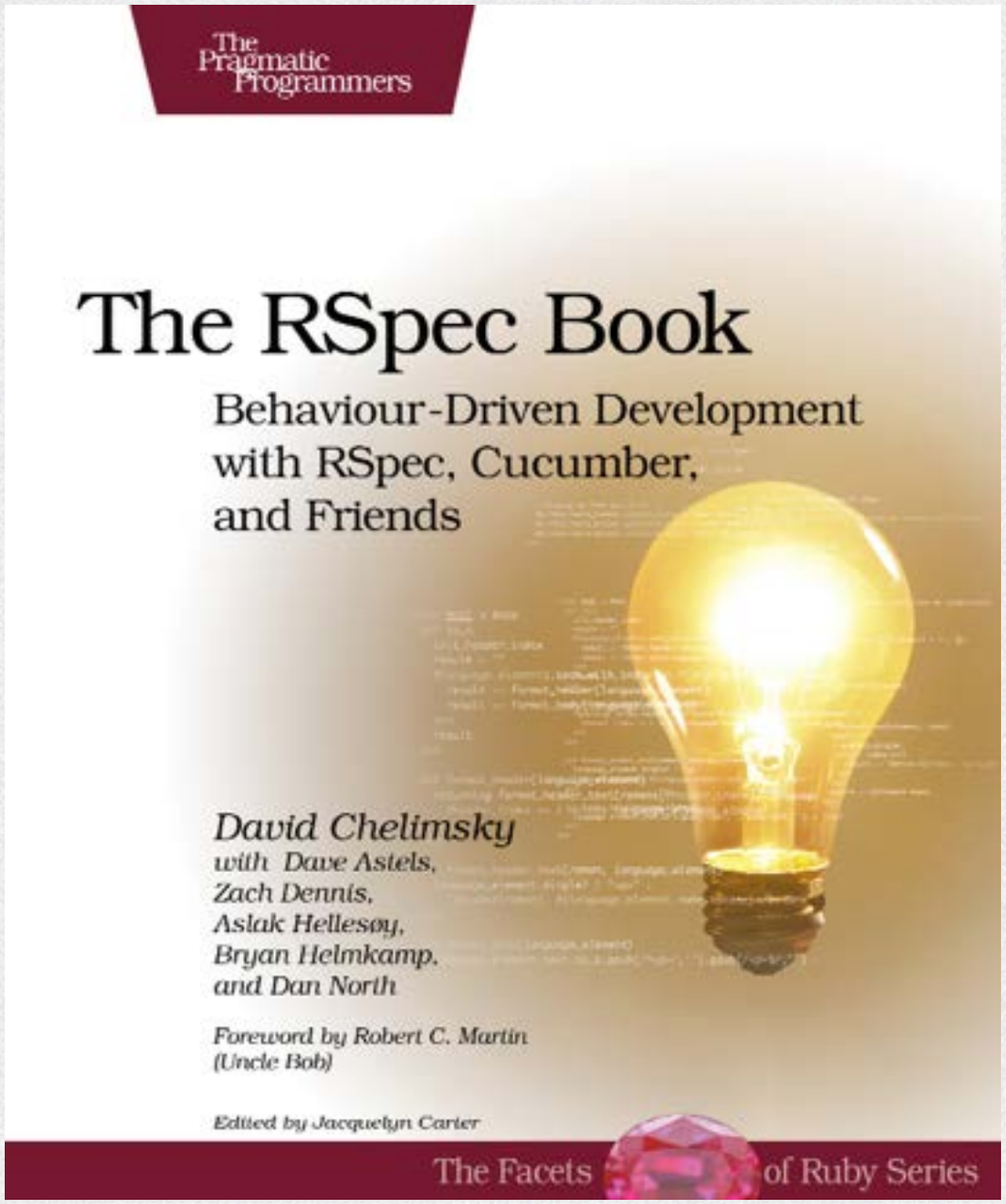
Práticas de Desenvolvimento de Software

#

Aula 07

Introdução a TDD, BDD e Dojo

Bibliografia



The RSpec Book

Behaviour-Driven Development with RSpec, Cucumber, and Friends

by **David Chelimsky**, with Dave Astels, Zach Dennis, Aslak Hellesøy, Bryan Helmkamp and Dan North

Bibliografia

Cucumber e RSpec

Construa aplicações Ruby com testes e especificações utilizando TDD e BDD.

