[IF977] Engenharia de Software

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

AssertLab
Advanced Software and Systems
Engineering Research Technologies

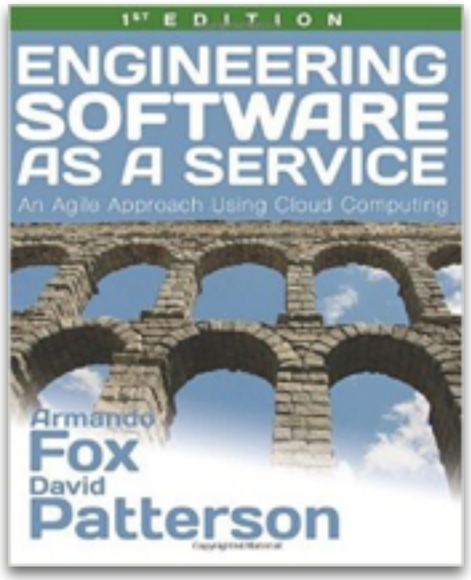
Centro de Informática

Universidade
Federal
de Pernambuco

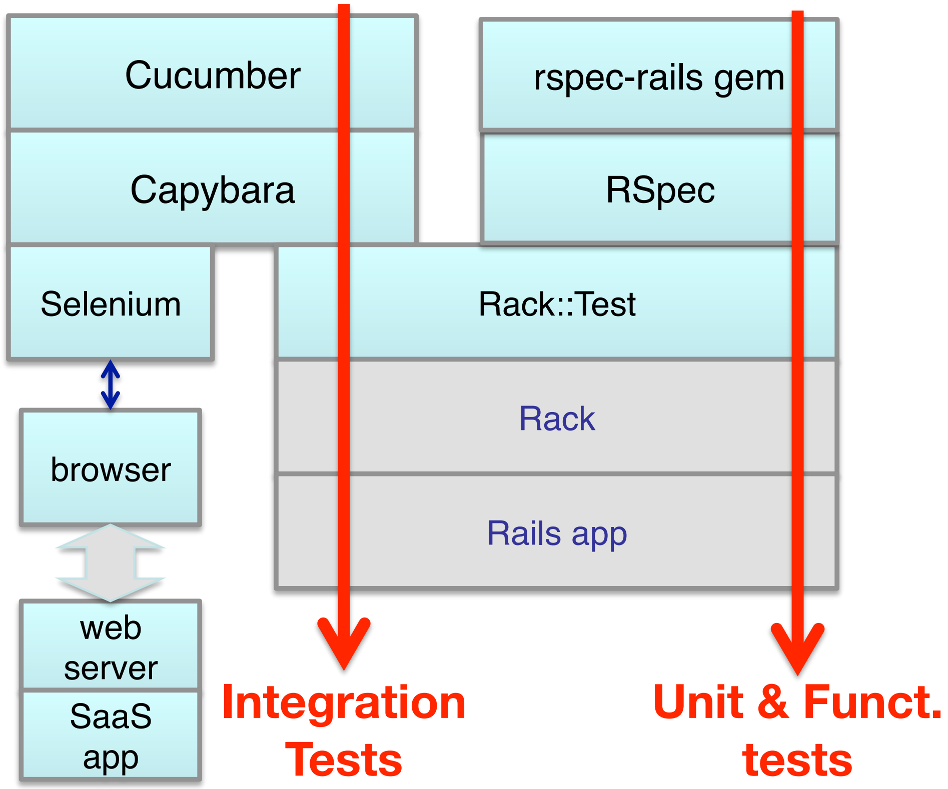
Licença do material

Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
    - http://bit.ly/TDOA5L
- SWEBOK
    - Guide to the Software Engineering Body of Knowledge (SWEBOK): http://www.computer.org/web/swebok
- Engineering Software as a Service: An Agile Approach Using Cloud Computing (Beta Edition)
    - http://www.saasbook.info/


1ST EDITION
ENGINEERING SOFTWARE AS A SERVICE
An Agile Approach Using Cloud Computing
Armando Fox
David Patterson

Testing stacks revisited

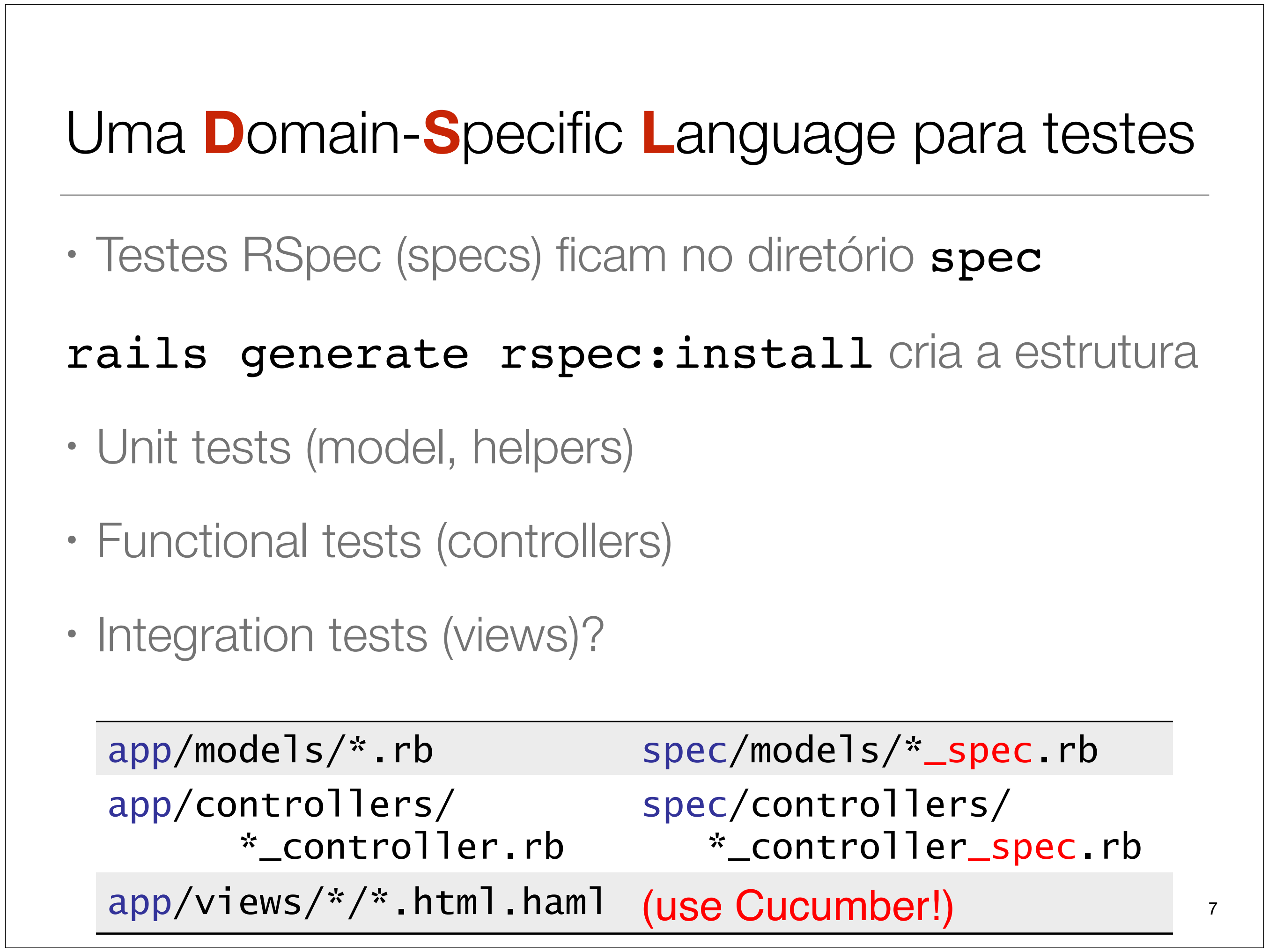


Refresher: Tests should be FIRST

- Fast
- Independent
- Repeatable
- Self-checking
- Timely

Behaviour Driven Development for Ruby.
Making TDD Productive and Fun.

RSpec on Rails



Uma Domain-Specific Language para testes

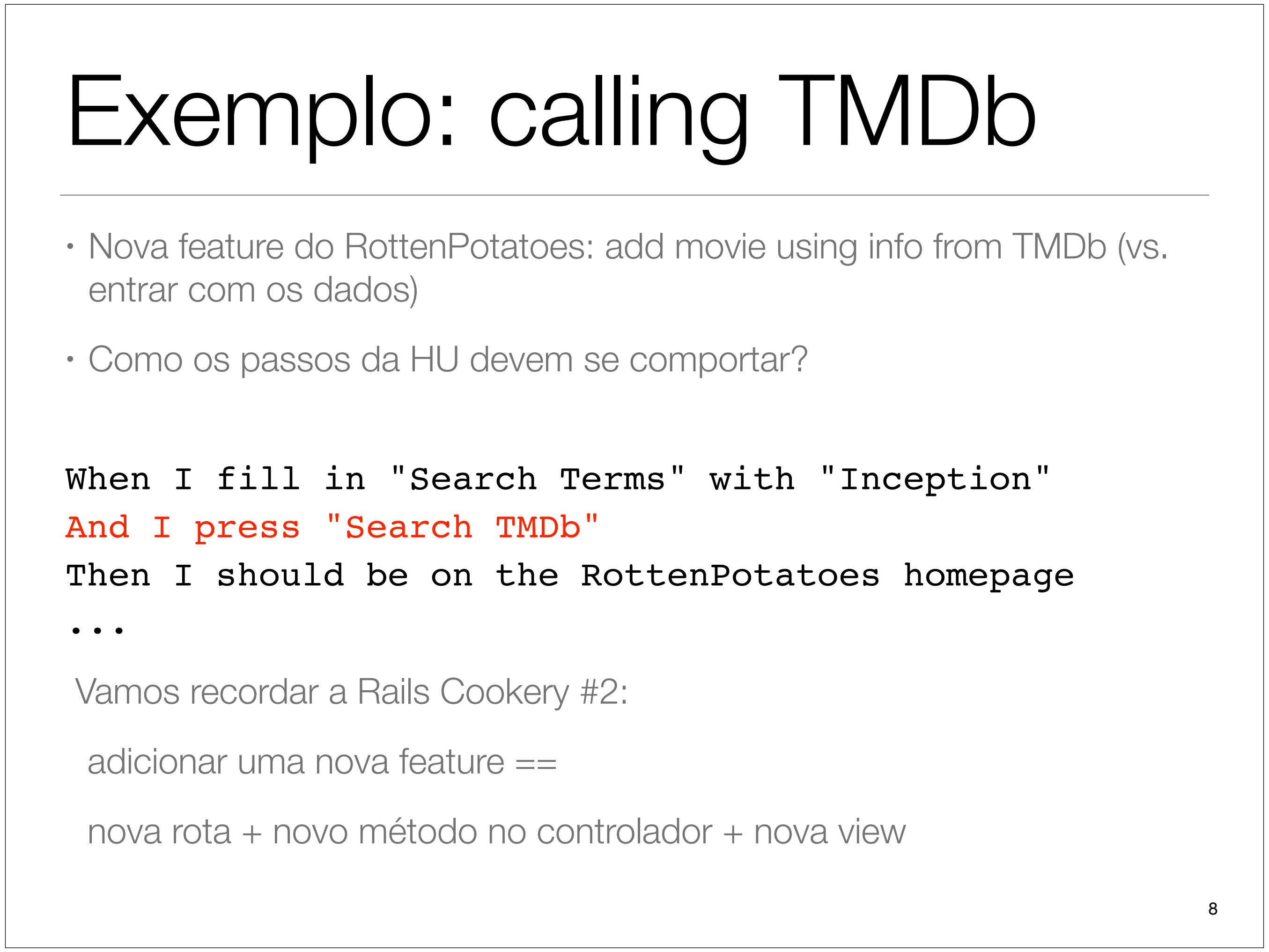
- Testes RSpec (specs) ficam no diretório `spec`

`rails generate rspec:install` cria a estrutura

- Unit tests (model, helpers)
- Functional tests (controllers)
- Integration tests (views)?

`app/models/*.rb`	`spec/models/*_spec.rb`
`app/controllers/` `*_controller.rb`	`spec/controllers/` `*_controller_spec.rb`
`app/views/*/*.html.haml`	(use Cucumber!)

Opinions vary on whether to test the views
Our position: views are user-facing, so use user stories to test => Cucumber



Exemplo: calling TMDb

- Nova feature do RottenPotatoes: add movie using info from TMDb (vs. entrar com os dados)
- Como os passos da HU devem se comportar?

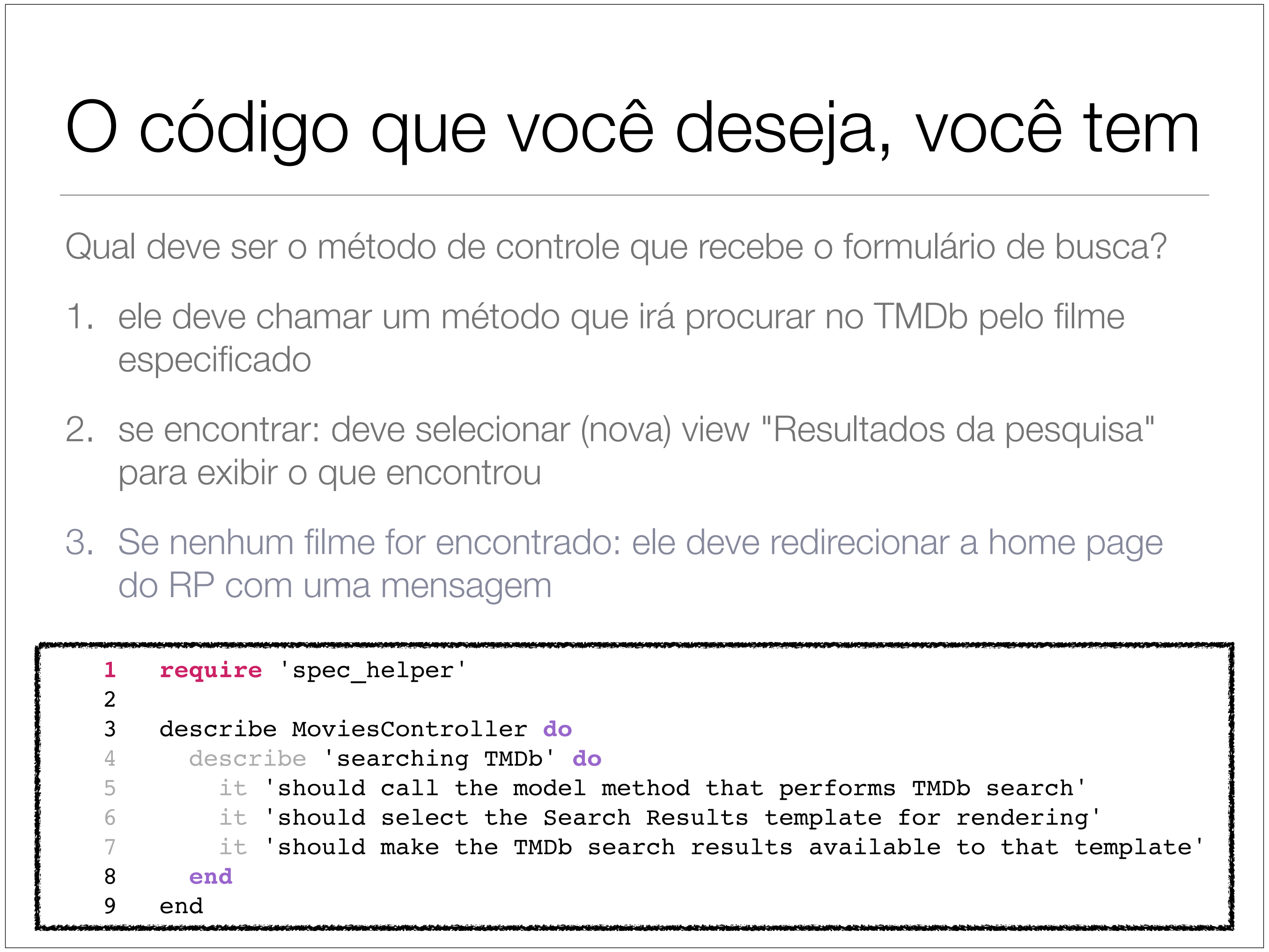
```
When I fill in "Search Terms" with "Inception"
And I press "Search TMDb"
Then I should be on the RottenPotatoes homepage
...
```

Vamos recordar a Rails Cookery #2:

adicionar uma nova feature ==

nova rota + novo método no controlador + nova view

This corresponds to the BDD part of the picture we just saw of how BDD + TDD work together.
the step in RED requires new code to be added to the app, so rest of this section of course is how to use TDD to create that code.
What code do we need? "Rails cookery #2" from Rails lecture reminds us that we need a new route, new controller action to receive the Search form, and a new view to render results.



