

# [IF977] Engenharia de Software

---

Prof. Vinicius Cardoso Garcia  
[vcg@cin.ufpe.br](mailto:vcg@cin.ufpe.br) :: [@vinicius3w](https://twitter.com/vinicius3w) :: [assertlab.com](http://assertlab.com)



## Licença do material

---

Este Trabalho foi licenciado com uma Licença

**Creative Commons - Atribuição-NãoComercial-  
Compartilhualgual 3.0 Não Adaptada.**

Mais informações visite

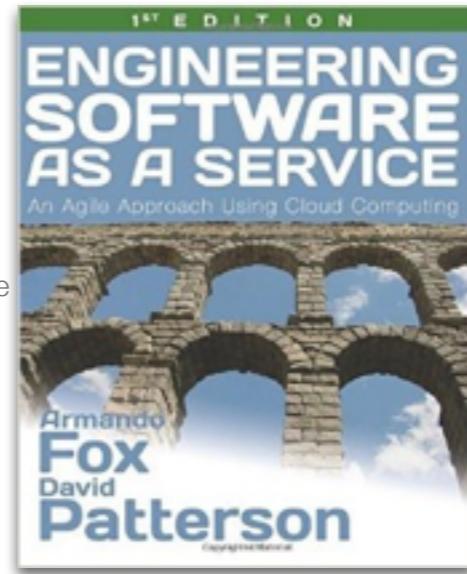
<http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt>



## Referências

---

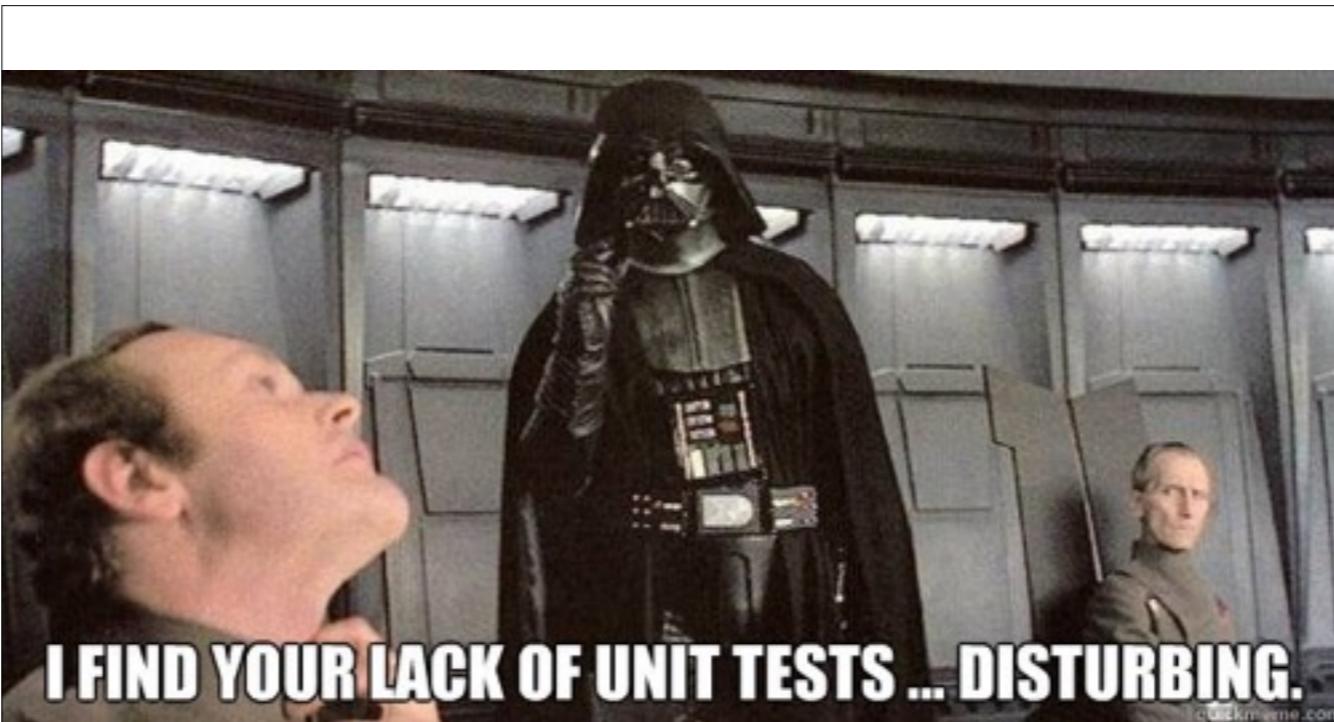
- A biblioteca do Desenvolvedor de Software dos dias de hoje
  - <http://bit.ly/TDOA5L>
- SWEBOK
  - Guide to the Software Engineering Body of Knowledge (SWEBOK): <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing (Beta Edition)
  - <http://www.saasbook.info/>



