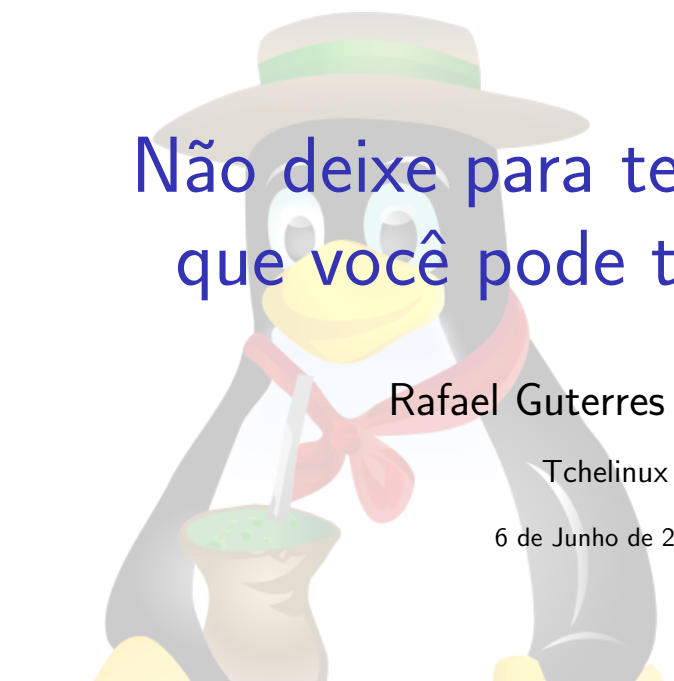


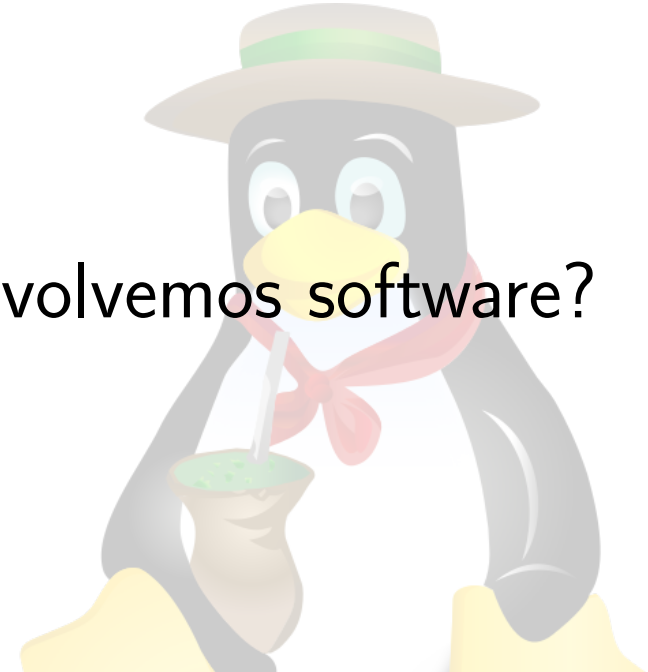
Não deixe para testar depois o que você pode testar antes.

Rafael Guterres Jeffman

Tchelinux

6 de Junho de 2018

Por que desenvolvemos software?



Como desenvolvemos software?

Ciclo do Desenvolvimento de Software

- Análise
- Projeto
- Implementação
- Testes
- Implantação

User Stories

- Descrição informal de uma funcionalidade.
- Escritas na perspectiva do usuário.
- As histórias devem dizer **quem** é o usuário, **o que** ele vai fazer, e qual o **objetivo** ele quer atingir.

Critérios de Aceitação

Para determinar se uma história de usuário está completa, deve haver um **critério de aceitação**, com objetivos que podem ser medidos e avaliados.

Test Driven Development

- Faz com que todo código escrito seja testado.
- Auxilia no design da aplicação.
- Inverte a ordem tradicional de testes de software.

Cria teste $\rightarrow$ Teste falha $\rightarrow$

Implementa código $\rightarrow$ Teste passa $\rightarrow$

Refatorar código

Críticas ao TDD

Definição do que é uma **unidade**

Quantidade de código de teste.

Uso de **mocks**.

Dificuldade de testar algumas situações, com bancos de dados e configurações específicas de plataforma.

Behavior Driven Development

- A unidade testada, é um requisito.
- Testa um comportamento esperado do sistema.
- Utiliza uma linguagem específica de domínio.
- Faz uso extensivo de ferramentas.

Gherkin DSL

- Desenvolvida para definir casos de teste no Cucumber.
- Procura forçar requisitos e resultados claros.
- Facilmente traduz os critérios de aceitação de histórias.
- Internacionalizável.

Exemplo

Feature: I want to code.

As a Developer, I want to code, so that I can pay my bills.

Scenario: I have to eat.

Given I know how to code.

Then I'm hired to code.

Then I'm payed in pizza.

