Rubyの テストカバレッジ 測定機能の 改良と展望

クックパッド株式会社
遠藤侑介
yusuke-endoh@cookpad.com

RubyWorld Conference 2017
(2017/11/01)

発表概要

- 発表内容
  - カバレッジとは
  - カバレッジとの付き合い方
  - Rubyでのカバレッジ測定方法
  - クックパッドでのカバレッジ利用事例
- 発表者について
  - フルタイム Ruby コミッタ (2017/09～) @ Cookpad
  - Rubyプログラムの堅牢性を上げたい➔カバレッジから

発表概要

- **➔カバレッジとは←**
- カバレッジとの付き合い方
- Rubyでのカバレッジ測定方法
- クックパッドでのカバレッジ利用事例
- まとめ

カバレッジとは

- 「テストの良さ」を測る指標
  - コードカバレッジ、テストカバレッジとも
- カバレッジを測定する効果
  - テストされていないコードがわかる
  - テストの度合いを見積もれる
- Ruby ではカバレッジが極めて重要
  - 現時点で、テスト以外の検証手段が事実上ない
  - Ruby 1.9 からカバレッジ測定機能を標準実装している
    - 発表者が開発・メンテナンス中

カバレッジの種類

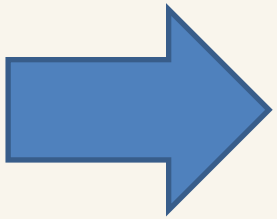
- 関数カバレッジ
- 行カバレッジ
- 分岐カバレッジ

  – 他にも条件カバレッジ、パスカバレッジなど

関数カバレッジ

- テストで実行された関数の割合

```
# code
def foo; …; end # ✓
def bar; …; end # X
def baz; …; end # ✓

# test
foo
baz
```

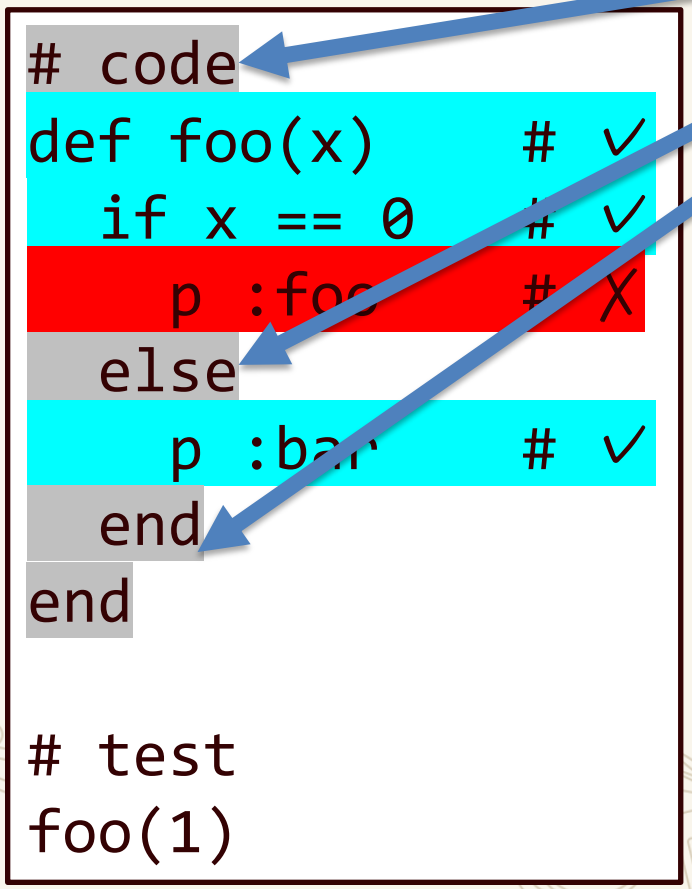


2/3
(67%)

- 😀わかりやすい
- 😀可視化しやすい
- 😞指標としては弱い

行カバレッジ

- テストで実行された行の割合



意味のない行は無視される

3/4
(75%)

- 😐わかりやすい
- 😐可視化しやすい
- 😞指標としてはまだ弱い

```
foo() if x == 0
```

分岐カバレッジ

- 真と偽の両方にジャンプした分岐の割合

```
# code
def foo(x)
  p :foo if x == 0 # ✓
  p :bar if x < 2  # X
end

# test
foo(0)
foo(1)
```