O código que você deseja, você tem

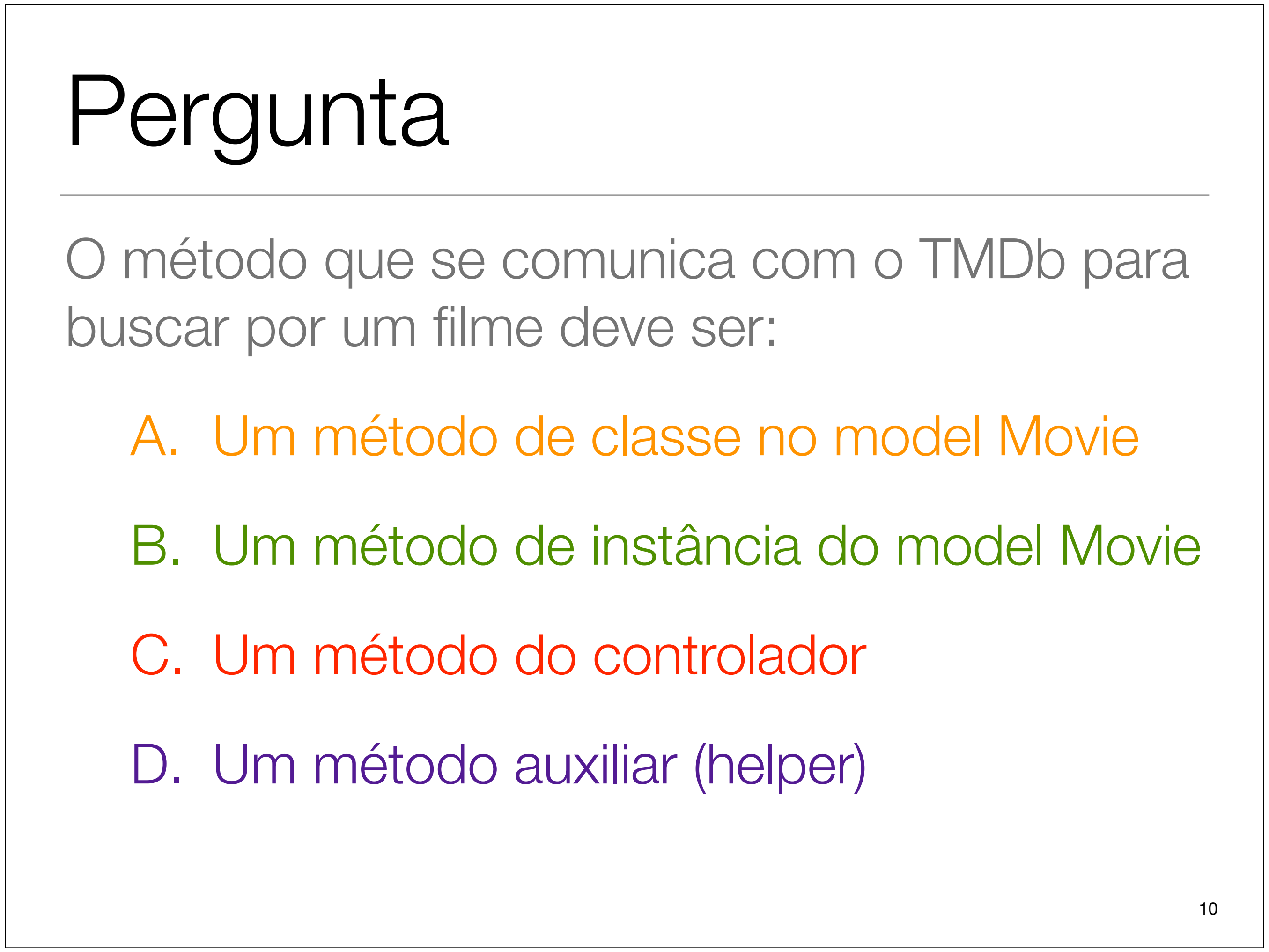
Qual deve ser o método de controle que recebe o formulário de busca?

1. ele deve chamar um método que irá procurar no TMDb pelo filme especificado
2. se encontrar: deve selecionar (nova) view "Resultados da pesquisa" para exibir o que encontrou
3. Se nenhum filme for encontrado: ele deve redirecionar a home page do RP com uma mensagem

```
1   require 'spec_helper'
2
3   describe MoviesController do
4     describe 'searching TMDb' do
5       it 'should call the model method that performs TMDb search'
6       it 'should select the Search Results template for rendering'
7       it 'should make the TMDb search results available to that template'
8     end
9   end
```

http://pastebin.com/kJxjwSF6
step 3 is the sad path – we'll do later.
for now concentrate on happy path.
Suppose controller method ALREADY existed.
Pastebin example shows the code that would test steps 1 & 2 on slide.
WALK THRU:

- line 1 loads helper methods used by all tests; in general should be first line of any spec file
- line 3 says that this group of tests is about the MoviesController class
- since that controller has many actions, line 4 says this subgroup of tests is about the functionality of searching tmdb.
- lines 5-7 are placeholders for the actual test cases, which we'll do next.



Pergunta

O método que se comunica com o TMDb para buscar por um filme deve ser:

A. Um método de classe no model Movie

B. Um método de instância do model Movie

C. Um método do controlador

D. Um método auxiliar (helper)

orange is correct

Pergunta

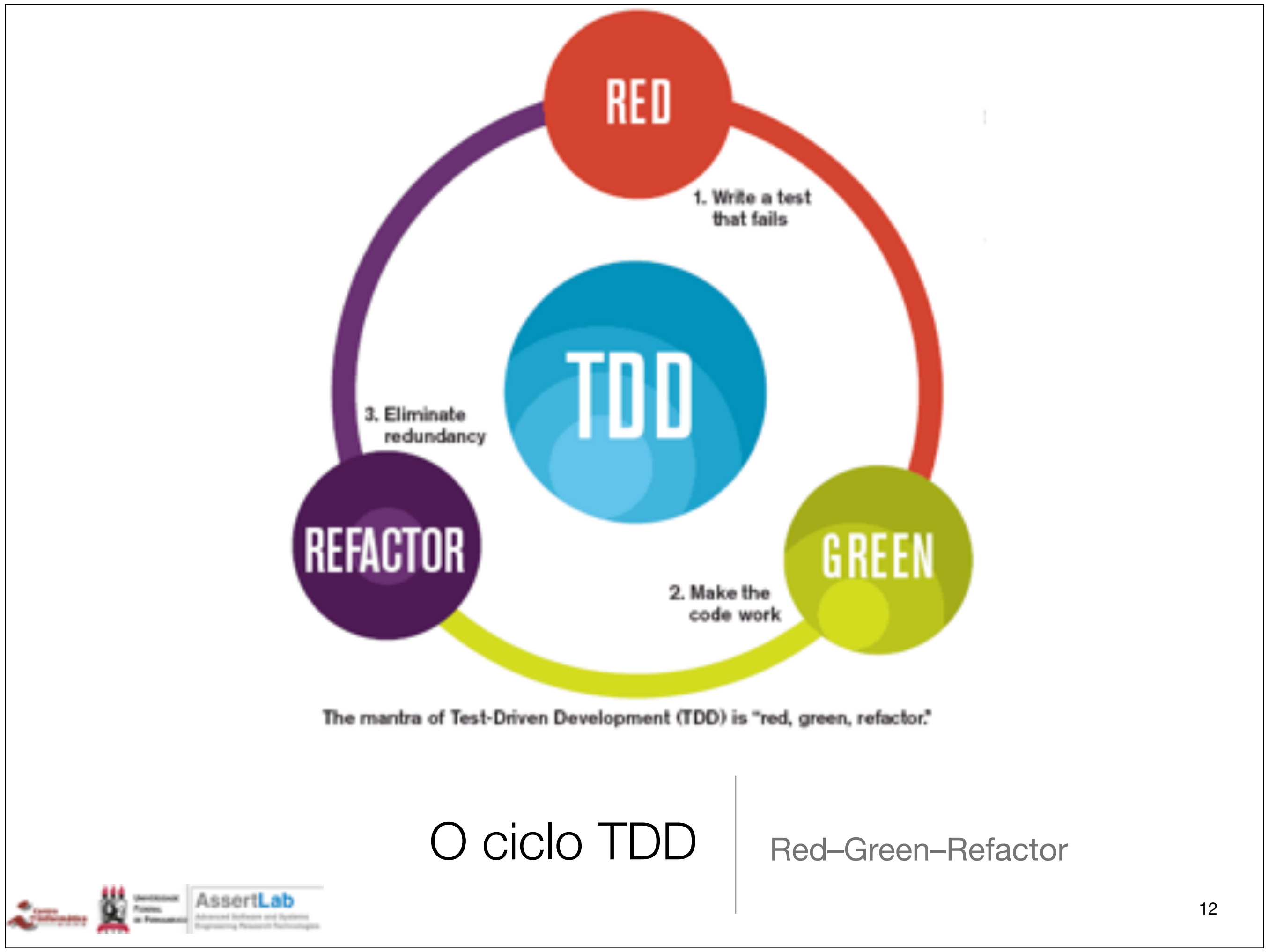
O método que se comunica com o TMDb para buscar por um filme deve ser:

[✓ A. Um método de classe no model Movie]

B. Um método de instância do model Movie

C. Um método do controlador

D. Um método auxiliar (helper)



RED

1. Write a test that fails

TDD

GREEN

2. Make the code work

REFACTOR

3. Eliminate redundancy

The mantra of Test-Driven Development (TDD) is "red, green, refactor."

O ciclo TDD

Red–Green–Refactor

(Engineering Software as a Service §8.3)

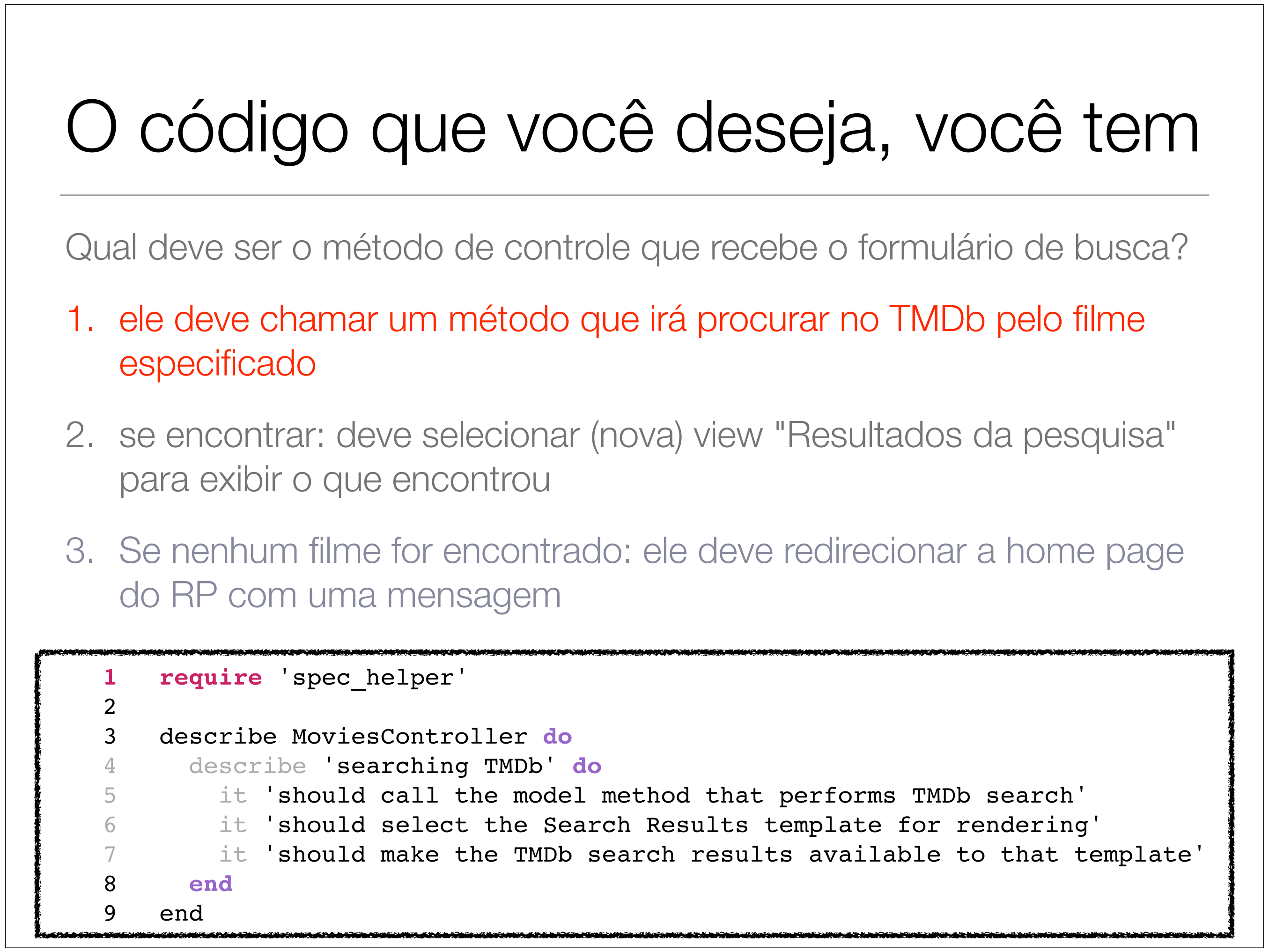
Test-First development

- Pense em uma coisa que o código *deve* fazer
- Capture esse pensamento em um teste, que falha
- Escreva o código mais simples possível que permite a passagem do teste
- Refatorar: implementar DRY com outros testes
- Continue com a próxima “coisa"que o código deve fazer

Red – **Green** – **Refactor**

O objetivo é “sempre ter código funcionando”

> Como testar alguma coisa "isoladamente" se ela possui dependências que afetariam o teste?
>
> –Johnny Appleseed



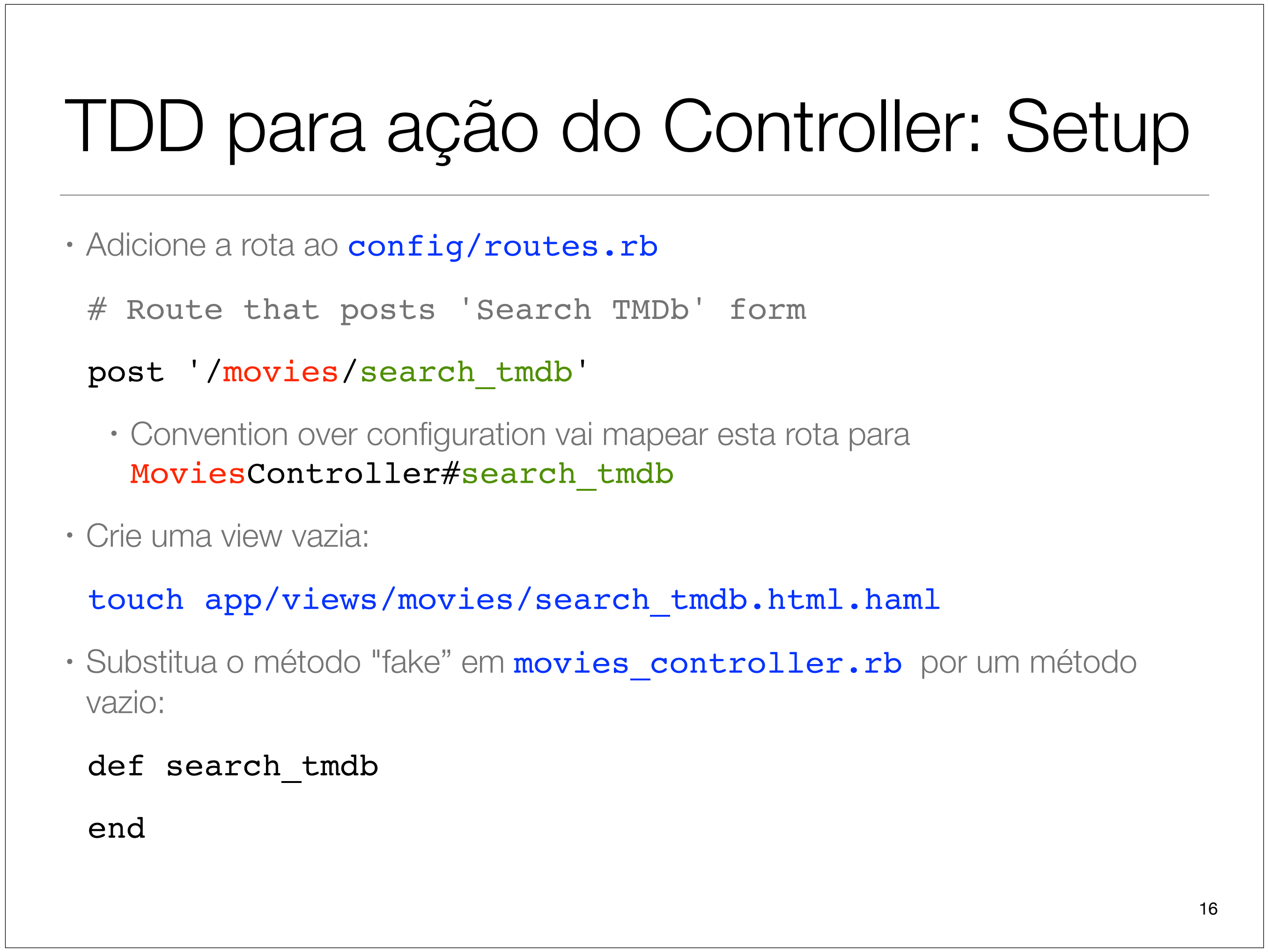
O código que você deseja, você tem

Qual deve ser o método de controle que recebe o formulário de busca?

1. ele deve chamar um método que irá procurar no TMDb pelo filme especificado
2. se encontrar: deve selecionar (nova) view "Resultados da pesquisa" para exibir o que encontrou
3. Se nenhum filme for encontrado: ele deve redirecionar a home page do RP com uma mensagem

```
1   require 'spec_helper'
2
3   describe MoviesController do
4     describe 'searching TMDb' do
5       it 'should call the model method that performs TMDb search'
6       it 'should select the Search Results template for rendering'
7       it 'should make the TMDb search results available to that template'
8     end
9   end
```

http://pastebin.com/kJxjwSF6
step 3 is the sad path – we'll do later.
for now concentrate on happy path.
Suppose controller method ALREADY existed.
Pastebin example shows the code that would test steps 1 & 2 on slide.
WALK THRU:
- line 1 loads helper methods used by all tests; in general should be first line of any spec file
- line 3 says that this group of tests is about the MoviesController class
- since that controller has many actions, line 4 says this subgroup of tests is about the functionality of searching tmdb.
- lines 5-7 are placeholders for the actual test cases, which we'll do next.



