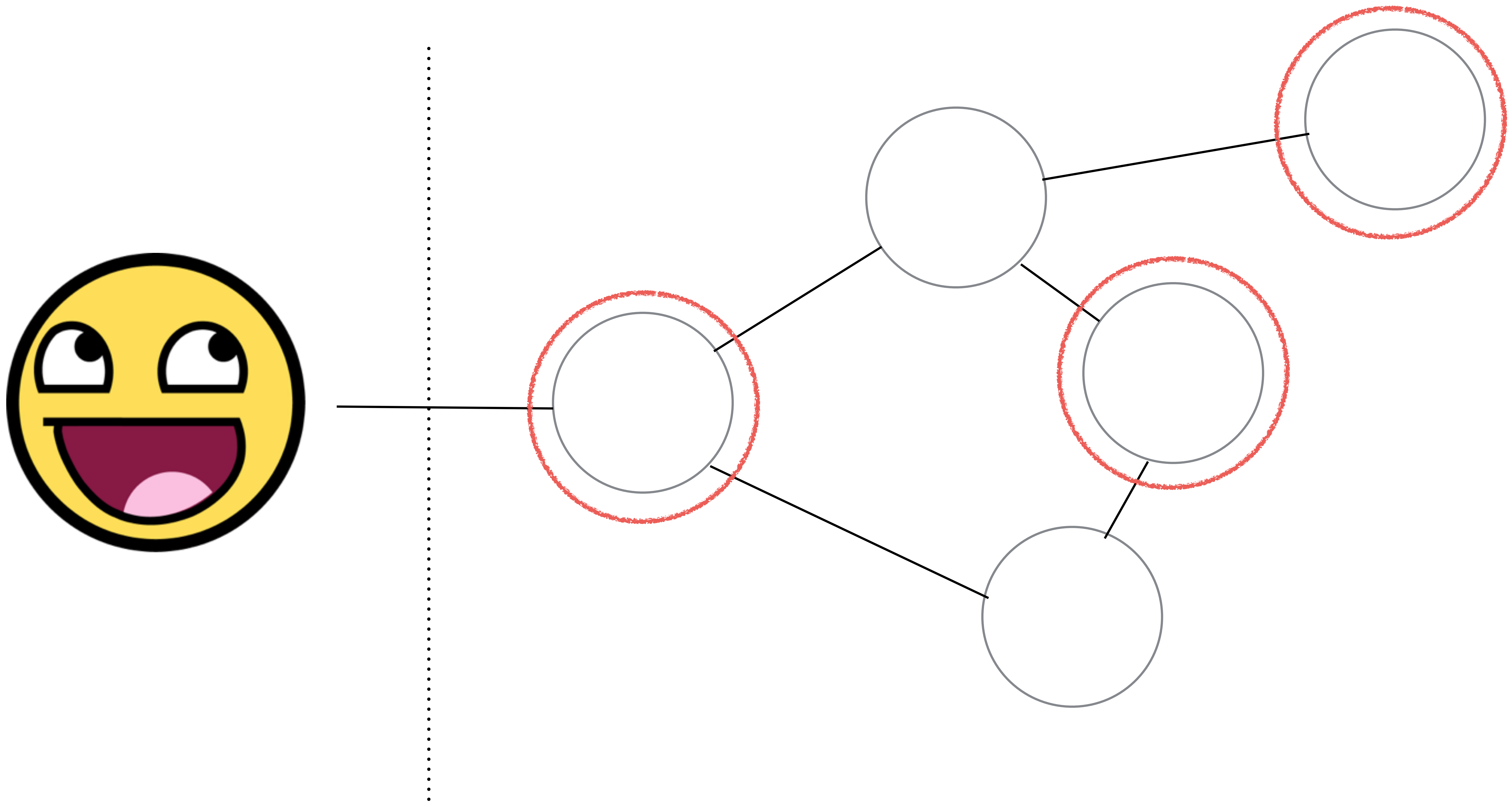
Does our code work against objects we can't change?