真の場合と偽の場合の両方が実行された

1/2 (50%)

- 😀わりと網羅的
- 😟可視化が難しい

カバレッジの種類とRubyの対応状況

	関数カバレッジ	行カバレッジ	分岐カバレッジ
わかりやすさ・ 可視化のしやすさ	○	○	△
網羅性	×	△	○
Ruby 2.4以下で 測定できるか	△	○	×

ユーザが
実装可能

組み込み
サポート

測定不可

カバレッジの種類とRubyの対応状況

	関数カバレッジ	行カバレッジ	分岐カバレッジ
わかりやすさ・ 可視化のしやすさ	○	○	△
網羅性	×	△	○
Ruby 2.4以下で 測定できるか	△	○	×
Ruby 2.5 で 測定できるか	○	○	○

Ruby 2.5 は関数・分岐カバレッジを標準サポート予定
（開発版で実装済み）

発表概要

- カバレッジとは
- **➔カバレッジとの付き合い方←**
- Rubyでのカバレッジ測定方法
- クックパッドでのカバレッジ利用事例
- まとめ

「良いテスト」とは

- コードに対して網羅的
  - カバレッジで測れる
- 仕様に対して網羅的
  - カバレッジで測れない

具体例

- 計算をする関数とテストを書いた

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end

# test
calculate("+", 3, 4) #=> 7
```

具体例

- 計算をする関数とテストを書いた
- 分岐カバレッジを測定した
  ➔テスト不足判明

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end

# test
calculate("+", 3, 4) #=> 7
```

具体例

- 計算をする関数とテストを書いた
- 分岐カバレッジを測定した
  ➔テスト不足判明
- テストを追加した
  ➔カバレッジ100%

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end

# test
calculate("+", 3, 4) #=> 7
calculate("-", 4, 3) #=> 1
```

具体例

- 計算をする関数とテストを書いた
- 分岐カバレッジを測定した
  ➔テスト不足判明
- テストを追加した
  ➔カバレッジ100%
- 仕様で掛け算も要求されてた…

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end

# test
calculate("+", 3, 4) #=> 7
calculate("-", 4, 3) #=> 1
```

具体例

- 計算をする関数とテストを書いた
- 分岐カバレッジを測定した
  ➔テスト不足判明
- テストを追加した
  ➔カバレッジ100%
- 仕様で掛け算も要求されてた…

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  when "*" then a * b
  end
end
# test
calculate("+", 3, 4) #=> 7
calculate("-", 4, 3) #=> 1
calculate("*", 2, 3) #=> 6
```

どうすればよかったか

- 計算をする関数とテストを書いた
- 分岐カバレッジを測定した

  ➔テスト不足判明

この時点で焦ってテストを書き足すのではなく、「どういう観点が不足していたか」を考える

```
# code
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end

# test
calculate("+", 3, 4) #=> 7
```

カバレッジとの付き合い方

- カバレッジは「考えるきっかけ」として使う
  - テスト不足のモジュール・コードを見て、テスト・コードに不足していた観点を考える
- カバレッジは「目標」や「管理ツール」ではない
  - 「カバレッジ90%以上が必達目標！」などというと、開発者は安易なテスト追加をしてしまう
  - 安易なテストも無いよりマシだが「考えるきっかけ」を失う

発表概要

- カバレッジとは
- カバレッジとの付き合い方
- **➔Rubyでのカバレッジ測定方法←**
- クックパッドでのカバレッジ利用事例
- まとめ

Ruby でのカバレッジ測定方法

- Ruby組み込みカバレッジ測定機能を使う方法
  - coverage.so: 発表者が実装した標準ライブラリ
  - Ruby 2.5 では関数・分岐カバレッジも測定できる
- SimpleCov を使う方法（本日紹介）
  - 広く普及している coverage.soのラッパ
  - Ruby 2.4 でも動作する
  - 現時点では行カバレッジのみ
    （将来的に関数・分岐カバレッジもサポート予定）

例

- コードとテストを書く

code.rb

```
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a – b
  end
end
```

code_spec.rb

```
require "./code"
describe "計算" do
 it "3 + 4 は 7 になること" do
  expect(calculate("+", 3, 4)).to eq 7
 end
end
```

例

- コードとテストを書く
- spec_helper.rb を右のように書く
  - 必ずファイル先頭に書き足す
  - .rspec ファイルも用意する

code.rb