TDD para ação do Controller: Setup

- Adicione a rota ao `config/routes.rb`

```
# Route that posts 'Search TMDb' form
post '/movies/search_tmdb'
```

  - Convention over configuration vai mapear esta rota para `MoviesController#search_tmdb`

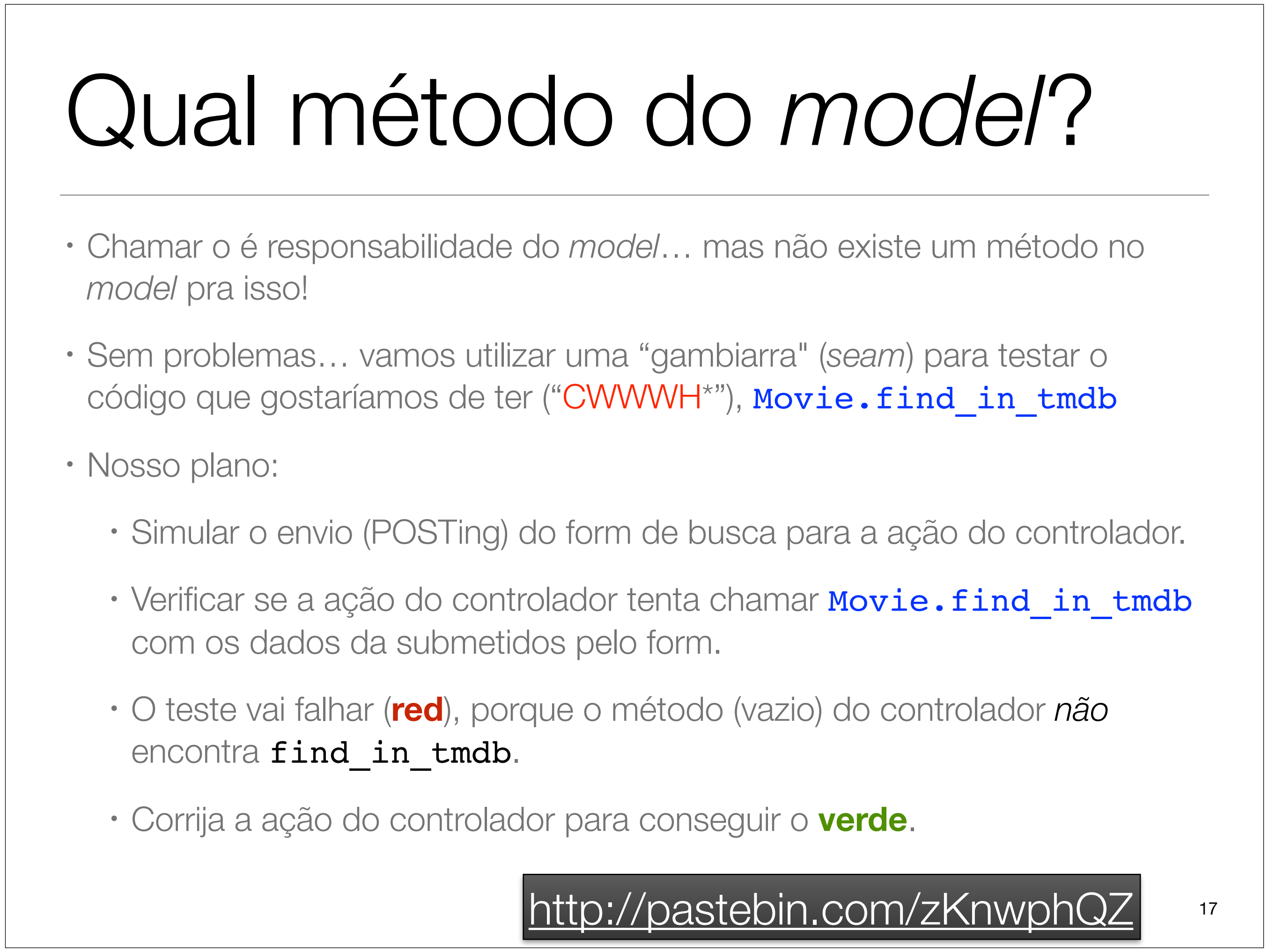
- Crie uma view vazia:

```
touch app/views/movies/search_tmdb.html.haml
```

- Substitua o método "fake" em `movies_controller.rb` por um método vazio:

```
def search_tmdb
end
```

Replacing hardwired method (which was a hack to make sad path of "add movie from TMDb" work in Cucumber) with an empty method, since we will use TDD to drive the creation of the code in that method.



Qual método do *model*?

- Chamar o é responsabilidade do *model*… mas não existe um método no *model* pra isso!
- Sem problemas… vamos utilizar uma “gambiarra" (*seam*) para testar o código que gostaríamos de ter (“CWWWH*”), `Movie.find_in_tmdb`
- Nosso plano:
  - Simular o envio (POSTing) do form de busca para a ação do controlador.
  - Verificar se a ação do controlador tenta chamar `Movie.find_in_tmdb` com os dados da submetidos pelo form.
  - O teste vai falhar (**red**), porque o método (vazio) do controlador *não* encontra `find_in_tmdb`.
  - Corrija a ação do controlador para conseguir o **verde**.

http://pastebin.com/zKnwphQZ

* code we wish we had

WALKTHROUGH OF PASTEBIN CODE:

start with line 7. RSpec provides a "post" method that simulates posting a form.

the first argument is a URI; it will be looked up in routes.rb just like any other URI, as we've seen before.

the second argument (a hash) is what will get stuffed into the params[] hash.

so the effect of line 7 is just as if someone had filled out a form that had a field named 'search_terms', entered the word 'hardware' in that field, and clicked Submit.

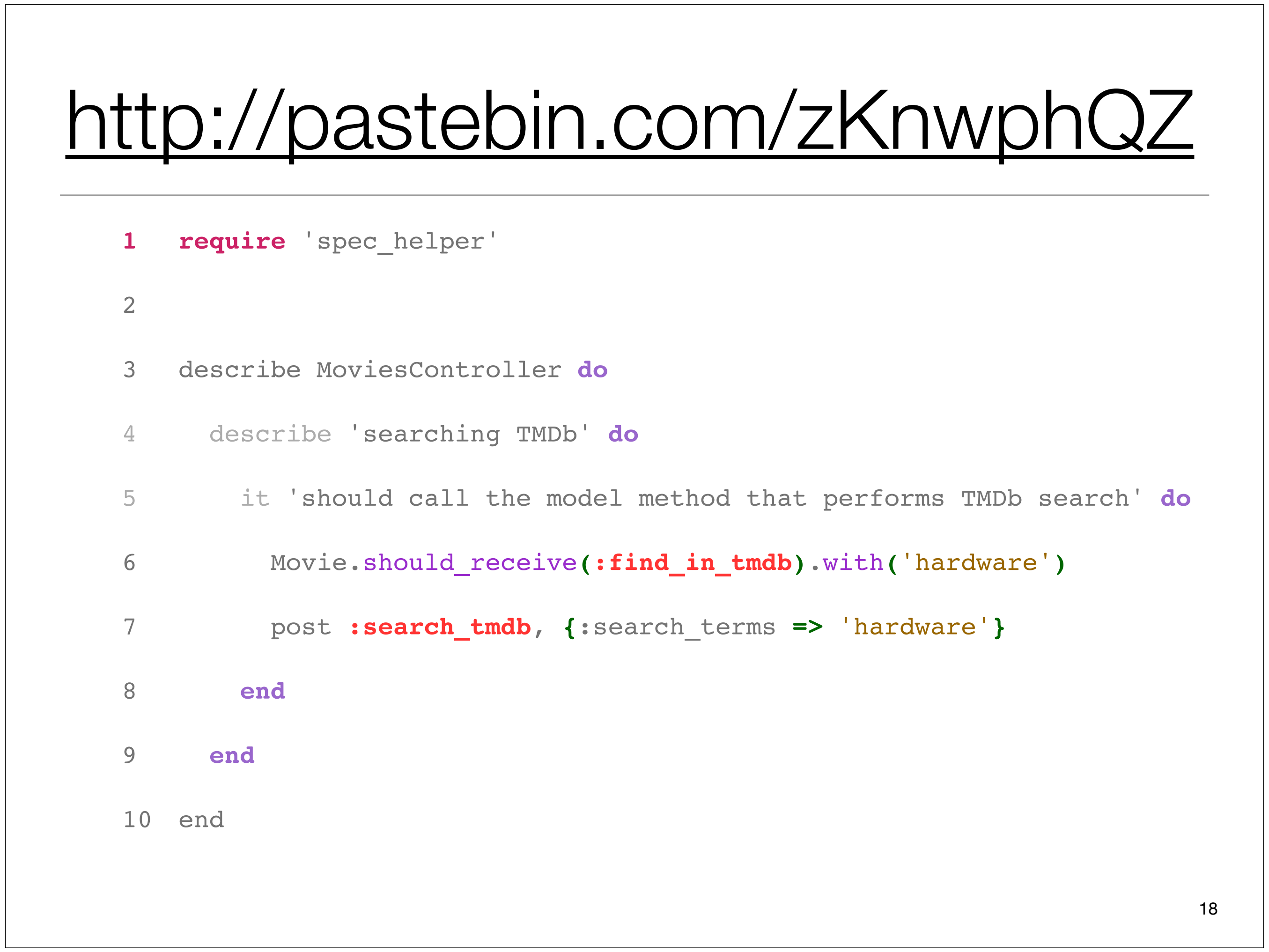
** THIS IS AN IMPORTANT CONCEPT: **

line 6 sets up an _expectation_ of what should happen when line 7 is executed.

should_receive *replaces* any existing method called 'find_in_tmdb' in the Movie class, with a "stub" method whose sole job is to monitor whether it gets called. (In this case, Movie.find_in_tmdb doesn't exist yet, so the "stub" method is the ONLY method. but even if the find_in_tmdb method existed, we'd still want to override it here, because we want to isolate the behavior of THIS test from any possible bugs in find_in_tmdb.

the 'with' method further enforces that not only should find_in_tmdb get called, but what argument it should receive.

the net effect is we have a test that checks whether the search_tmdb controller action, when triggered, tries to call the model method that we will eventually create.



http://pastebin.com/zKnwphQZ

```
1   require 'spec_helper'
2
3   describe MoviesController do
4     describe 'searching TMDb' do
5       it 'should call the model method that performs TMDb search' do
6         Movie.should_receive(:find_in_tmdb).with('hardware')
7         post :search_tmdb, {:search_terms => 'hardware'}
8       end
9     end
10  end
```

- code we wish we had

WALKTHROUGH OF PASTEBIN CODE:

start with line 7. RSpec provides a "post" method that simulates posting a form.

the first argument is a URI; it will be looked up in routes.rb just like any other URI, as we've seen before.

the second argument (a hash) is what will get stuffed into the params[] hash.

so the effect of line 7 is just as if someone had filled out a form that had a field named 'search_terms', entered the word 'hardware' in that field, and clicked Submit.

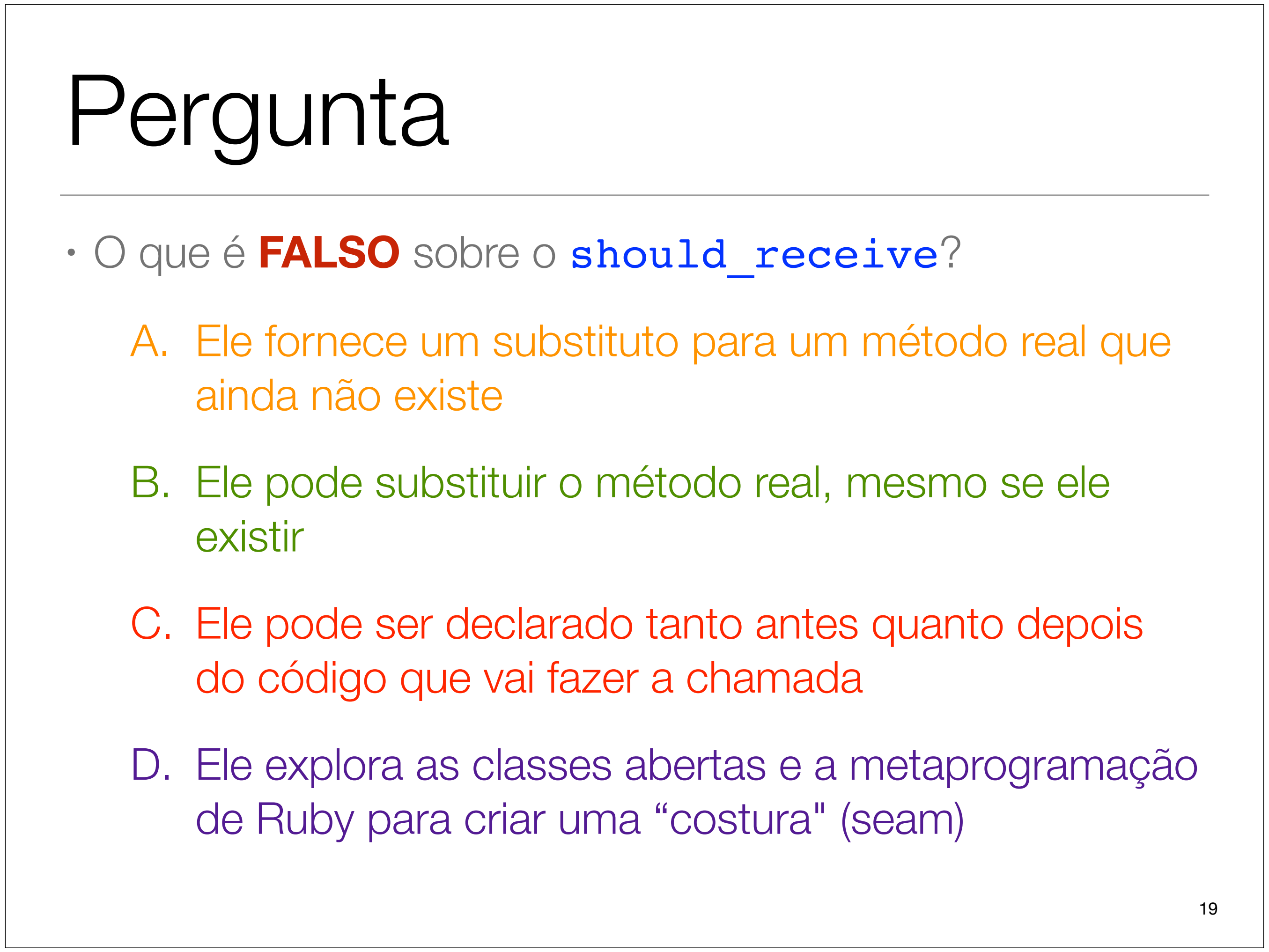
** THIS IS AN IMPORTANT CONCEPT: **

line 6 sets up an _expectation_ of what should happen when line 7 is executed.

should_receive *replaces* any existing method called 'find_in_tmdb' in the Movie class, with a "stub" method whose sole job is to monitor whether it gets called. (In this case, Movie.find_in_tmdb doesn't exist yet, so the "stub" method is the ONLY method. but even if the find_in_tmdb method existed, we'd still want to override it here, because we want to isolate the behavior of THIS test from any possible bugs in find_in_tmdb.

the 'with' method further enforces that not only should find_in_tmdb get called, but what argument it should receive.

the net effect is we have a test that checks whether the search_tmdb controller action, when triggered, tries to call the model method that we will eventually create.



Pergunta

- O que é **FALSO** sobre o `should_receive`?