# Outline

---

- (ESaaS 8.1) Intro to RSpec & Unit Tests
- (ESaaS 8.2) FIRST, TDD, and Getting Started With RSpec
- (ESaaS 2.1-2.2) The Web as a Client-Server System; TCP/IP intro
- (ESaaS 2.3) HTML + CSS
- (ESaaS 2.4) 3-tier shared-nothing architecture & scaling



clickkarma.com

## Introdução ao RSpec & Testes Unitários



5

(Engineering Software as a Service §8.1)



Turing Award winner Brian Kernighan, co-inventor of the C language  
Turing Award winner Edsger Dijkstra, "father of object-oriented programming"  
Seen in bathrooms at Google Inc. as part of "Testing on the Toilet" program

## **Survey Finds 58% of Software Bugs Result from Test Infrastructure and Process, Not Design Defects**

### *Developers Prefer Taxes to Dealing with Software Testing*

**Sunnyvale, Calif. — June 2, 2010** Electric Cloud®, the leading provider of software production management (SPM) solutions, today released the results of a survey conducted in partnership with Osterman Research showing that the majority of software bugs are attributed to poor testing procedures or infrastructure limitations rather than design problems. Additionally, the software test process is generally considered an unpleasant process, with software development professionals rating the use of their companies' test systems more painful than preparing taxes.

Fifty-eight percent of respondents pointed to problems in the testing process or infrastructure as the cause of their last major bug found in delivered or deployed software, not design defects.

Specifically, the survey found:

- ✓ Completely automated software testing environments are still rare, with just 12 percent of software development organizations using fully automated test systems. Almost 10 percent reported that all testing was done manually.

## Testes hoje em dia

---

- Antes
  - Desenvolvedores finalizam o código, alguns testes ad-hoc
  - “jogar pro alto a Garantia de Qualidade [QA]”
  - Equipe de QA manualmente exercita o software
- Hoje em dia / Ágil
  - Testes é parte de **TODA** iteração Ágil
  - Desenvolvedores **testam seu próprio** código
  - Ferramentas e processos de testes **automatizados**
  - Equipe de testes/QA melhora as **ferramentas/testabilidade**

## Testes hoje em dia

---

- Antes
  - Desenvolvedores finalizam o código, alguns testes ad-hoc
  - “jogando no escuro”
- Em vez de
- Hoje
  - Testes integrados
  - Desenvolvedores realizam testes
- Ferramentas e processos de testes **automatizados**
- Equipe de testes/QA melhora as **ferramentas/testabilidade**

Qualidade de Software é o resultado de um bom **processo**, ao invés de responsabilidade de um grupo específico

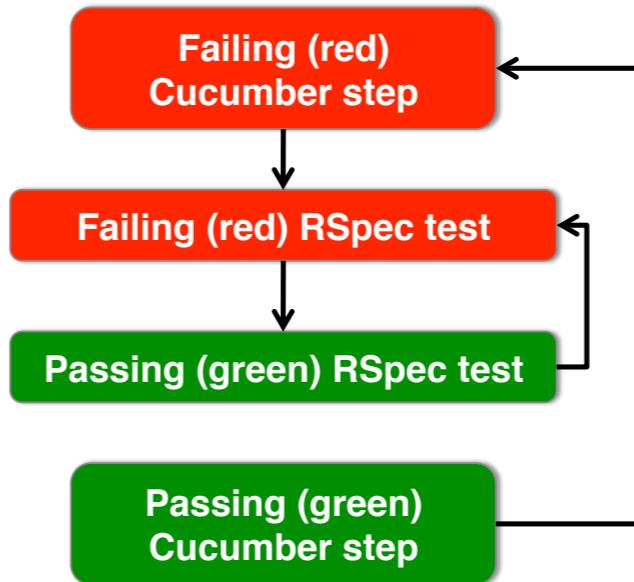
## BDD+TDD: Visão Geral

---

- Behavior-driven design (BDD)
  - Desenvolver histórias do usuário [HU] (**as características desejadas**) para descrever como a aplicação deve funcionar
    - via **Cucumber**, HU se tornam testes de aceitação e integração
  - Test-driven development (TDD)
    - **Definir** uma nova HU pode requerer a escrita de novo código
    - TDD diz: escreva testes unitários e funcionais primeiro, **antes** do código propriamente dito
    - Isto é: escreva o teste do código que você deseja ter

## Cucumber & RSpec

- Cucumber descreve o comportamento via features & cenários (BDD)
- RSpec testa módulos individuais que contribuem para estes comportamentos (TDD)



11

1st step is thinking of “code you wish you had”, assuming methods that if existed would make it a perfect match to the user story.