```
def calculate(type, a, b)
  case type
  when "+" then a + b
  when "-" then a - b
  end
end
```

code_spec.rb

```
require "./code"
describe "計算" do
 it "3 + 4 は 7 になること" do
  expect(calculate("+", 3, 4)).to eq 7
 end
end
```

spec_helper.rb

```
require "simplecov"
SimpleCov.start
```

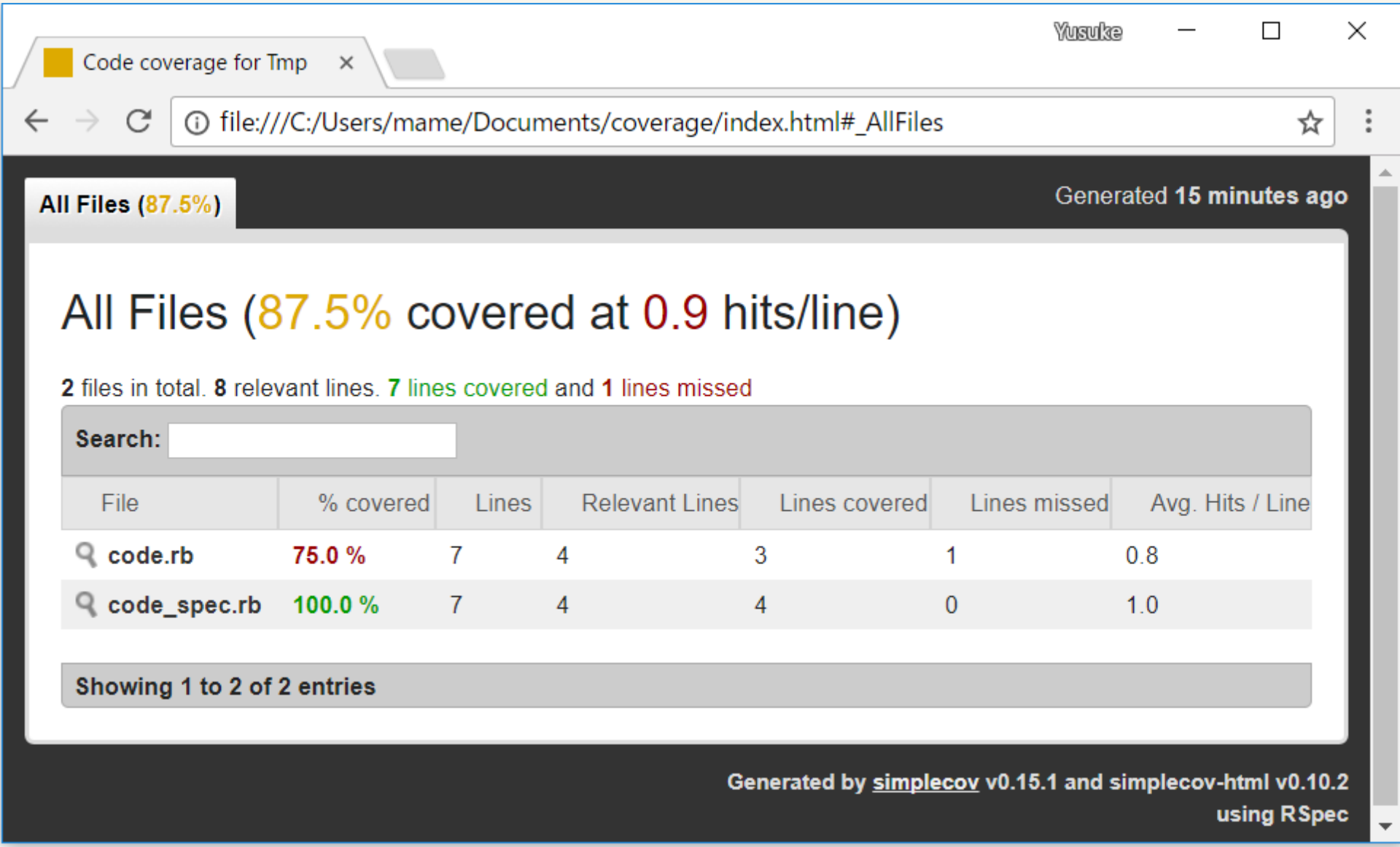
.rspec