Acceptance Test

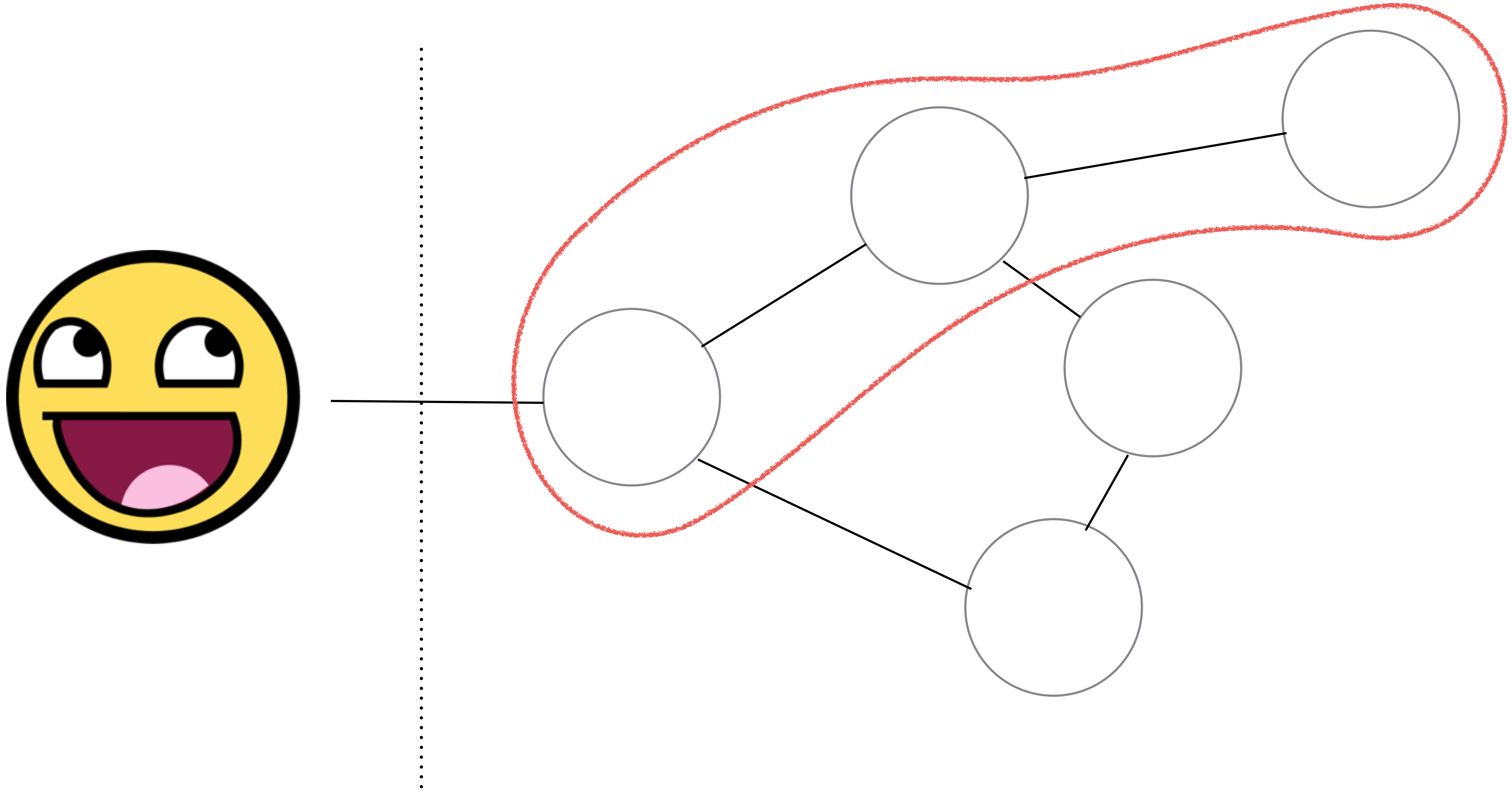
Does the whole system work?

Growing OO Software, Guided by Tests, p.