Rather than inside – out (start with building blocks and compose to provide desired functionality), go from users outside-in, to reduce wasted coding; e.g., until get to user view, won’t know what you really need. Especially Web apps, since easy to see what the user is doing.

Both cycles involve taking small steps and listening to the feedback you get from the tools. We start with a failing step (red) in Cucumber (the outer cycle). To get that step to pass, we’ll drop down to RSpec (the inner cycle) and drive out the underlying code at a granular level (red/green/refactor).

At each green point in the RSpec cycle, we’ll check the Cucumber cycle.

If it is still red, the resulting feedback should guide us to the next action in the RSpec cycle.

If it is green, we can jump out to Cucumber, refactor

if appropriate, and then repeat the cycle by writing a new failing Cucumber step.

## Pergunta

---

O Que é verdade sobre BDD & TDD?

1. Requisitos guiam a implementação
2. Eles podem ser utilizados somente no desenvolvimento Ágil
3. Eles suportam e lidam com mudanças
  - A. Somente (1)
  - B. Somente (1) & (2)
  - C. Somente (1) & (3)
  - D. (1), (2) e (3)

## Pergunta

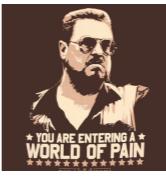
---

O Que é verdade sobre BDD & TDD?

1. Requisitos guiam a implementação
2. Eles podem ser utilizados somente no desenvolvimento Ágil
3. Eles suportam e lidam com mudanças
  - A. Somente (1)
  - B. Somente (1) & (2)
  - C. Somente (1) & (3)**
  - D. (1), (2) e (3)



## Administrivia



- Se você não está envolvido na disciplina até agora, você está em sérios apuros...
- Se você pensa em pular fora e ainda não o fez
- Wise up: verifique o site e os demais canais de apoio a disciplina
- Já está inserido em algum projeto? Deadline: **próxima segunda!**
- Leia as mensagens de erro antes de se desesperar!
- Sempre busque informações com os monitores sobre como melhorar e/ou executar as atividades



## FIRST, TDD, Introdução ao RSPEC

(Engineering Software as a Service §8.2)

## A propriedade FIRST

---

- **F**ast
- **I**ndependent
- **R**epeatable
- **S**elf-checking
- **T**imely

## A propriedade FIRST

---

- **Fast**: executar (um conjunto de) testes rapidamente
- **Independent**: Nenhum teste depende de outro, então podemos executá-los em qualquer ordem
- **Repeatable**: Execute N vezes e obtenha o mesmo resultado (isolamento de erros e automação)
- **Self-checking**: Podem checar automaticamente se passaram (sem intervenção humana)
- **Timely**: Escritos ao mesmo tempo que o código testado (com TDD, escreva-o primeiro)

## Rspec, uma linguagem de domínio específico para testes

---

- DSL: uma “pequena” linguagem que simplifica uma tarefa em detrimento da generalidade
  - Migrações, SQL, expressões regulares
- Testes Rspec são chamados **specs** ou **exemplos**  
<http://pastebin.com/LTK36Pb>
- Executa os testes em um arquivo: **rspec filename**
  - Vermelho falhou, Verde passou, Amarelo pendente
- Muito melhor: executando **autotest**

18

Opinions vary on whether to test the views

Our position: views are user-facing, so use user stories to test => Cucumber

## Pergunta

---

Que tipos de código podem ser testados Repetidamente e Independentemente?

1. Código que depende da aleatoriedade (por exemplo embaralhar um baralho de cartas)
2. Código que depende da hora do dia (por exemplo, os backups executados todos os domingos à meia-noite)
  - A. Somente (1)
  - B. Somente (2)
  - C. Os dois
  - D. Nenhum dos dois

## Pergunta

---

Que tipos de código podem ser testados Repetidamente e Independentemente?

1. Código que depende da aleatoriedade (por exemplo embaralhar um baralho de cartas)
2. Código que depende da hora do dia (por exemplo, os backups executados todos os domingos à meia-noite)
  - A. Somente (1)
  - B. Somente (2)
  -  C. Os dois
  - D. Nenhum dos dois

## TDD & Testing

---

- “TDD is hard at first, but it gets easier”
- “Was great for quickly noticing bugs [regression] and made them easy to fix”
- “Helped me organize my thoughts as well as my code” [The code you wish you had]
- “Wish we had committed to it earlier & more aggressively”
- “We didn’t always test before pushing, and it caused a lot of pain”