```
--require ./spec_helper
```

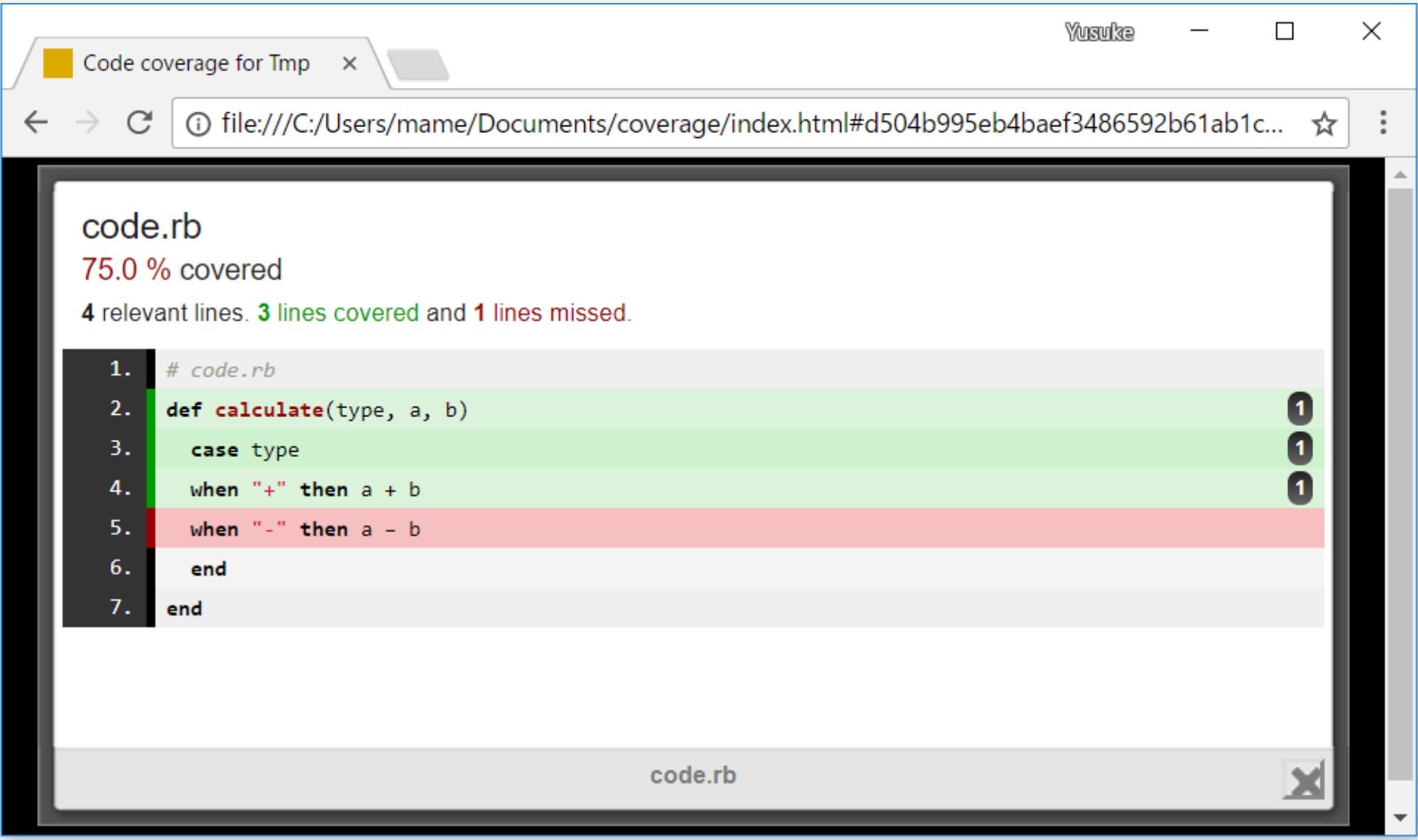
例：実行

```
$ rspec code_spec.rb
.