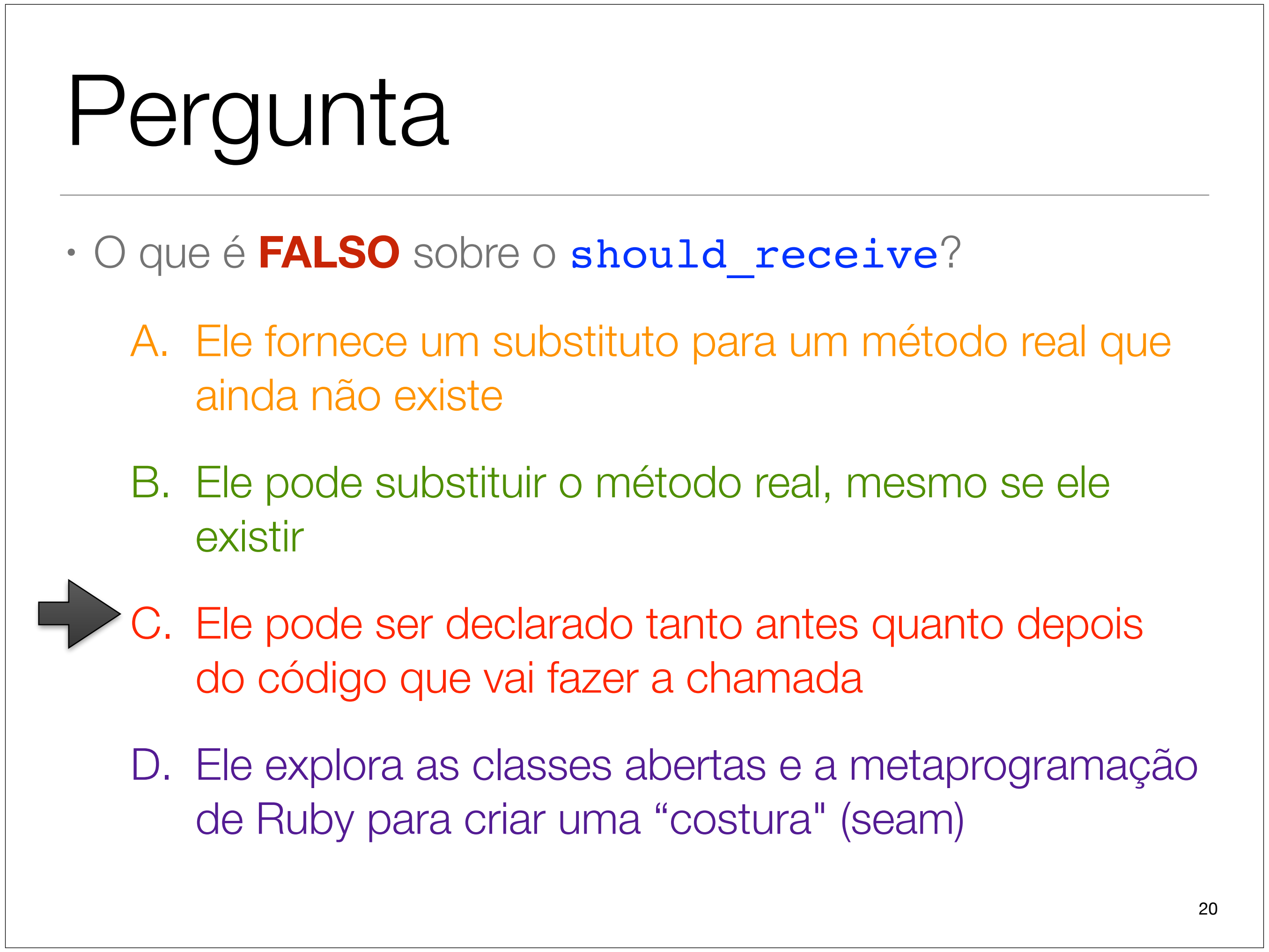
  A. Ele fornece um substituto para um método real que ainda não existe

  B. Ele pode substituir o método real, mesmo se ele existir

  C. Ele pode ser declarado tanto antes quanto depois do código que vai fazer a chamada

  D. Ele explora as classes abertas e a metaprogramação de Ruby para criar uma “costura" (seam)

Correct answer: (c) is false. it must come before the call in order to setup the seam correctly



Pergunta

- O que é **FALSO** sobre o `should_receive`?

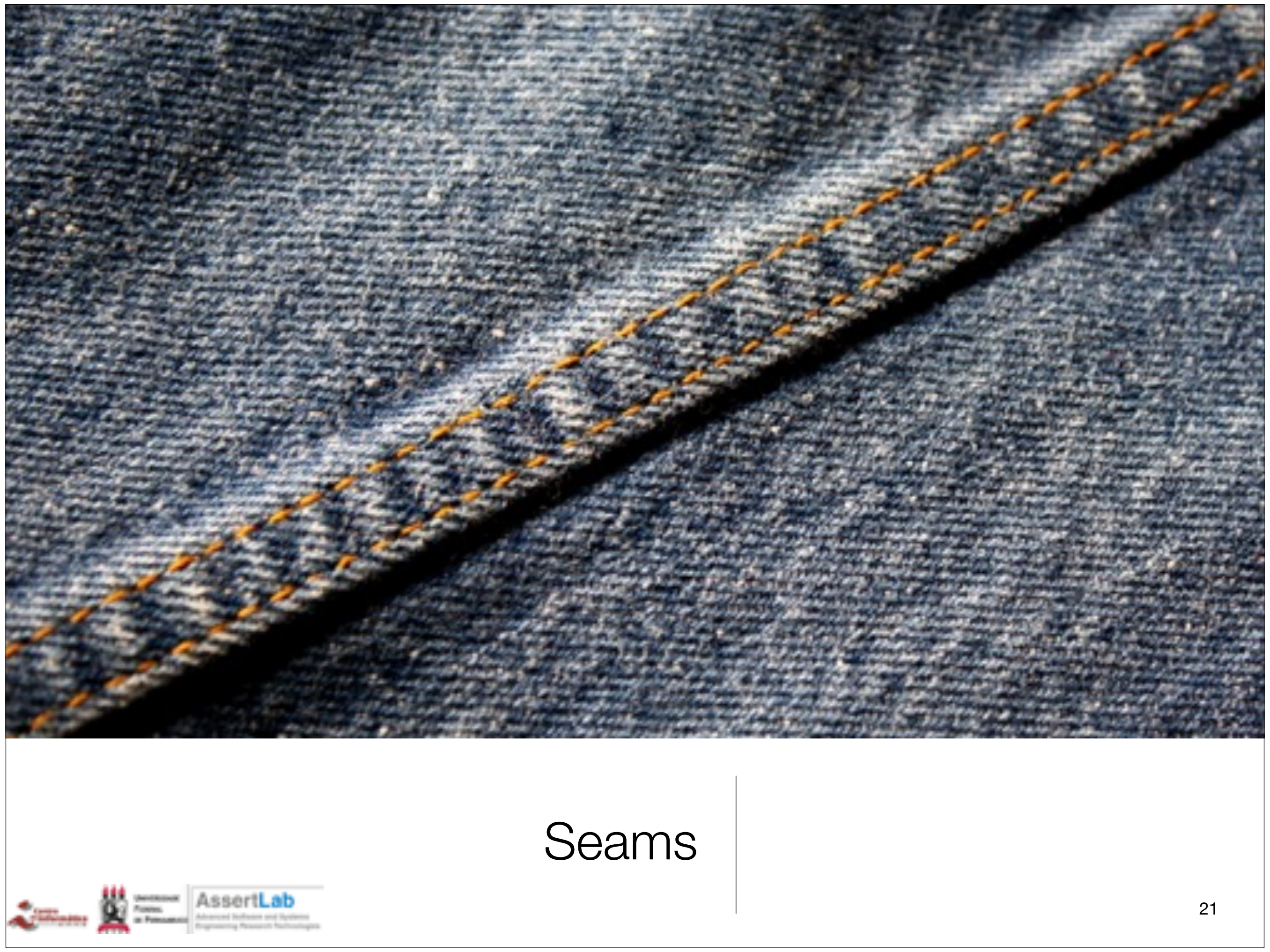
A. Ele fornece um substituto para um método real que ainda não existe

B. Ele pode substituir o método real, mesmo se ele existir

➡ C. Ele pode ser declarado tanto antes quanto depois do código que vai fazer a chamada

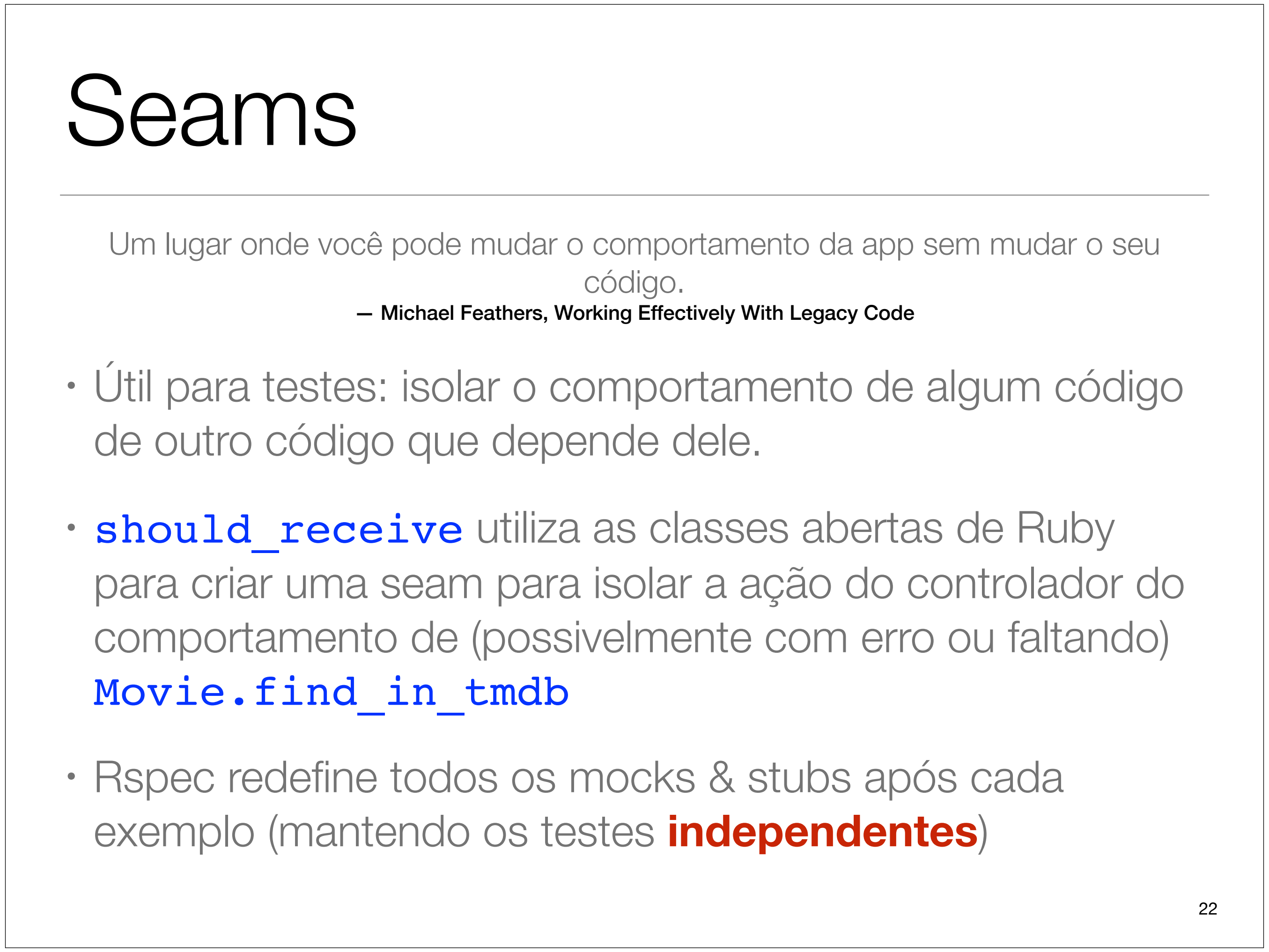
D. Ele explora as classes abertas e a metaprogramação de Ruby para criar uma “costura" (seam)

Correct answer: (c) is false. it must come before the call in order to setup the seam correctly



Seams

(Engineering Software as a Service §8.3)



Seams

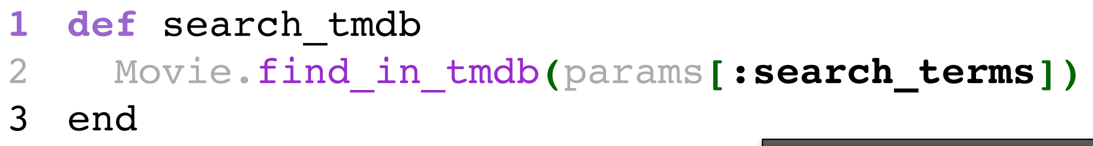
> Um lugar onde você pode mudar o comportamento da app sem mudar o seu código.
>
> — **Michael Feathers, Working Effectively With Legacy Code**

- Útil para testes: isolar o comportamento de algum código de outro código que depende dele.
- `should_receive` utiliza as classes abertas de Ruby para criar uma seam para isolar a ação do controlador do comportamento de (possivelmente com erro ou faltando) `Movie.find_in_tmdb`
- Rspec redefine todos os mocks & stubs após cada exemplo (mantendo os testes **independentes**)

this kind of seam is possible in Ruby because of open classes: you can add or change behaviors of any method at any time, even in another class. RSpec takes advantage of this to make seams very easy to create. You can create a seam almost anywhere to isolate some code under test from the other methods it needs to collaborate with.

Como deixar essa spec verde?

- A expectativa diz que a ação do controlador deve chamar `Movie.find_in_tmdb`
    - Então, vamos chamá-lo!



```
def search_tmdb
  Movie.find_in_tmdb(params[:search_terms])
end
```

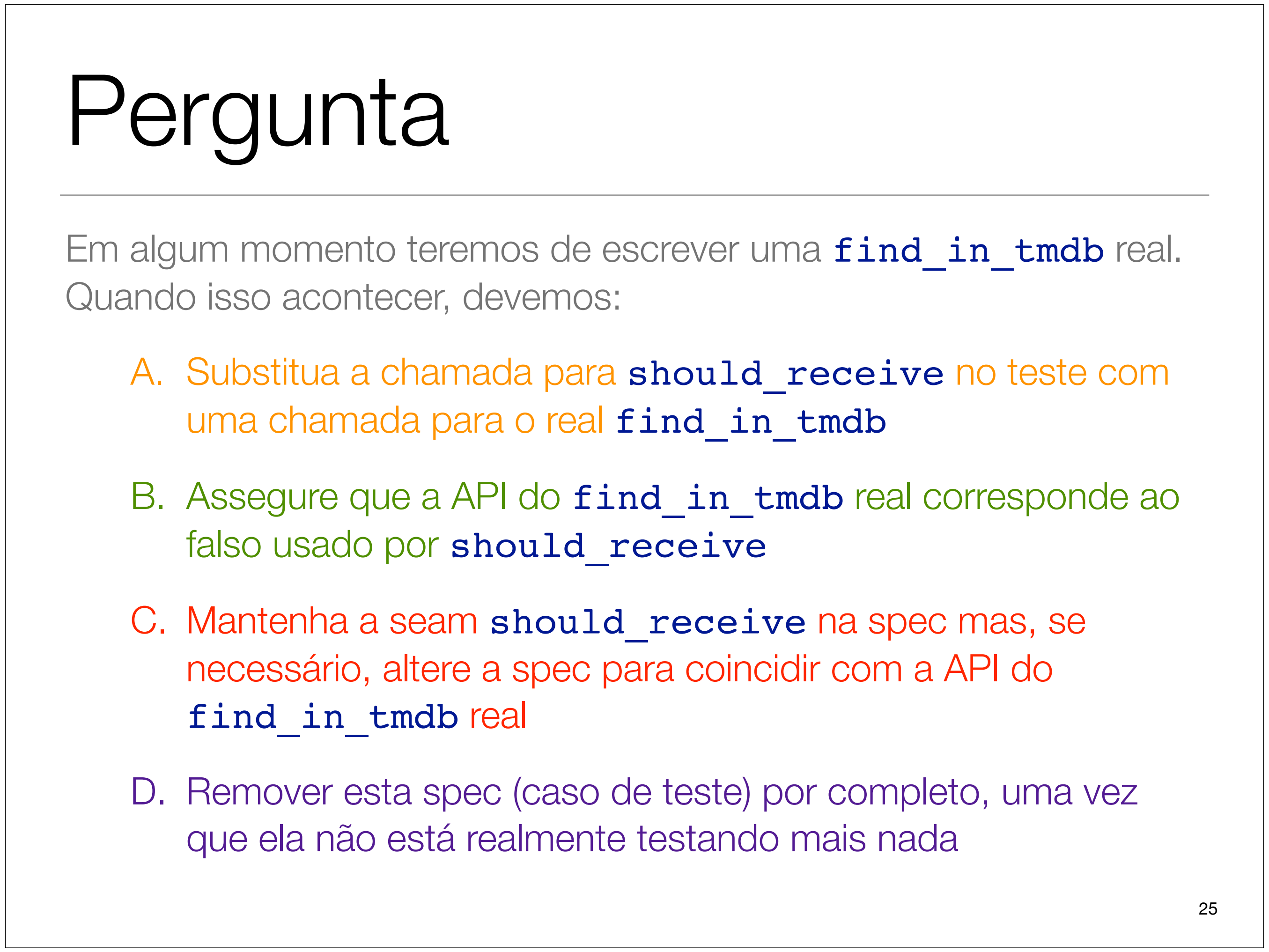
http://pastebin.com/DxzFURiu

- A spec tem impulsionado a criação do método de controle para passar no teste.
    - Mas `find_in_tmdb` não deve retornar algo?

Técnicas de teste que conhecemos

```
obj.should_receive(a).with(b)
```

opcional!