Behave

- Behave é uma ferramenta que auxilia no uso de BDD com Python.
- Auxilia na automação de testes de aceitação (e unitários).
- Facil de integrar ao projeto.
- Busca facilitar a criação do código de teste.

Features

- Features são escritas pelos Stakeholders, Analistas, etc.
- Uma feature terá, provavelmente, vários cenários.
- Podemos utilizar uma História de Usuário como uma feature, e os cenários como os testes de aceitação dessa feature.
- Descritas em linguagem natural, baseado no formato Gherkin.

Cenários

- Descrevem os passos para validar um cenário de uma feature.
- Cada cenário descreve apenas uma característica da feature.
- Cenários são criados a partir do ponto de vista do usuário.
- Podemos ter cenários para situações de erro.

Given, When, Then

Cenários são descritos em passos que, se executados corretamente, mostram que o cenário está corretamente implementado.

Given Utilizado para colocar o sistema em um estado conhecido.

When Ações são executadas, modificando, ou não, o estado do sistema.

Then Saídas observadas e avaliadas.

Um cenário completo.

Scenario: Change data of an inexistent institution.
Given the database has data for two institutions
When changing the data of an institution with id "inexistent" to

```
"""
{ "long_name": "A Nice Place to Be" }
"""
```

Then the status code is 404

Testando com o Behave.

- As **features** devem ficar em um diretório *features*
- As implementações dos **passos** devem ficar em um subdiretório **steps**, dentro de *features*.
- Você pode controlar diversos aspectos dos testes através de parâmetros da linha de comando.

```
You can implement step definitions for undefined steps
with these snippets:

@given(u'the database has data for two institutions')
def step_impl(context):
    raise NotImplementedError(u'STEP: Given the ...')

@when(u'changing the data of an institution...')
def step_impl(context):
    raise NotImplementedError(u'STEP: When changing...')

@then(u'the status code is 404')
def step_impl(context):
    raise NotImplementedError(u'STEP: Then ...')
```

Implementando os Testes: *Given*

```
@given('the database has data for two institutions')
def given_two_institutions_in_two_cities(context):
    """Add two institutions to the database."""
    url = "http://127.0.0.1:5000/api/city"
    for v in context.cidades.values():
        response = requests.post(url, json=v)
        assert response.status_code == 201
    url = "http://127.0.0.1:5000/api/institution"
    for institution in context.institutions:
        response = requests.post(url, json=institution)
        assert response.status_code == 201
```

Implementando os Testes: *When*

```
@when('changing the data of an institution "{inst_id}"')
def when_changing_institution_data(context, inst_id):
    """Request changes to the institution data."""
    context.institution_id = institution_id
    context.input = json.loads(context.text)
    url = "http://127.0.0.1:5000/api/institution/{}"
    url = url.format(institution_id)
    context.response = requests.put(url,
                                    json=context.input)
```

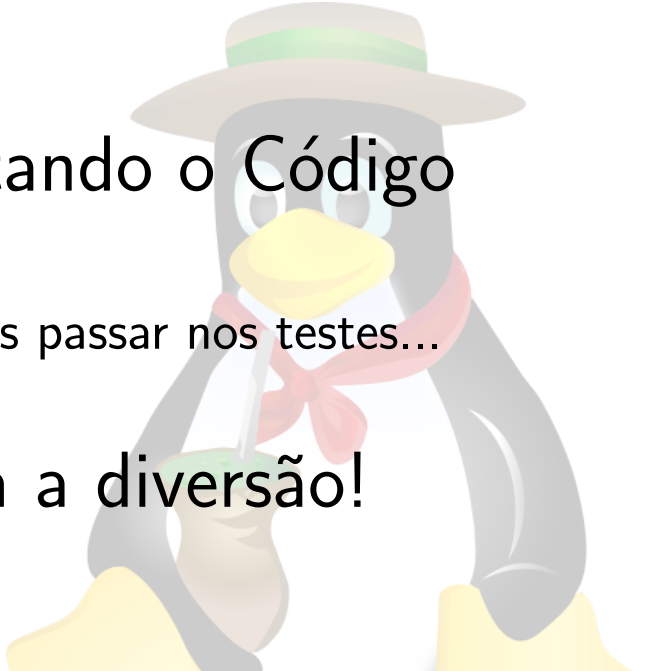
Implementando os Testes: *Then*

```
@then('the status code is {status_code}')
def then_verify_status_code(context, status_code):
    """Test the context response status code."""
    expected = int(status_code)
    observed = context.response.status_code
    assert expected == observed
```

Implementando o Código

Agora devemos passar nos testes...

Começa a diversão!



Para onde ir agora?

- http://github.com/behave
- http://behave.readthedocs.io/en/latest/tutorial.html
- http://behave.readthedocs.io/en/latest

Obrigado!

http://rafaeljeffman.com?tchelinux

rafasgj@gmail.com