Finished in 0.00629 seconds (files took 0.20378
seconds to load)
1 example, 0 failures

Coverage report generated for RSpec to
/home/mame/tmp/coverage. 7 / 8 LOC (87.5%)
covered.
```

例：カバレッジの確認（俯瞰）



例：カバレッジの確認（ファイル）



発表概要

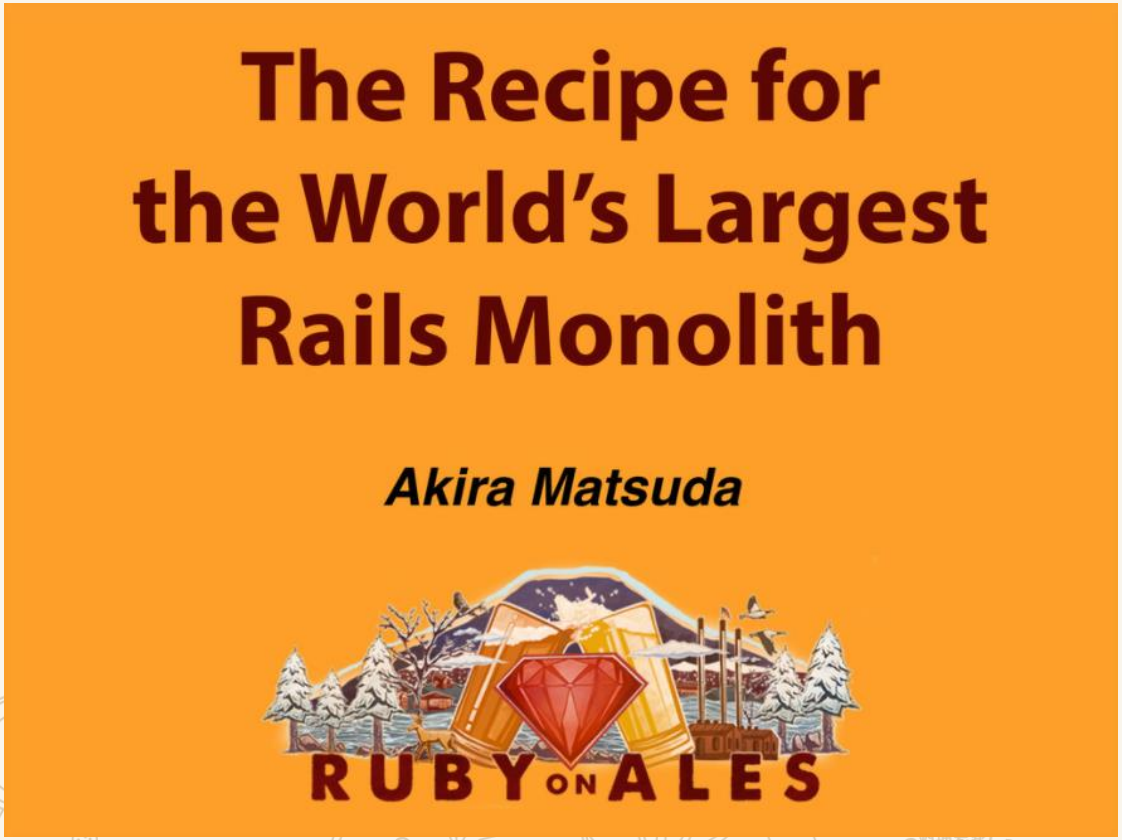
- カバレッジとは
- カバレッジとの付き合い方
- Rubyでのカバレッジ測定方法
- **➔クックパッドでのカバレッジ利用事例➔**
- まとめ

クックパッドのサービス規模

- レシピ数：270万品以上
- 月次利用者数：約6000万人
- 対応言語：21言語67ヶ国
- 海外の月次利用者数：3000万人以上
- プレミアム会員数：190万人以上

クックパッドのプログラム規模

- 世界でも有数の巨大Railsアプリ



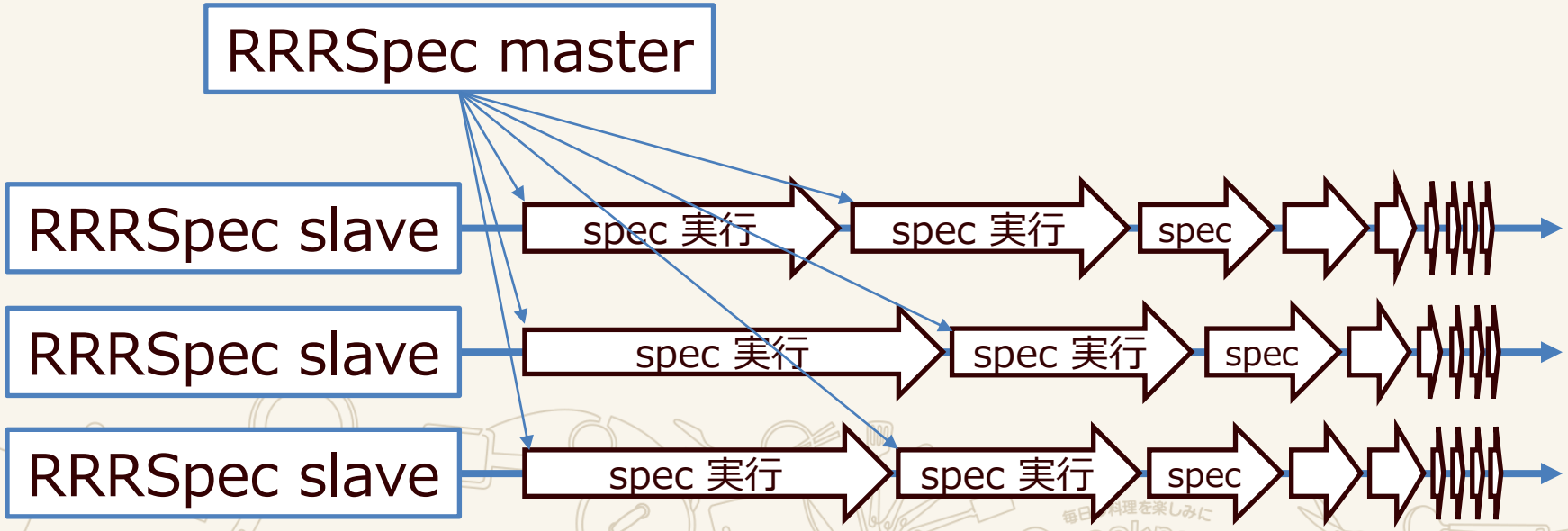
https://speakerdeck.com/a_matsuda/the-recipe-for-the-worlds-largest-rails-monolith

% rake stats

Name	Lines	LOC	Classes	Methods	M/C	LOC/M
Controllers	48552	39075	518	3941	7	7
Helpers	14660	12012	14	1390	99	6
Models	95193	74916	1732	8489	4	6
Mailers	2197	1757	44	204	4	6