por **Hugo Baraúna**

O processo de testar

Muito antigamente (sem processo)

- Debug vs Testing
- Não existiam testes automatizados

O processo de testar

Antigo, mas nem tanto (waterfall/em cascata)

- Quem testa é um grupo independente (QA)
- O teste ocorre entre o fim do desenvolvimento e o lançamento do produto…
- ou em paralelo com o desenvolvimento de forma contínua, sempre que um módulo é concluído

O processo de testar

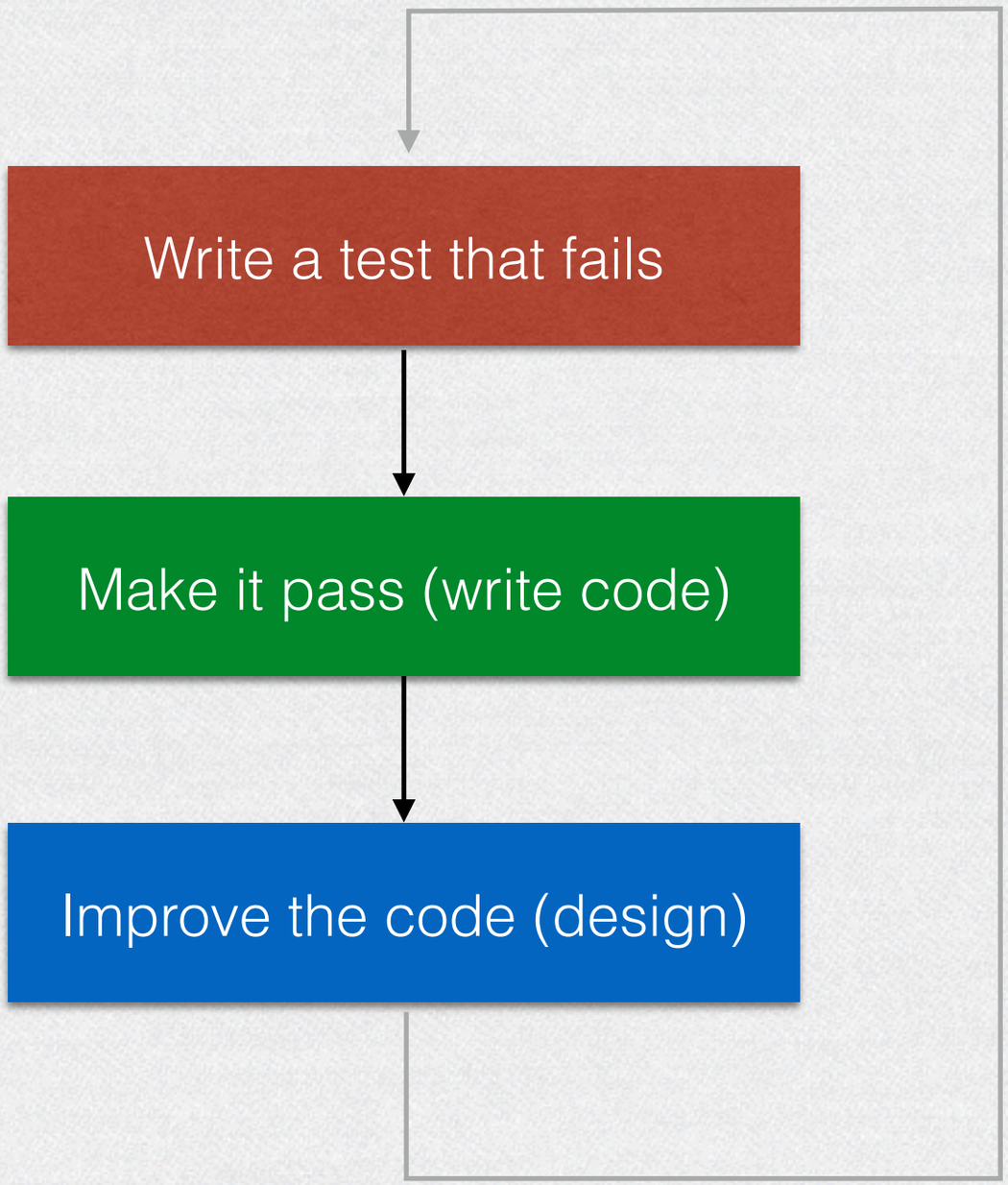
Modelo atual (ágil)