Pergunta

Em algum momento teremos de escrever uma `find_in_tmdb` real. Quando isso acontecer, devemos:

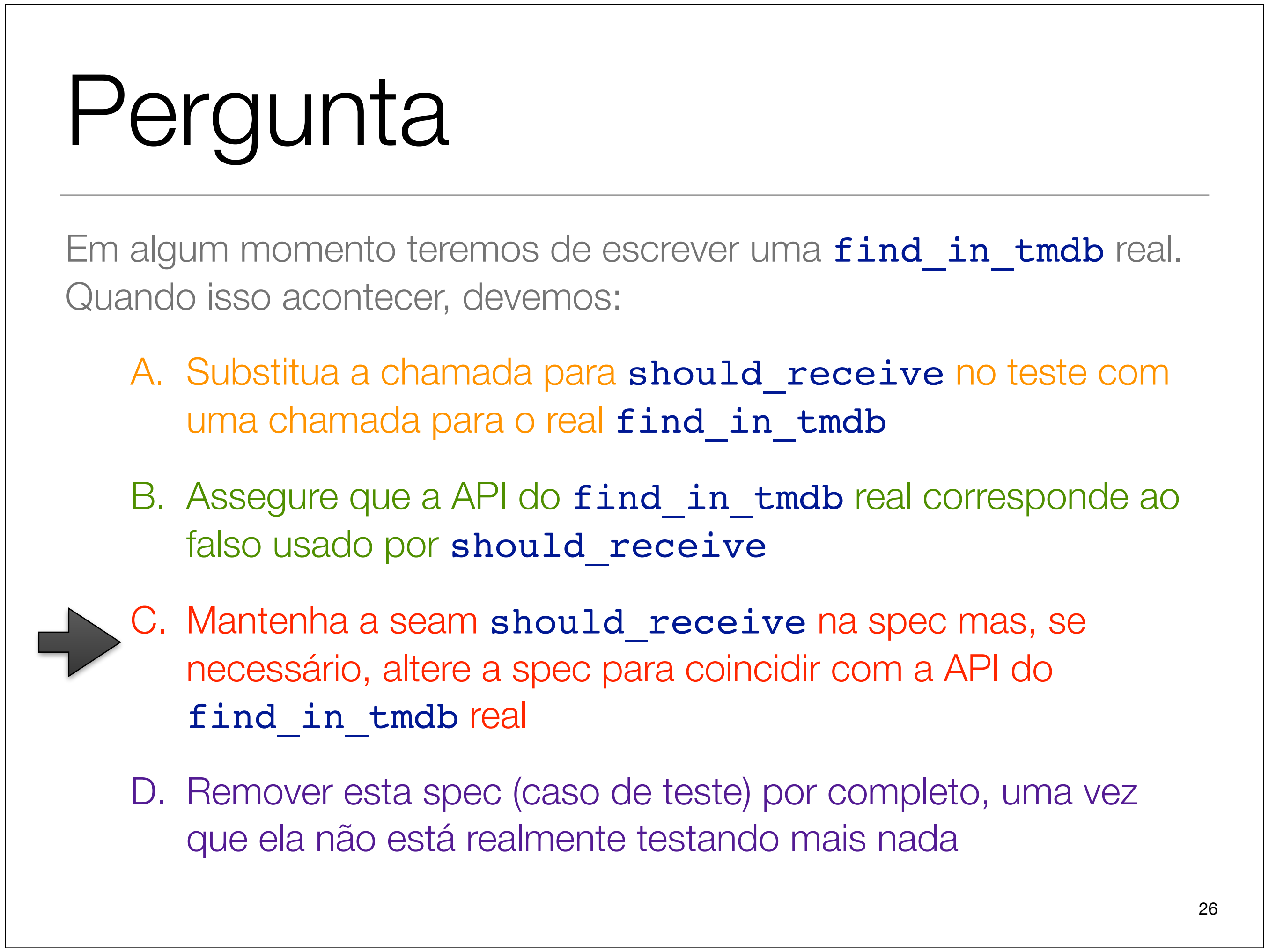
A. Substitua a chamada para `should_receive` no teste com uma chamada para o real `find_in_tmdb`

B. Assegure que a API do `find_in_tmdb` real corresponde ao falso usado por `should_receive`

C. Mantenha a seam `should_receive` na spec mas, se necessário, altere a spec para coincidir com a API do `find_in_tmdb` real

D. Remover esta spec (caso de teste) por completo, uma vez que ela não está realmente testando mais nada

Correct answer: (c) is false. it must come before the call in order to setup the seam correctly



Pergunta

Em algum momento teremos de escrever uma `find_in_tmdb` real. Quando isso acontecer, devemos:

A. Substitua a chamada para `should_receive` no teste com uma chamada para o real `find_in_tmdb`

B. Assegure que a API do `find_in_tmdb` real corresponde ao falso usado por `should_receive`

➡ C. Mantenha a seam `should_receive` na spec mas, se necessário, altere a spec para coincidir com a API do `find_in_tmdb` real

D. Remover esta spec (caso de teste) por completo, uma vez que ela não está realmente testando mais nada

Correct answer: (c) is false. it must come before the call in order to setup the seam correctly

MY EXPECTATIONS JUST GET LOWER AND LOWER.
mimiandeunice.com
THAT'S GREAT!
EVENTUALLY I'LL BE ABLE TO MEET ALL OF THEM!

http://mimiandeunice.com/2011/07/19/expectations/

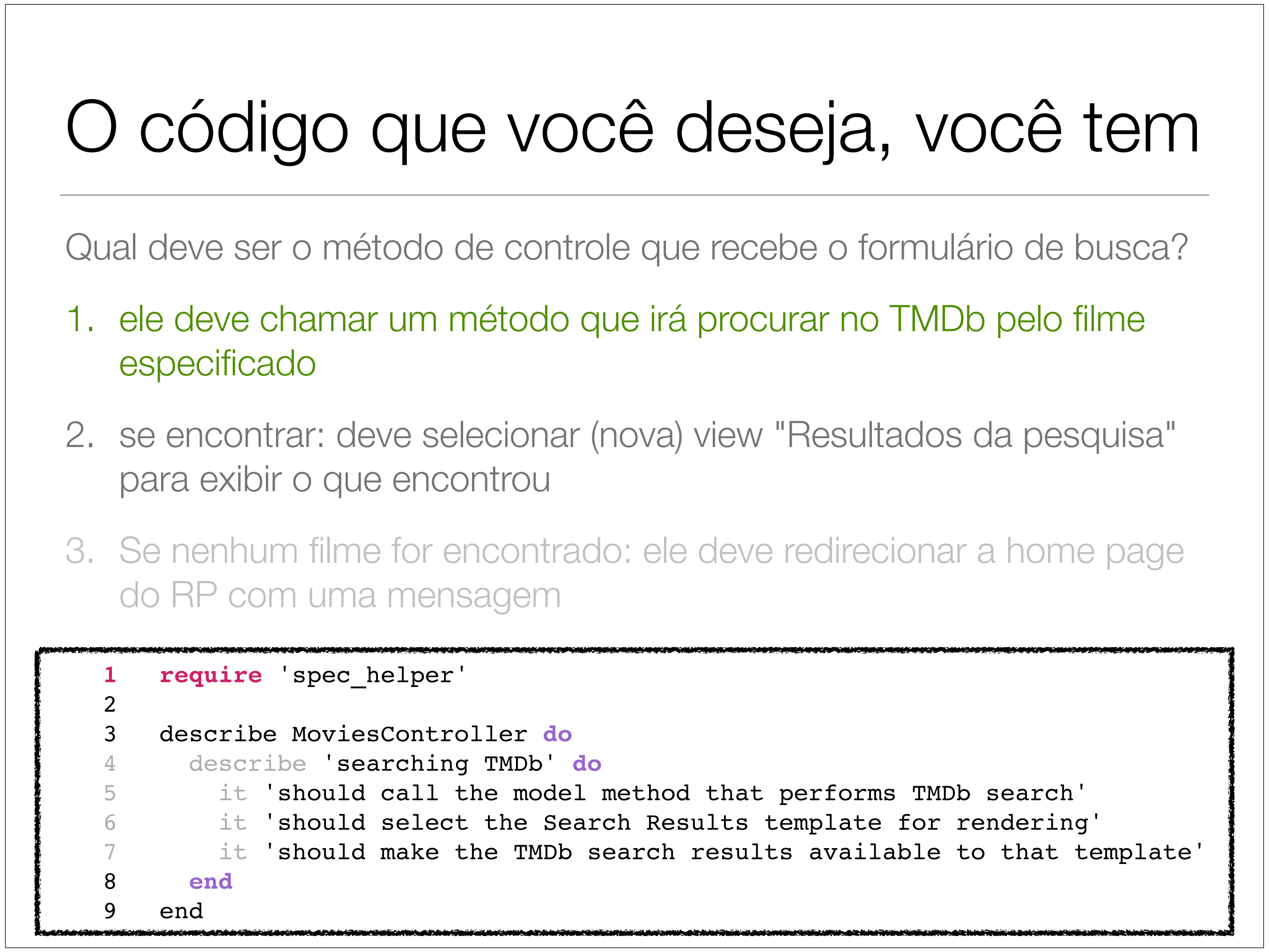
Expectations

(Engineering Software as a Service §8.4)

Onde estamos e para onde estamos indo: “de fora para dentro" no desenvolvimento

- Foco: escrever expectativas que orientam o desenvolvimento do método do controlador
  - Descoberta: devem colaborar com o método do modelo
  - Use "de fora para dentro" de forma recursiva: método stub do modelo neste teste, escrevê-lo mais tarde
- Ideia chave: **quebrar a dependência** entre o método em teste e seus colaboradores
- Conceito-chave: **seam** - onde pode afetar o comportamento do aplicativo sem código de edição

Next: finish controller spec
Preview: more kinds of seams & expectations



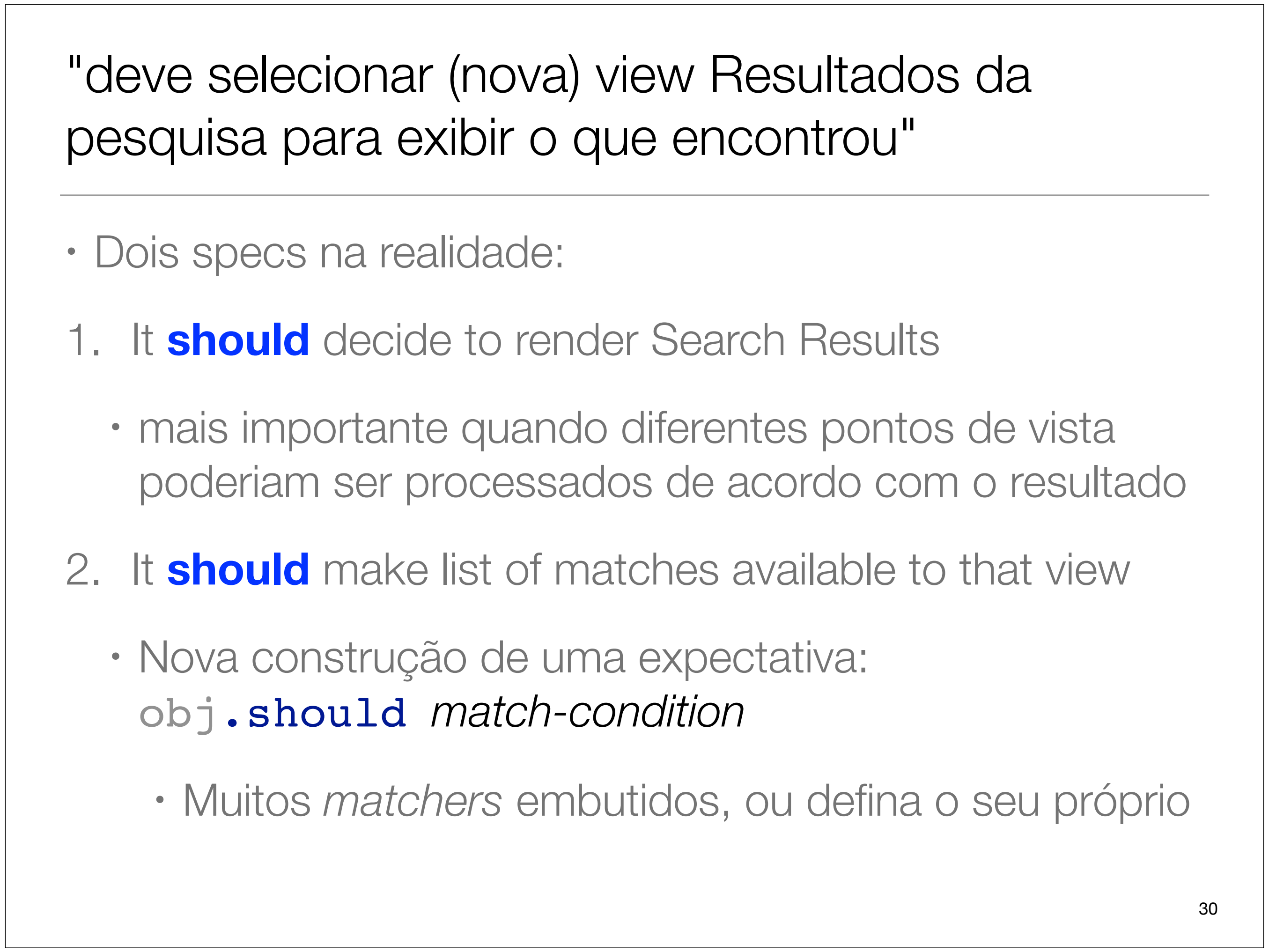
O código que você deseja, você tem

Qual deve ser o método de controle que recebe o formulário de busca?

1. ele deve chamar um método que irá procurar no TMDb pelo filme especificado
2. se encontrar: deve selecionar (nova) view "Resultados da pesquisa" para exibir o que encontrou
3. Se nenhum filme for encontrado: ele deve redirecionar a home page do RP com uma mensagem

```
1   require 'spec_helper'
2
3   describe MoviesController do
4     describe 'searching TMDb' do
5       it 'should call the model method that performs TMDb search'
6       it 'should select the Search Results template for rendering'
7       it 'should make the TMDb search results available to that template'
8     end
9   end
```

http://pastebin.com/kJxjwSF6
step 3 is the sad path – we'll do later.
for now concentrate on happy path.
Suppose controller method ALREADY existed.
Pastebin example shows the code that would test steps 1 & 2 on slide.
WALK THRU:
- line 1 loads helper methods used by all tests; in general should be first line of any spec file
- line 3 says that this group of tests is about the MoviesController class
- since that controller has many actions, line 4 says this subgroup of tests is about the functionality of searching tmdb.
- lines 5-7 are placeholders for the actual test cases, which we'll do next.

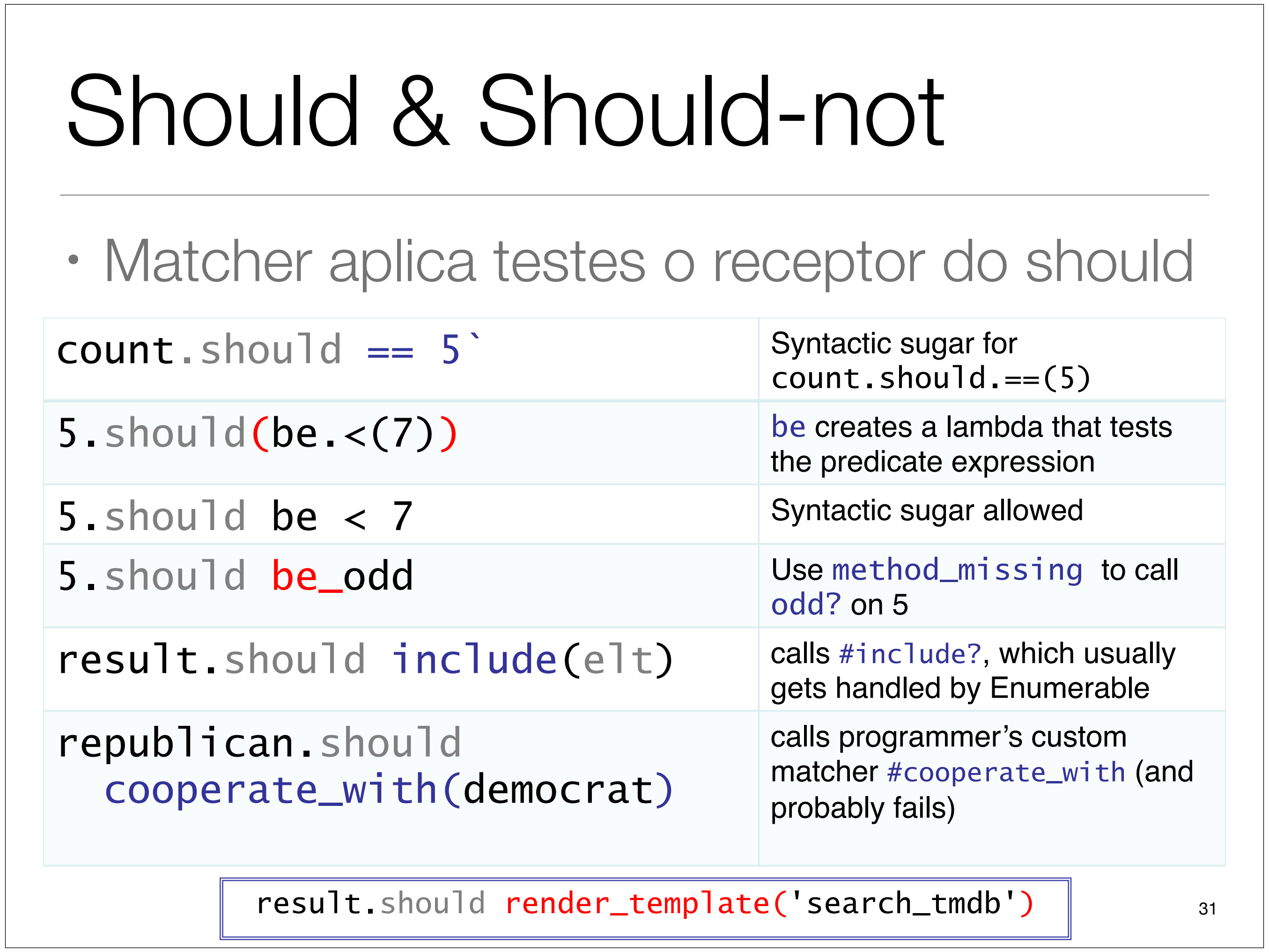


"deve selecionar (nova) view Resultados da pesquisa para exibir o que encontrou"

- Dois specs na realidade:

1. It **should** decide to render Search Results
   - mais importante quando diferentes pontos de vista poderiam ser processados de acordo com o resultado
2. It **should** make list of matches available to that view
   - Nova construção de uma expectativa: `obj.should` *match-condition*
     - Muitos *matchers* embutidos, ou defina o seu próprio

Q: how does RSpec arrange for 'should' to work?



