Understanding the RSpec Ecosystem

Xavier Shay

http://xaviershay.com | @xshay

```
(1+1).should == 2
```

equivalent to

```
expect(1+1).to eq(2)
```

Why not should?

Requires monkey-patching every object

Can generate warnings

Needs workarounds for operator precedence

Semantic Versioning

$2.14 \rightarrow 3.0$

Backwards Incompatible

$3.3 \rightarrow 3.4$

Backwards Compatible

Upgrading RSpec

.99 versions and transpec

Code Coverage

Using simplecov

Coverage Limitations

Tracks lines, not branches.

100% doesn’t mean all code is executed.

… < 100% means it certainly isn’t.

Custom Formatters

The formatter API and Fivemat

Conclusion

We’re done!

The RSpec Family

Core	Expectations	Mocks

RSpec Core

rspec command-line tool

describe and it

Encourages Behaviour-Driven Development

RSpec Expectations

Fancy syntax for raise

expect(1+1).to eq(2)

Nicer error messages

Custom matchers

RSpec Mocks

Provide alternate method implementations

Construct fake objects

Provides design feedback

Concepts

How to use RSpec effectively

Behavior

not

Implementation

Testing implementation tells

Many expectations in a single example

Too many mocks

Mocks

are are for

Design

Using Mocks Effectively

Primarily for design feedback

Can be used as scaffolding

Testing Ruby Applications with RSpec

Xavier Shay

http://xaviershay.com | @xshay