- Quem cria o teste é o próprio desenvolvedor
- Teste são feitos antes do desenvolvimento de cada módulo
- Os testes evoluem em paralelo com o produto
- Pair programming permite a revisão do código de forma instantânea por um colega da equipe

TDD (Test Driven Development)

- TDD é a prática de escrever testes antes de escrever o código sendo testado
- O teste é escrito para um código que ainda não existe, i.e, você escreve testes para o código que gostaria de ter
- O teste escrito é utilizado como especificação do código

TDD (Test Driven Development)



Red

Green

Refactor

```
# Code
registrations = []
def register
  new_registration = Registration.new
  registrations.push(new_registration)
  new_registration
end
```

```
# Tests
def test_that_registration_is_an_array
  raise "Registrations should be an array" if !registrations.is_a?(Array)
end

def test_register_creates_one_registration
  initial_count = registrations.size
  register()
  new_registrations = registrations.size - initial_count
  raise "Should have created one registration" if new_registrations != 1
end
```

BDD (Behaviour Driven Development)

- É uma evolução de TDD com o objetivo de colocar o foco no cliente e em suas necessidades ao invés do código
- O teste escrito é utilizado como especificação de funcionalidades (e *user stories*) do produto
- Testes de integração e de aceitação são criados para garantir que o cliente está recebendo o que pediu

Não existe um consenso na comunidade sobre as diferenças de TDD e BDD. Esta é a forma que decidimos apresentar no curso.

Discussão:
Quais são as vantagens de testes?

Discussão:
Quais são as vantagens de testes?

- Especificação
- Feedback
- Regressão
- Granularidade

Discussão:
Quais são as desvantagens de testes?

Discussão:
Quais são as desvantagens de testes?

- Custo
- Desenvolvedores precisam ser mais qualificados
- Não existe uniformidade de processos e ferramentas*

*Eu ainda não encontrei, mas se um dia você encontrar, aqui vai o meu email: rafael.barbolo@infosimples.com.br

TDD vs BDD

```
#
# Considering the language, BDD is more humanised
#

# TDD
assert_equals(count, 5)

# BDD
count.should_be 5
```

TDD vs BDD

Como seria o teste da ordenação de um array?

- Existem diversos algoritmos de ordenação
- Em TDD, cada algoritmo teria um teste diferente, pois o teste verificaria detalhes de estrutura e implementação
- Em BDD, o teste seria o mesmo para todos, pois o que importa é o comportamento final

Ferramentas

RSpec

- Comumente, usada para testes unitários

Cucumber

- Comumente, usada para testes de aceitação/integração

Há quem diga que RSpec está para TDD assim como Cucumber para BDD.

RSpec

```ruby
# 1. Write test: bowling_spec.rb
require './bowling'
describe Bowling, "#score" do
  it "returns 0 for all gutter game" do
    bowling = Bowling.new
    20.times { bowling.hit(0) }
    bowling.score.should eq(0)
  end
end

# 2. See testing failing: rspec bowling_spec.rb

# 3. Write bowling.rb
class Bowling
  def hit(pins)
  end
  def score
    0
  end
end

# 4. See tests passing: rspec bowling_spec.rb
```

Cucumber

```
# This is how a small acceptance test looks like

Feature: In order to let customers organise their information across pages
         As an administrator of a micro-site
         I want to be able to add subpages

Scenario: Adding a subpage
  Given I am logged in
  When I press "Add subpage"
  And I fill in "Title" with "Gallery"
  And I press "Ok"
  Then I should see a document called "Gallery"
```

Cucumber

```
# Administrators should be able to create pages
```

RSpec - describe, context, it

```ruby
#
# Basic structure of a test
#
# What we want to test is:
# "Describe an order. It sums the prices of its items."
describe Order do
  it "sums the prices of its items" do
    order = Order.new
    order.add_item(Item.new(price: 1.11))
    order.add_item(Item.new(price: 2.22))
    expect(order.total).to 3.33
  end
end
```

RSpec - describe, context, it

```ruby
#
# Structuring a test with nestes groups.
#
describe Order do
  context "with no items" do
    it "behaves one way" do
      # ...
    end
  end

  context "with one item" do
    it "behaves another way" do
      # ...
    end
  end
end
```