Should & Should-not

- Matcher aplica testes o receptor do should

`count.should == 5``	Syntactic sugar for `count.should.==(5)`
`5.should(be.<(7))`	`be` creates a lambda that tests the predicate expression
`5.should be < 7`	Syntactic sugar allowed
`5.should be_odd`	Use `method_missing` to call `odd?` on 5
`result.should include(elt)`	calls `#include?`, which usually gets handled by Enumerable

`republican.should`
`cooperate_with(democrat)`
	calls programmer's custom matcher `#cooperate_with` (and probably fails)

```
result.should render_template('search_tmdb')
```

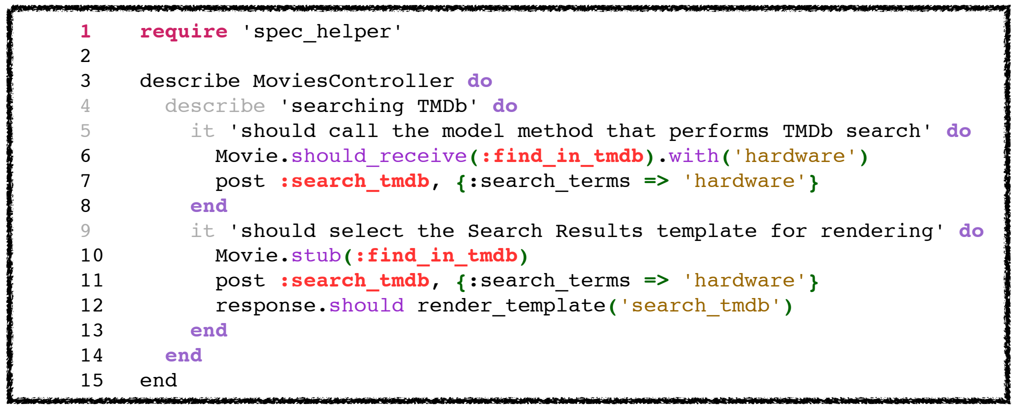
First example is how == is defined on object returned by should

Last line is additional RSpec support for Rails apps (part of rspec-rails gem)

But what should be the receiver for 'result'?

Checking for rendering

- Após o post de `:search_tmdb`, o método `response()` retorna o objeto de resposta do controlador
    - o matcher `render_template` pode verificar qual a view o controlador tentou processar

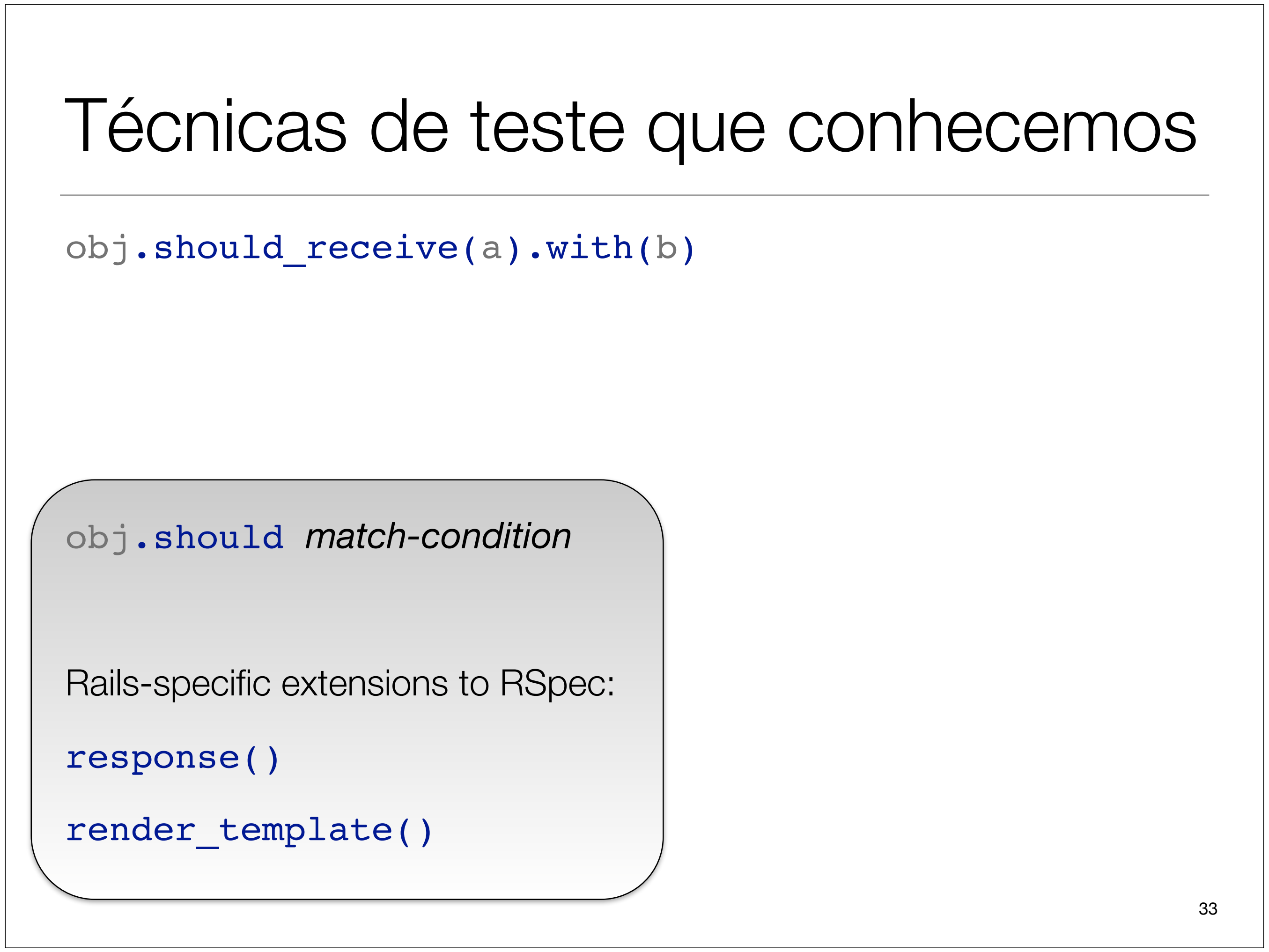


```ruby
require 'spec_helper'

describe MoviesController do
  describe 'searching TMDb' do
    it 'should call the model method that performs TMDb search' do
      Movie.should_receive(:find_in_tmdb).with('hardware')
      post :search_tmdb, {:search_terms => 'hardware'}
    end
    it 'should select the Search Results template for rendering' do
      Movie.stub(:find_in_tmdb)
      post :search_tmdb, {:search_terms => 'hardware'}
      response.should render_template('search_tmdb')
    end
  end
end
```

- Note que esta view deve existir!
    - `post :search_tmdb` vai tentar seguir todo fluxo MVC, incluindo o processamento da view
    - assim, specs do controlador podem ser vistas como testes funcionais

http://pastebin.com/C2x13z8M



Técnicas de teste que conhecemos

```
obj.should_receive(a).with(b)
```

`obj.should` *match-condition*

Rails-specific extensions to RSpec:

```
response()
```

```
render_template()
```

Difference between stub and should_receive

Pergunta

Qual destes, se houver, não é um uso válido de `should` ou `should_not`?

A. `result.should_not be_empty`

B. `5.should be <=> result`

C. `result.should_not match /^D'oh!$/`

D. Todas as anteriores são usos válidos

Pergunta

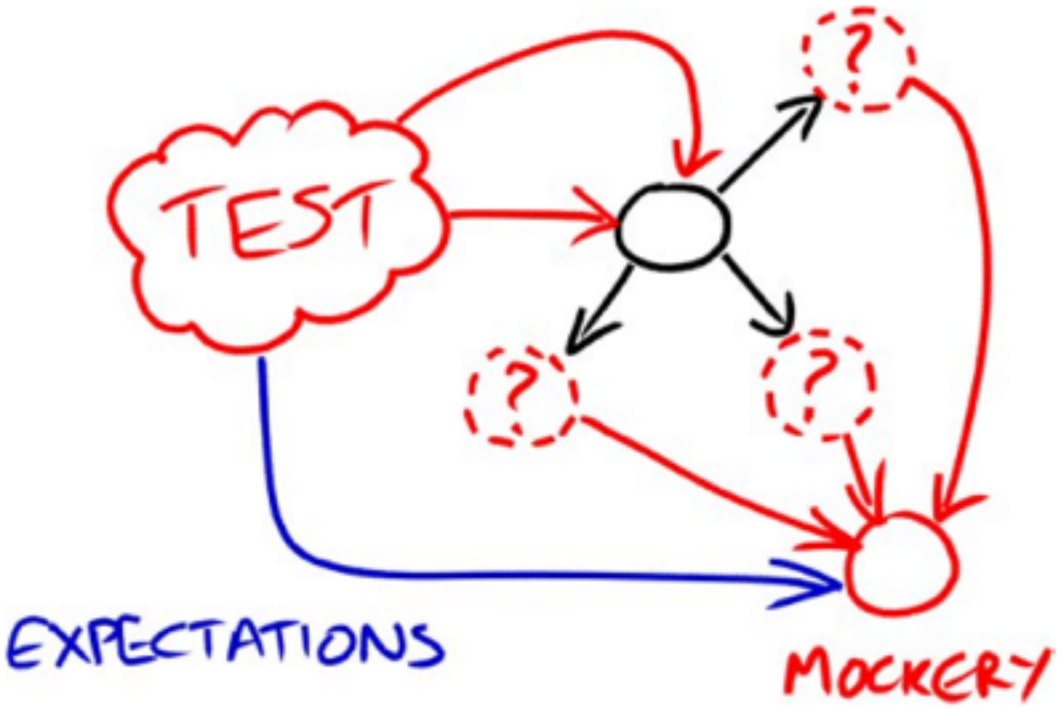
Qual destes, se houver, não é um uso válido de `should` ou `should_not`?

A. `result.should_not be_empty`

[✓ B. `5.should be <=> result`]

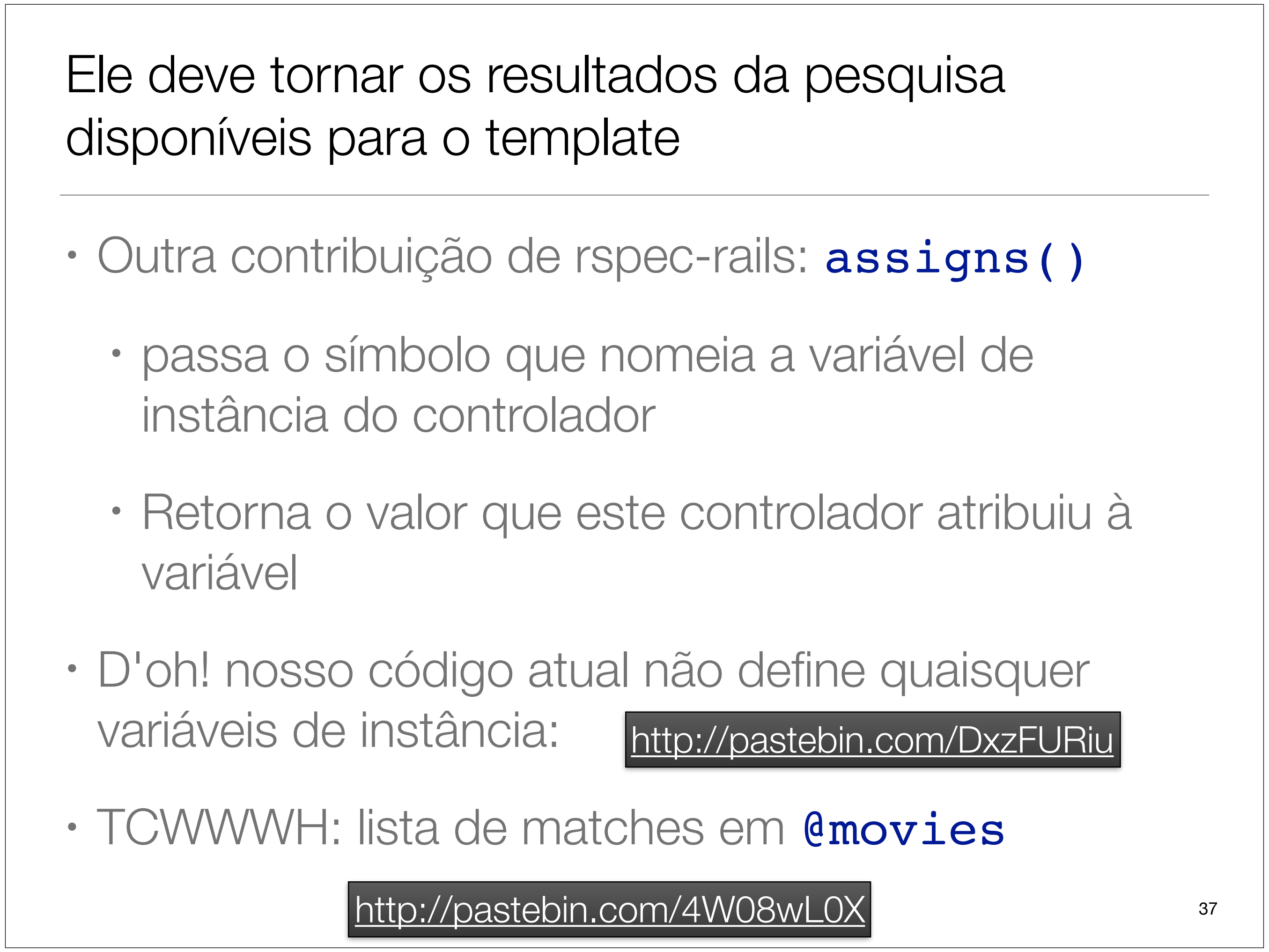
C. `result.should_not match /^D'oh!$/`

D. Todas as anteriores são usos válidos



Mocks, Stubs e Test Setup

(Engineering Software as a Service §8.4)



Ele deve tornar os resultados da pesquisa disponíveis para o template

- Outra contribuição de rspec-rails: `assigns()`
  - passa o símbolo que nomeia a variável de instância do controlador
  - Retorna o valor que este controlador atribuiu à variável
- D'oh! nosso código atual não define quaisquer variáveis de instância: http://pastebin.com/DxzFURiu
- TCWWWH: lista de matches em `@movies`

http://pastebin.com/4W08wL0X

DRY out code using Before block while we're at it

Dois novos conceitos de seams

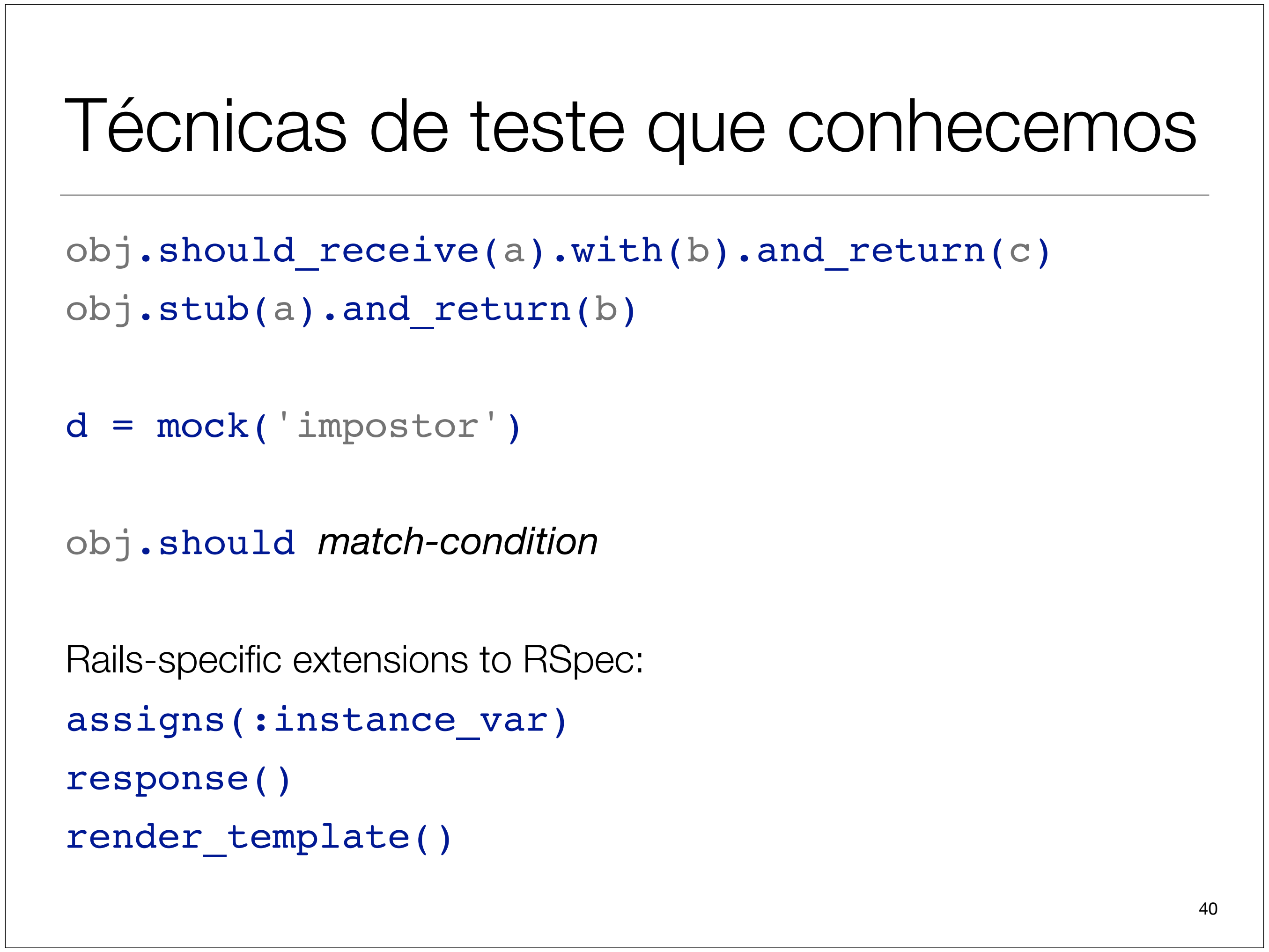
- `stub`
  - similar ao `should_receive`, mas não é expectativa
  - `and_return` opcionalmente controla o valor de retorno
- `mock`: objeto "stunt double", muitas vezes utilizados para verificação de comportamento (o método foi chamado)
  - pode fazer stub de métodos nele

```
m=mock('movie1', :title=>'Rambo')
```

cada seam permite apenas as funcionalidades suficientes para alguns comportamentos específicos em teste

RSpec Cookery #1

- Cada spec deve testar apenas um comportamento
- Use seams conforme necessário para isolar este comportamento
- Determinar que tipo de expectativa irá verificar o comportamento
- Escreva o teste e verifique se ele falha pelo motivo certo
- Adicione código até o teste ficar verde
- Procure oportunidades para refatorar/embelezar



Técnicas de teste que conhecemos

```
obj.should_receive(a).with(b).and_return(c)
obj.stub(a).and_return(b)
```

```
d = mock('impostor')
```

`obj.should` *match-condition*

Rails-specific extensions to RSpec:

```
assigns(:instance_var)
response()
render_template()
```

Difference between stub and should_receive

Pergunta

`should_receive` combina _____ e _____,
enquanto que `stub` é somente _____.