Workers	593	501	20	31	1	14
Chanko units	11816	9732	6	247	41	37
Libraries	2781	2213	134	290	2	5
Feature specs	43536	35864	0	196	0	180
Request specs	36432	31235	0	16	0	1950
Routing specs	639	516	0	0	0	0
Controller specs	60543	50042	7	123	17	404
Helper specs	4195	3436	1	10	10	341
Model specs	75517	62368	4	72	18	864
Worker specs	862	715	0	1	0	713
Chanko unit specs	11636	9411	0	24	0	390
Library specs	22983	19202	27	131	4	144
Total	432135	352995	2507	15165	6	21

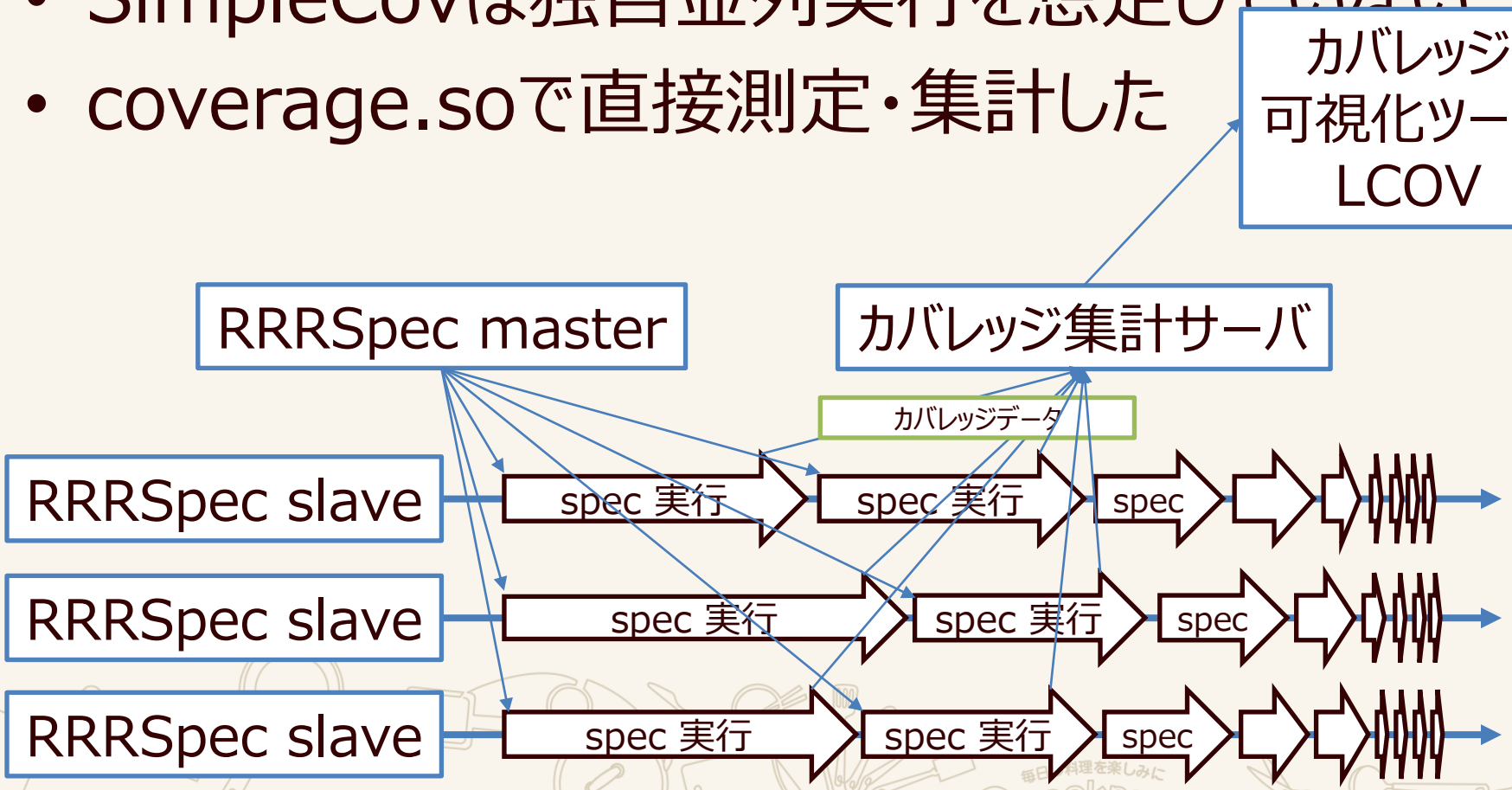
クックパッドのサービスのテスト

- 1 台の PC ではテスト実行に 2 日半
- 並列テスト実行システム(RRRSpec)で10分弱



カバレッジ測定

- SimpleCovは独自並列実行を想定していない
- coverage.soで直接測定・集計した



カバレッジ集計結果

	Lines	LOC	クラス数	メソッド数	関数 Cov.	行 Cov.	分岐 Cov.
Controllers	47390	38179	516	3901	88%	90%	73%
Helpers	15624	12763	18	1455	65%	73%	55%
Models	91785	77238	1796	8537	63%	79%	52%