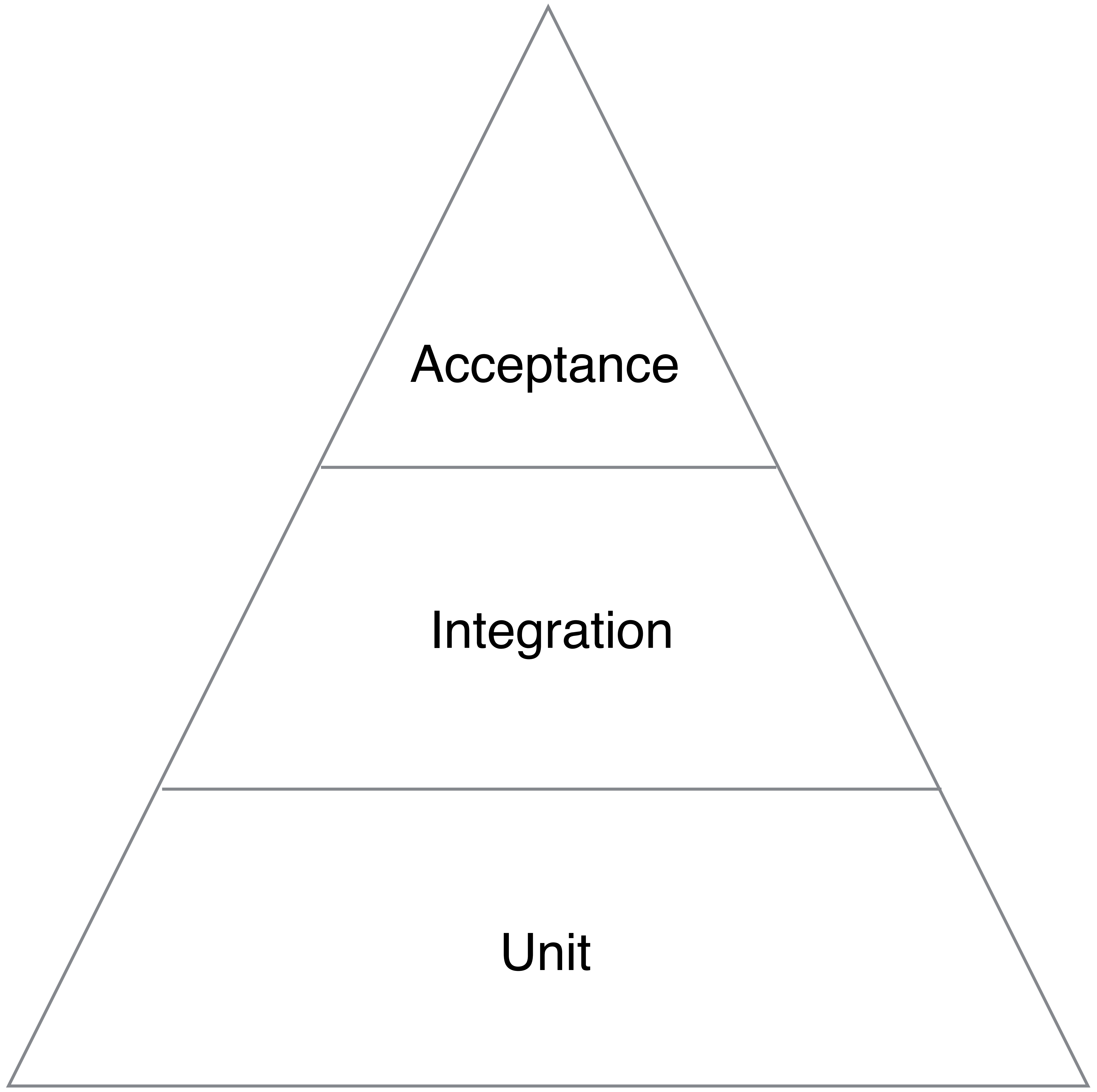
Types of tests



Types of tests



Types of tests



Hooks and Metadata

More ways to organize your specs

Hook scopes

Keyword	Old Keyword	Scope
example	each	Once per example (it)
context	all	Once per example group (describe)
suite	suite	Once per spec run

Hook types

Keyword
before
after
around

Why not hooks?

Not localized to the spec.

Trends worse over time.

Recap

Where are we now?

Concepts

Confidence, not proof.

Unit and Acceptance Tests.

Behavior

not

Implementation