A. Um mock e uma expectativa;
um mock

B. Um mock e uma expectativa;
uma expectativa

C. Uma seam e uma expectativa;
uma expectativa

D. Uma seam e uma expectativa;
uma seam

Pergunta

`should_receive` combina _____ e _____,
enquanto que `stub` é somente _____.

A. Um mock e uma expectativa;
um mock

B. Um mock e uma expectativa;
uma expectativa

C. Uma seam e uma expectativa;
uma expectativa

➡ D. Uma seam e uma expectativa;
uma seam

Fixtures and Factories

(Engineering Software as a Service §8.5)

When you need the real thing

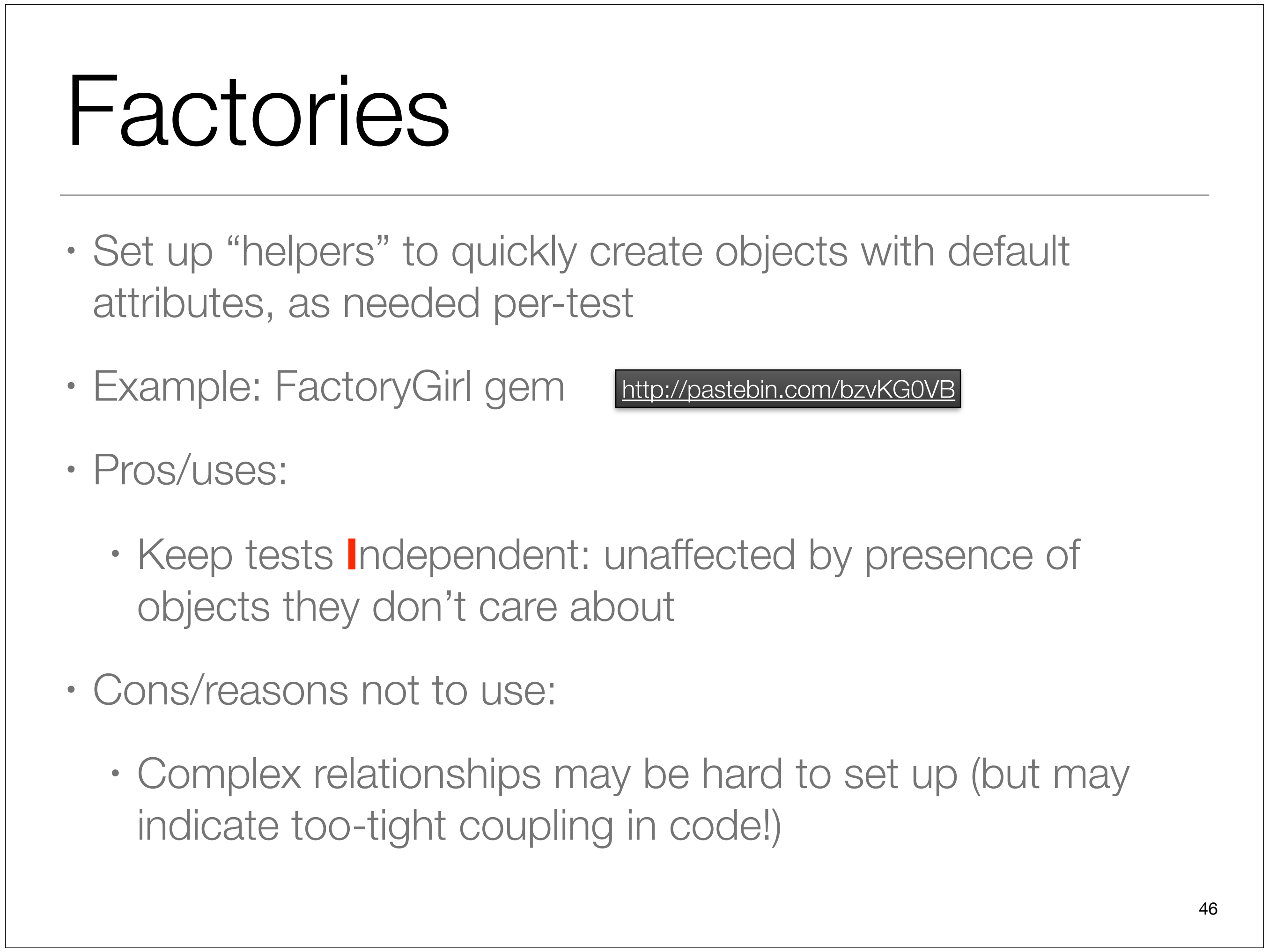
- Where to get a real object:

http://pastebin.com/N3s1A193

  - **Fixture**: statically preload some known data into database tables
  - **Factory**: create only what you need per-test

Fixtures

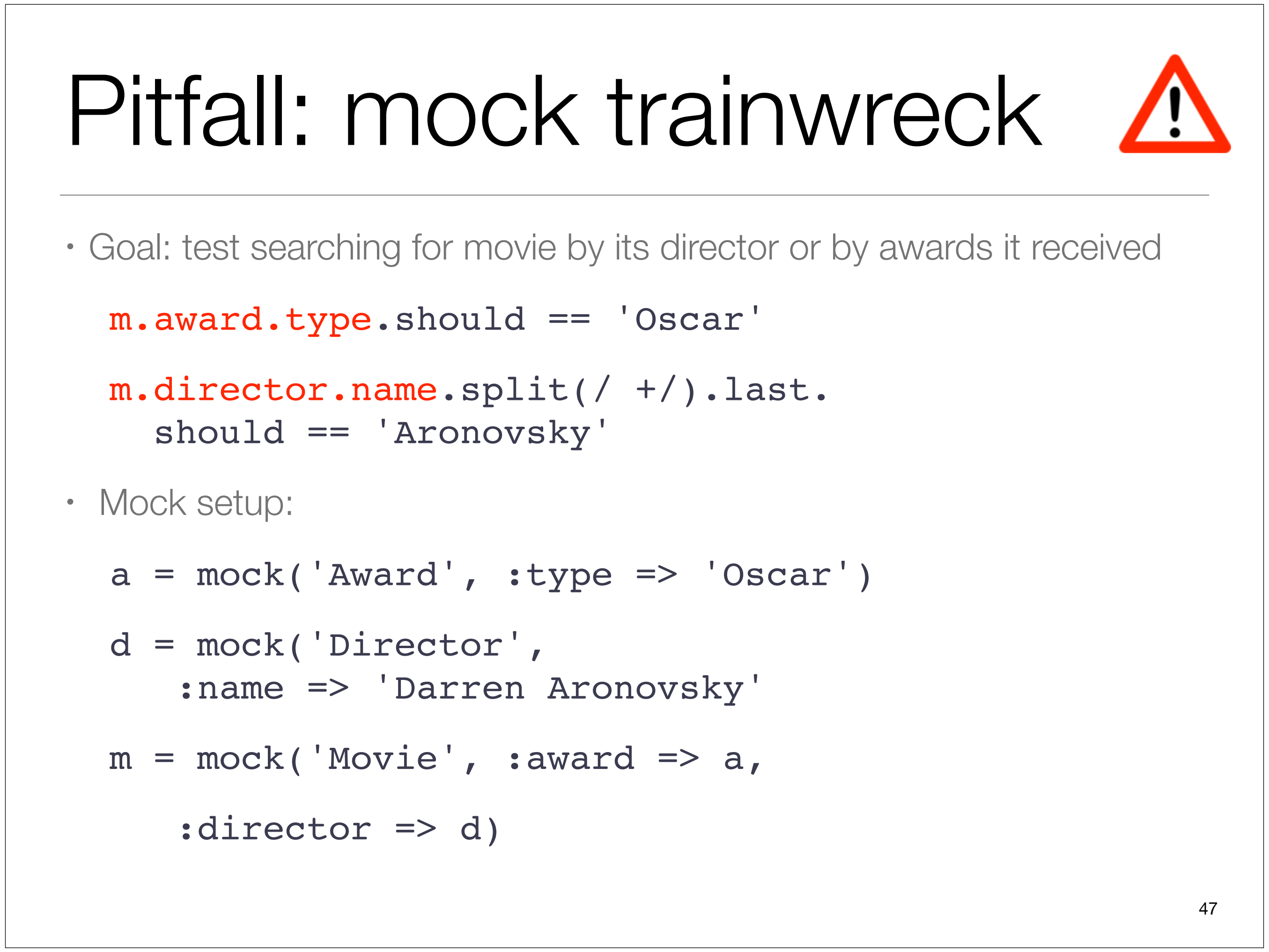
- database wiped & reloaded before each spec
  - add `fixtures :movies` at beginning of `describe`
  - `spec/fixtures/movies.yml` are `Movies` and will be added to `movies` table
- Pros/uses
  - truly static data, e.g. configuration info that never changes
  - easy to see all test data in one place
- Cons/reasons not to use
  - may introduce dependency on fixture data



Factories

- Set up “helpers” to quickly create objects with default attributes, as needed per-test
- Example: FactoryGirl gem http://pastebin.com/bzvKG0VB
- Pros/uses:
  - Keep tests Independent: unaffected by presence of objects they don’t care about
- Cons/reasons not to use:
  - Complex relationships may be hard to set up (but may indicate too-tight coupling in code!)

Don't forget how to use a Gem: add to Gemfile, if appropriate under :test group only, then 'bundle install'



Pitfall: mock trainwreck ⚠

- Goal: test searching for movie by its director or by awards it received

```
m.award.type.should == 'Oscar'

m.director.name.split(/ +/).last.
  should == 'Aronovsky'
```

- Mock setup:

```
a = mock('Award', :type => 'Oscar')

d = mock('Director',
   :name => 'Darren Aronovsky'

m = mock('Movie', :award => a,

   :director => d)
```

If a movie's award type or last name of its director are important entitites in a testing scenario, Movie class should abstract how to get those things.
We will learn a design pattern called 'delegation' that helps with this, among other things.

Pergunta

Which of the following kinds of data, if any, should not be set up as fixtures?

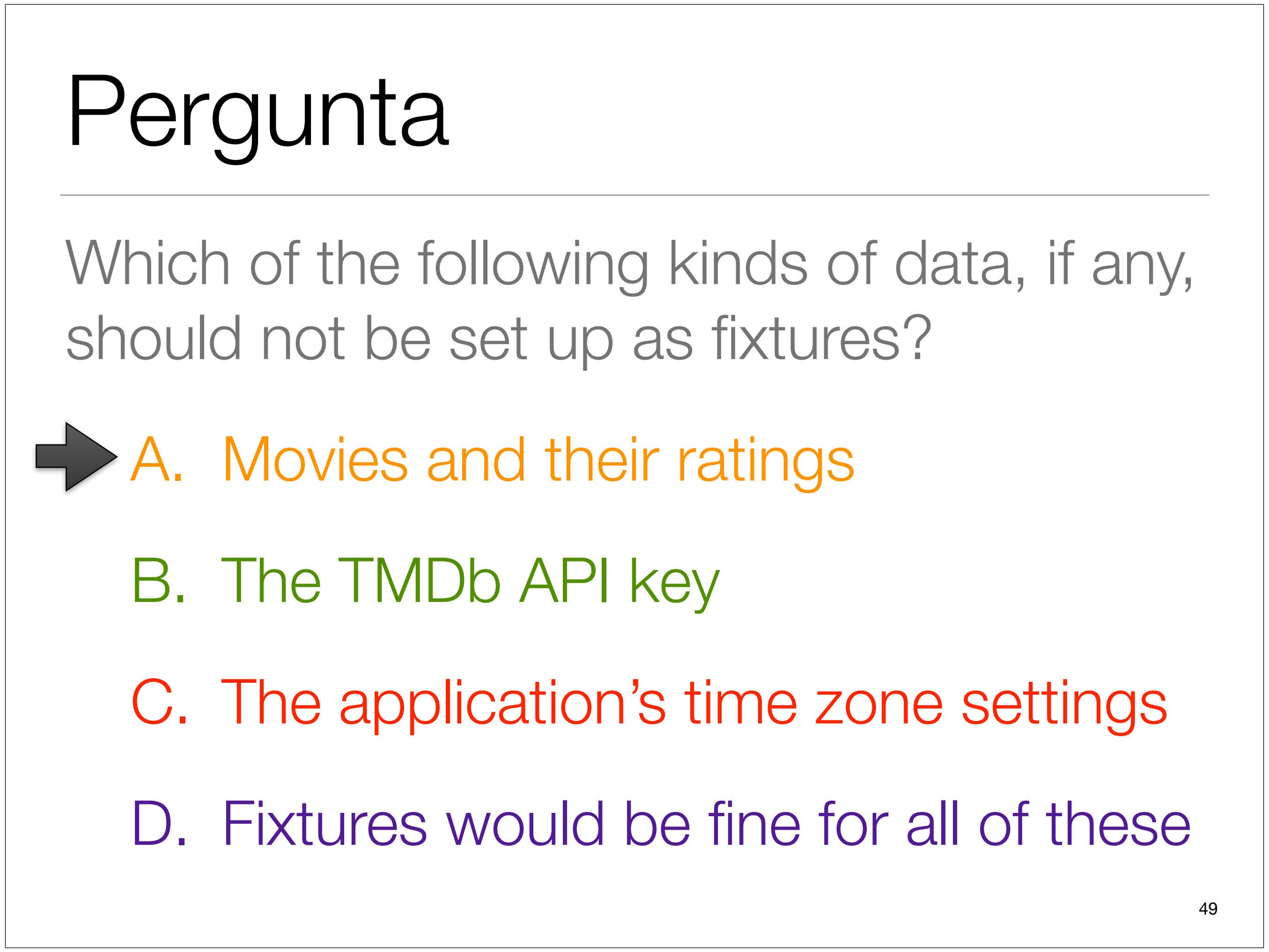
A. Movies and their ratings

B. The TMDb API key

C. The application’s time zone settings

D. Fixtures would be fine for all of these

orange is the answer



Pergunta

Which of the following kinds of data, if any, should not be set up as fixtures?

➡ A. Movies and their ratings

B. The TMDb API key

C. The application’s time zone settings

D. Fixtures would be fine for all of these

orange is the answer

Coverage, Unit vs. Integration Tests

How much testing is enough?

- Bad: “Until time to ship”
- A bit better: (Lines of test) / (Lines of code)
  - 1.2–1.5 not unreasonable
  - often much higher for production systems
- Better question: “How thorough is my testing?”
  - Formal methods
  - Coverage measurement
  - We focus on the latter, though the former is gaining steady traction

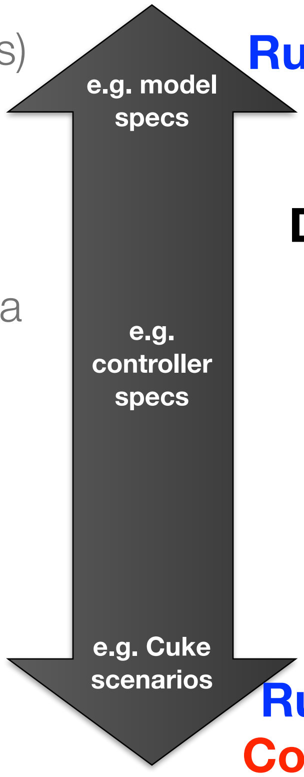
Measuring Coverage - Basics

```
class MyClass
 def foo(x,y,z)
  if x
   if (y && z) then bar(0) end
  else
   bar(1)
  end
 end
 def bar(x) ; @w = x ; end
end
```

- S0: every method called
- S1: every method from every call site
- C0: every statement
  - Ruby SimpleCov gem
- C1: every branch in both directions
- C1+decision coverage: every subexpression in conditional
- C2: every path (difficult, and disagreement on how valuable)

What kinds of tests?