RSpec - expectations

```
# Equivalence
expect(actual).to eq(expected)  # passes if actual == expected
expect(actual).to eql(expected) # passes if actual.eql?(expected)

# Identity
expect(actual).to be(expected)    # passes if actual.equal?(expected)
expect(actual).to equal(expected) # passes if actual.equal?(expected)

# Comparisons
expect(actual).to be >  expected
expect(actual).to be >= expected
expect(actual).to be <= expected
expect(actual).to be <  expected
expect(actual).to be_within(delta).of(expected)

# Regular expressions
expect(actual).to match(/expression/)
```

RSpec - expectations

```ruby
# Types/classes
expect(actual).to be_an_instance_of(expected)
expect(actual).to be_a_kind_of(expected)

# Truthiness
expect(actual).to be_true  # passes if actual is truthy (not nil or false)
expect(actual).to be_false # passes if actual is falsy (nil or false)
expect(actual).to be_nil   # passes if actual is nil

# Collection membership
expect(actual).to include(expected)
expect(actual).to start_with(expected)
expect(actual).to end_with(expected)
```

RSpec - one convention

```ruby
# Class/instance methods convention

class User
  # class methods
  def self.authenticate(username, password)
    User.where(username: username, password: password_hash(password)).first
  end
  # instance methods
  def admin?
    self.groups.include?(:admin)
  end
end

# BAD
describe 'the authenticate method for User' do
describe 'if the user is an admin' do

# GOOD
describe '.authenticate' do
describe '#admin?' do
```

Dojo

Dojo é o lugar onde você vai toda semana para aprender e treinar Karatê.

No nosso caso, para aprender e treinar **programação**.

RandoriKata

Um movimento (Kata) será aprendido hoje com a participação de todos.

Regras:

- Deve ser utilizado TDD (teste primeiro, codifique depois)
- Programação em par (piloto e copiloto)
- Turnos de 5 minutos de programação para cada piloto
- Passos de bebê: vamos implementar pequenas mudanças de cada vez
- Comentários da plateia somente se testes estiverem passando **(green)**
- Piloto e copiloto podem conversar a qualquer momento
- Sensei (professor/monitor) pode interromper a qualquer momento

Kata: Roman Numerals

Given a positive integer number (eg 42) determine its Roman numeral representation as a String (eg "XLII").

You cannot write numerals like IM for 999. Wikipedia states "Modern Roman numerals are written by expressing each digit separately starting with the leftmost digit and skipping any digit with a value of zero."

Examples:

```
1 ->    "I" | 10 ->    "X" | 100 ->    "C" | 1000 ->    "M"
2 ->   "II" | 20 ->   "XX" | 200 ->   "CC" | 2000 ->   "MM"
3 ->  "III" | 30 ->  "XXX" | 300 ->  "CCC" | 3000 ->  "MMM"
4 ->   "IV" | 40 ->   "XL" | 400 ->   "CD" | 4000 -> "MMMM"
5 ->    "V" | 50 ->    "L" | 500 ->    "D" |
6 ->   "VI" | 60 ->   "LX" | 600 ->   "DC" |
7 ->  "VII" | 70 ->  "LXX" | 700 ->  "DCC" |
8 -> "VIII" | 80 -> "LXXX" | 800 -> "DCCC" |
9 ->   "IX" | 90 ->   "XC" | 900 ->   "CM" |

1990 -> "MCMXC"  (1000 -> "M"  + 900 -> "CM" + 90 -> "XC")
2008 -> "MMVIII" (2000 -> "MM" + 8 -> "VIII")
  99 -> "XCIX"   (90 -> "XC" + 9 -> "IX")
  47 -> "XLVII"  (40 -> "XL" + 7 -> "VII")
```

Acompanhamento do Dojo

https://www.dropbox.com/sh/5tz1wn9uwz3e6lk/Kbn9fi8Fdw

Recursos extras

- BDD vs TDD (explained) - https://www.youtube.com/watch?v=mT8QDNNhExg
- Why Bother With Cucumber Testing? - http://www.jackkinsella.ie/2011/09/26/why-bother-with-cucumber-testing.html
- RSpec - http://rspec.info/
- Better Specs (RSpec Guidelines with Ruby) - http://betterspecs.org/
- Cucumber - http://cukes.info/
- Coding Dojo - http://codingdojo.org/
- Cyber Dojo - http://cyber-dojo.org/
- How do you put on a coding dojo event? - https://www.youtube.com/watch?v=gav9fLVkZQc