Mailers	1847	1488	42	180	51%	65%	35%
Workers	136	119	4	15	77%	87%	50%
Chanko units	8387	6926	2	194	39%	57%	31%
Libraries	43037	35667	598	3187	58%	69%	45%

- 😀 Controllers は非常によくテストされている
- 😠 それ以外は改善の余地がありそう
  - （既に不要なコードもあるので一概には言えないが）

カバレッジの分析例（1）

- RecipeController#show

- 不要なコードを発見できた
  - 事後検証：先頭のチェックがあとから追加された

カバレッジの分析例（2）

- レシピのモデル

このあたりのメソッド群だけカバーされていない

現在はどこからも使われていないと判明

- 不要なコードを発見できた

カバレッジ分析の考察

- 事前知識なしでコードの改善点を発見できた
  - 半日程度の分析で数カ所を発見した
  - 発表者はクックパッドのソースコードだけでなく、Rails の知識もほぼ持たない

発表概要

- カバレッジとは
- カバレッジとの付き合い方
- Rubyでのカバレッジ測定方法
- クックパッドでのカバレッジ利用事例
- **➔まとめ←**

まとめ

- カバレッジとは何か：「テストの良さ」を測る指標
  - Ruby 2.4で測定できる➔行カバレッジ
  - Ruby 2.5で測定できる➔行・関数・分岐カバレッジ
- カバレッジとの付き合い方：考えるきっかけ
  - 「目標」や「管理ツール」として使わない
- Ruby でのカバレッジ測定方法
- クックパッドの Rails アプリでカバレッジ利用事例
  - 簡単な分析事例を紹介した