- Unit (one method/class)
- Functional or module (a few methods/classes)
- Integration/system



Runs fast High coverage

Fine resolution

Many mocks;
Doesn’t test interfaces

Few mocks;
tests interfaces

Runs slow Low coverage

Coarse resolution

Going to extremes

☒ “I kicked the tires, it works”

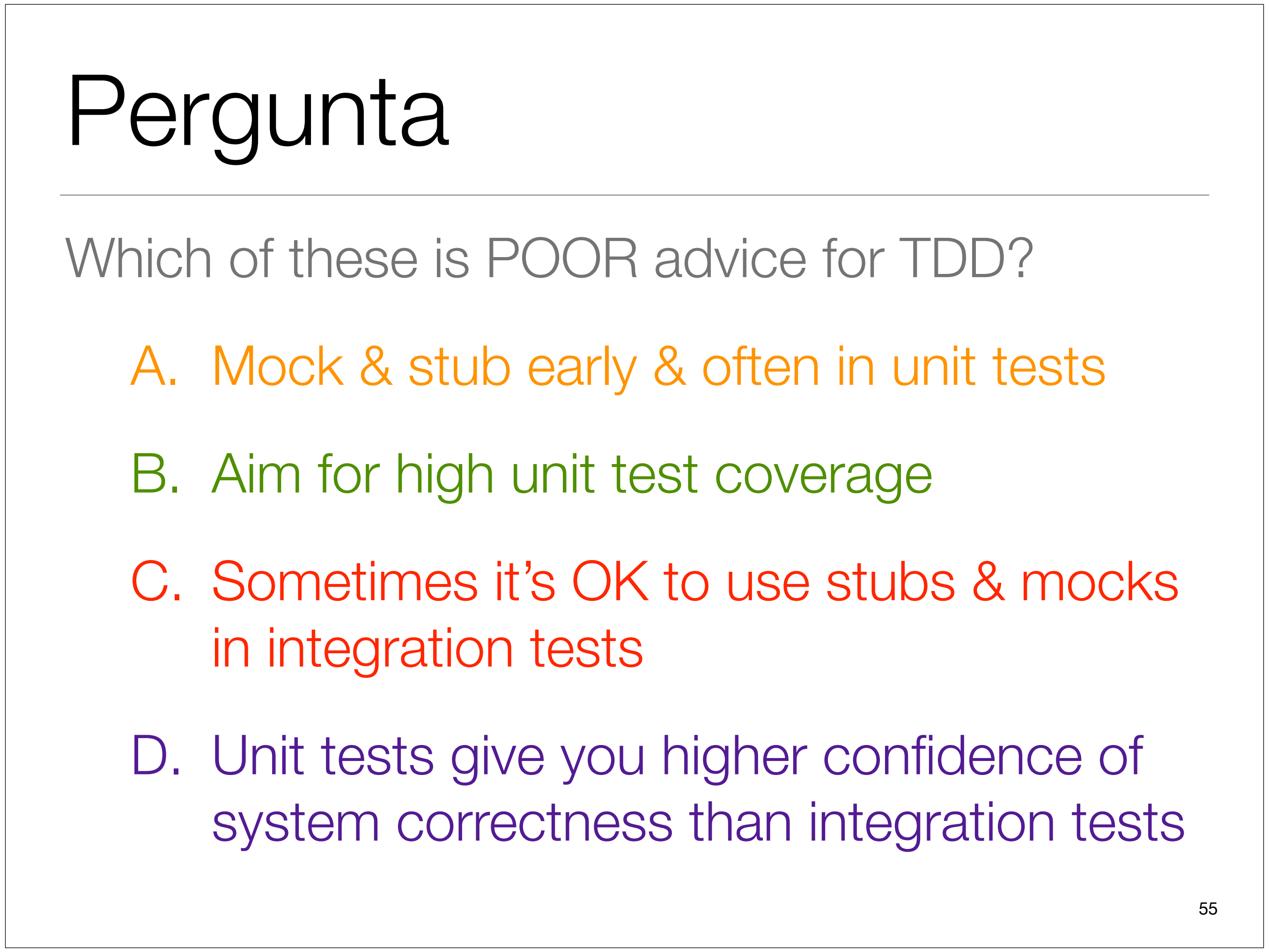
☒ “Don’t ship until 100% covered & green”

☑ use coverage to identify untested or undertested parts of code

☒ “Focus on unit tests, they’re more thorough”

☒ “Focus on integration tests, they’re more realistic”

☑ each finds bugs the other misses



Pergunta

Which of these is POOR advice for TDD?

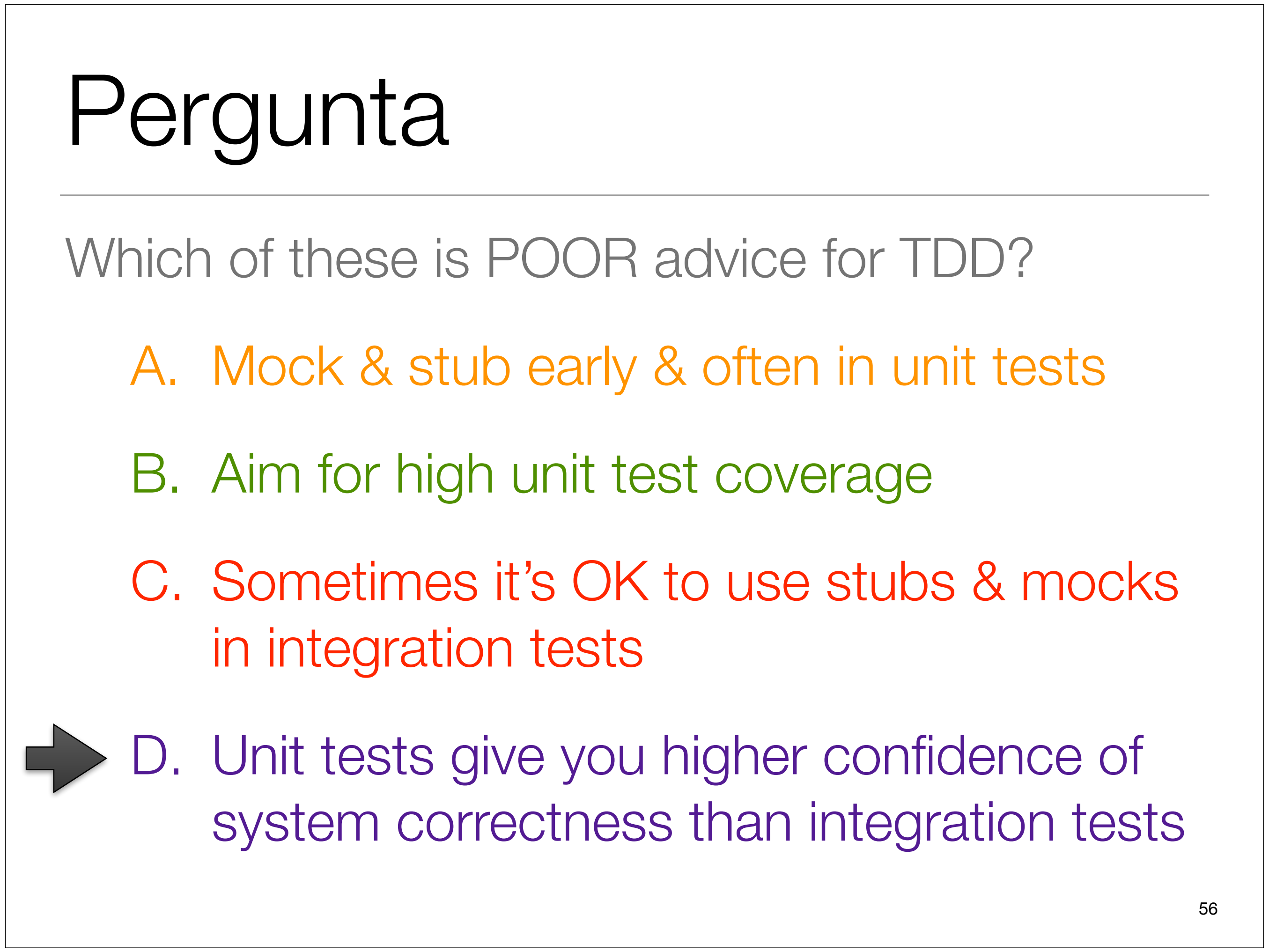
A. Mock & stub early & often in unit tests

B. Aim for high unit test coverage

C. Sometimes it’s OK to use stubs & mocks in integration tests

D. Unit tests give you higher confidence of system correctness than integration tests

purple is the answer



Pergunta

Which of these is POOR advice for TDD?

A. Mock & stub early & often in unit tests

B. Aim for high unit test coverage

C. Sometimes it’s OK to use stubs & mocks in integration tests

➡ D. Unit tests give you higher confidence of system correctness than integration tests

purple is the answer

Other Testing Concepts

Testing vs. Debugging

Other testing terms you may hear

- Mutation testing: if introduce deliberate error in code, does some test break?
- Fuzz testing: 10,000 monkeys throw random input at your code
  - Find ~20% MS bugs, crash ~25% Unix utilities
  - Tests app the way it wasn’t meant to be used
- DU-coverage: is every pair <define x/use x> executed?
- Black-box vs. white-box/glass-box

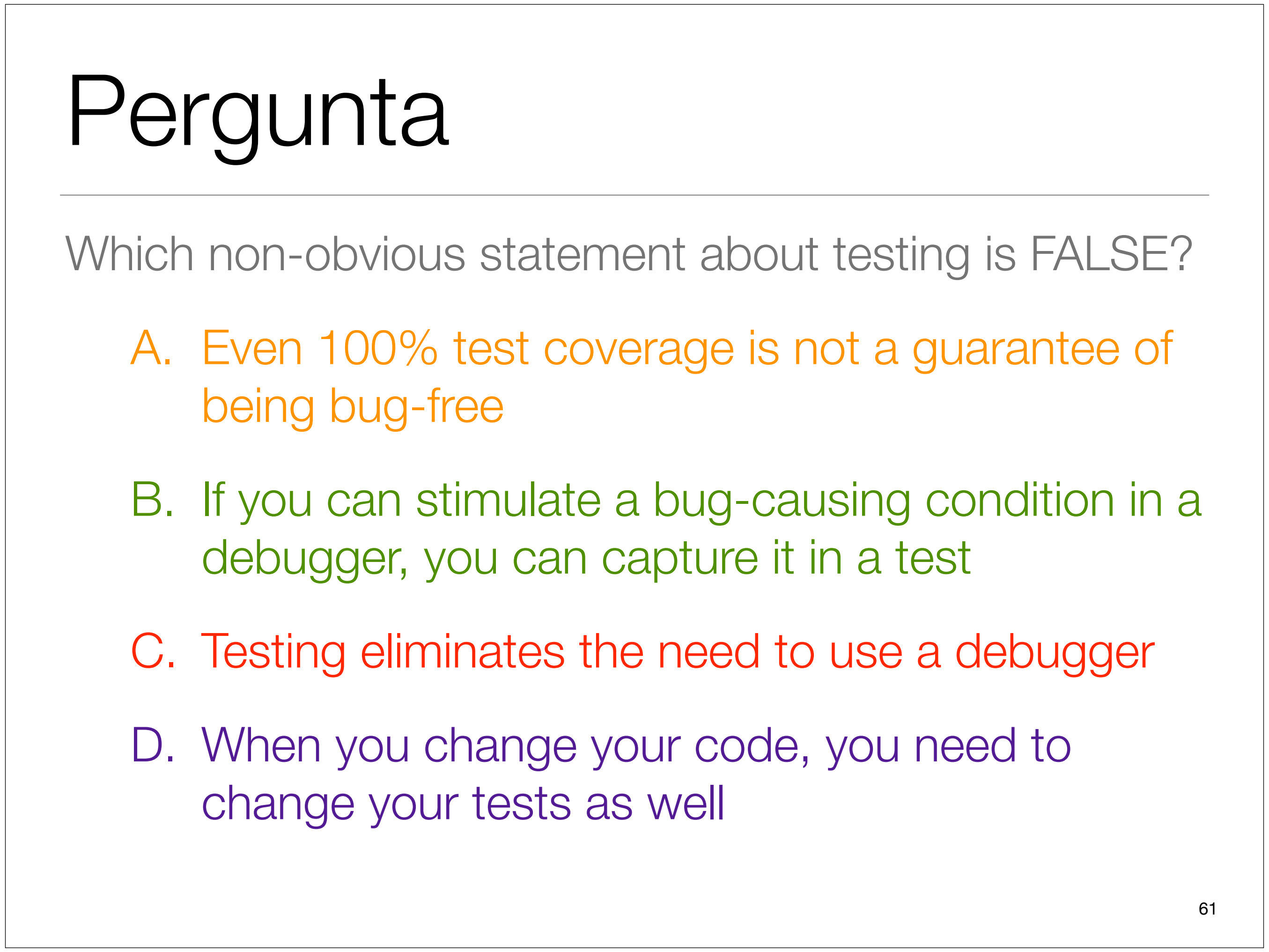
TDD vs. Conventional debugging

Conventional	TDD
Write 10s of lines, run, hit bug: break out debugger	
Insert printf's to print variables while running repeatedly	
Stop in debugger, tweak/set variables to control code path	
Dammit, I thought for sure I fixed it, now I have to do this all again	

- Lesson 1: TDD uses same skills & techniques as conventional debugging - but more productive (FIRST)
- Lesson 2: writing tests before code takes more time up-front, but often less time overall

TDD Summary

- **Red** – **Green** – **Refactor**, and always have working code
- Test one behavior at a time, using seams
- Use `it` “placeholders” or `pending` to note tests you know you’ll need
- Read & understand coverage reports
- “Defense in depth”: don’t rely too heavily on any one kind of test



Pergunta

Which non-obvious statement about testing is FALSE?

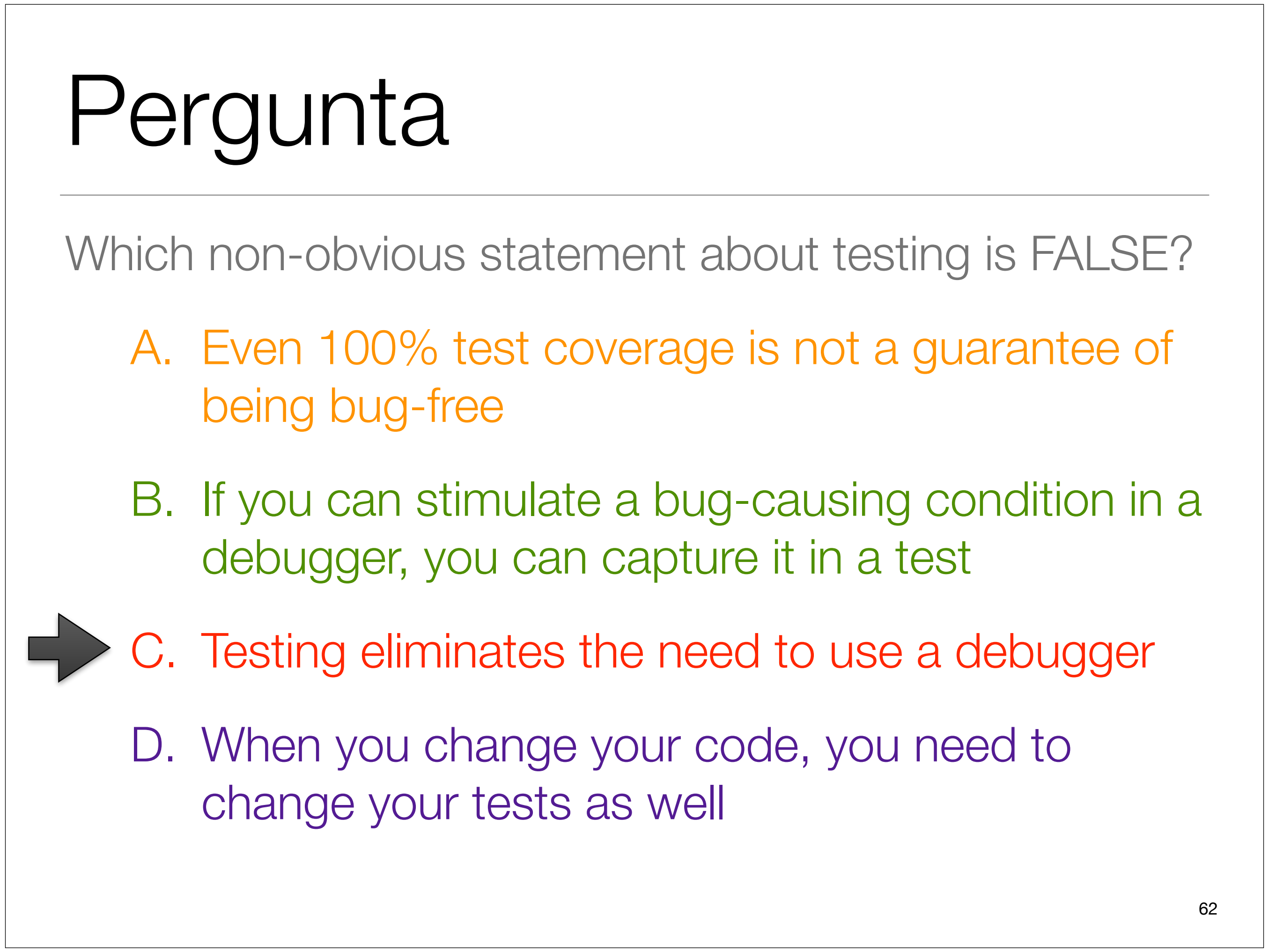
A. Even 100% test coverage is not a guarantee of being bug-free

B. If you can stimulate a bug-causing condition in a debugger, you can capture it in a test

C. Testing eliminates the need to use a debugger

D. When you change your code, you need to change your tests as well

red is the answer



Pergunta

Which non-obvious statement about testing is FALSE?

A. Even 100% test coverage is not a guarantee of being bug-free

B. If you can stimulate a bug-causing condition in a debugger, you can capture it in a test

➡ C. Testing eliminates the need to use a debugger

D. When you change your code, you need to change your tests as well

red is the answer