Software development

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Testing: implementation, maintenance and execution

Acceptance tests

Directly connecting requirements (scenarios) and tests

Exercising the system through its GUI

GUI based scenario

Scenario: Registering student with non registered CPF

Given I am at the students page
Given I cannot see a student with CPF "683" in the students list
When I try to register the student "Paulo" with CPF "683"
Then I can see "Paulo" with CPF "683" in the students list

GUI based test step

```
Given(/^I am at the students page$/, async () => {

  await browser.get("http://localhost:4200/");
  await expect(browser.getTitle()).to
              .eventually.equal('TaGui');
  await $(“a[name='alunos']").click();

})
```

Steps exercise the system under test (SUT) by simulating user actions in the browser

Application as a state machine

- Graph representing states and possible transitions
- Behavior desired by the user should appear as paths in this graph (traces), undesired behavior should not appear
- Scenarios should verify that desired traces can be observed by testing the application

Exercising the system through its server

Service based scenario

Scenario: Registering student with non registered CPF, service

Given the system has no student with CPF "683"
When I register the student "Paulo" with CPF "683"
Then the system now stores "Paulo" with CPF "683"

Service based test step

```
Given(/^the system has no student with CPF “(.*)”$/,
        async (cpf) => {

    await request.get(base_url + "alunos").then(body =>
        expect(body).toBe("[]")).catch(e =>
        expect(e).toEqual(null));

})
```

Steps exercise the system under test (SUT) by invoking services

Service tests (unit, integration, system)

Service test, no link with scenarios

```
describe("O servidor", () => {

  it("inicialmente retorna uma lista de alunos vazia", () => {
    return request.get(base_url + "alunos").then(body =>
      expect(body).toBe("[]")).catch(e =>
      expect(e).toEqual(null));
  })
```

Test suites

```
it("só cadastra alunos", () => {
    var options = {method: 'POST', uri: (base_url + "aluno"),
                   body: {name: "Mari", cpf: "962"},
                   json: true};
    return request(options)
        .then(body =>
            expect(body).toEqual({failure: "O aluno não
                                  pode ser cadastrado”}))
        .catch(e => expect(e).toEqual(null))
});
```

Common actions and state for tests in a suite

```
var server:any;

beforeAll(() => {
    server = require('../ta-server')
});

afterAll(() => {
    server.closeServer()
});
```

Code exercises the system under test (SUT) by invoking services

Class tests (unit, integration)

Class test, no link with scenarios

```
describe("O cadastro de alunos", () => {
  var cadastro: CadastroDeAlunos;

  beforeEach(() => cadastro = new CadastroDeAlunos())

  it("é inicialmente vazio", () => {
    expect(cadastro.getAlunos().length).toBe(0);
  })
```

```
it("não aceita alunos com CPF duplicado", () => {
    var aluno: Aluno = new Aluno();
    aluno.nome = "Mariana";
    aluno.cpf = "683";
    cadastro.criar(aluno);

    aluno = new Aluno();
    aluno.nome = "Pedro";
    aluno.cpf = "683";
    cadastro.criar(aluno);

    expect(cadastro.getAlunos().length).toBe(1);
})
```

Code exercises the system under test (SUT) by invoking methods

Principles for all tests

- Run test with different arguments and state content
  - boundary cases, focus on coverage
- Positive and negative cases
- Regression
- Balance between
  - service, class and acceptance tests
  - unit and integration tests

Practices for all tests

- Parametrize tests
- Each test can run independently of the others, assuming a fresh application instance
- Passing state between steps
- Before pushing (sometimes committing), make sure all tests pass
  - Regression testing

Behavior driven design: test implementation before feature implementation

Create interface when little functionality is available

```
static public void createArticle(String title, filename) {
    def cont = new PeriodicoController()
    cont.params << TestData.findArticleByTitle(title)
                << [file: filename]
    cont.request.setContent(new byte[1000])
    cont.create()
    cont.save()
    cont.response.reset()
}
```

functionality interface

```
class PeriodicoController {
    create() {}
    save() {}
}
```

Checklist

Test behavior should strictly conform to scenario semantics

```
When(~'^I select to view "([^"]*)" in resulting list$') {
    String title ->
    at ArticlesPage
    page.selectViewArticle(title)
    at ArticleShowPage
}
```

third command is
not implied by the scenario
step semantics; the first is also
not implied, but is a
precondition

Avoid ambiguities due to similar step sentences

```
Then(~'^I can fill the article details$') {->
    at ArticleCreatePage
    page.fillArticleDetails()
}

Then(~'^I can fill the tool details$') { ->
    at FerramentaCreatePage
    page.fillToolDetails()
}
```

Do not duplicate test code

```
def fillLoginDataOnly(...) {
        $("form").username = username
        $("form").password = password
 }
 def fillLoginDataAndSubmit(...) {
        $("form").username = username
        $("form").password = password
        $("form").signIn().click()
 }
```

```
class ResearchGroupPage extends Page {
    static url = "researchGroup/list" ...
}
class ResearchGroupListPage extends Page {
    static url = "researchGroup/list" ...
}
```

Tests should clean up environment at the end

```
After() {
  ...
  def uploadsFolder = new File(...)
  uploadsFolder.listFiles().each {
    innerFile ->
    innerFile.deleteOnExit()
  }
}
```

Tests should be plataform (including browser) and language independent

```
class ArticleCreatePage extends Page {
  static url = "article/create"

  static at = {
    def gp = new GetPageTitle()
    def a = gp.msg("...article.label")
    def t = gp.msg("...create.label", [a])

    title ==~ t
  ...
```

Testing research at CIn

- Test generation and static analysis tools: Marcelo e Paulo
- Model-based testing: Alexandre Mota, Juliano e Augusto
- Test selection and execution: Juliano

To do after class

- Answer questionnaire (check classroom assignment), study correct answers
- Finish exercise (check classroom assignment), study correct answers
- Read, again, chapter 7 and basic concepts of chapter 6 in the textbook
- Evaluate classes (check classroom assignment)
- Study questions from previous exams

Questions from previous exams

- Explique brevemente a diferença entre testes de unidade e testes de integração (a). Qual o impacto negativo de realizar apenas os testes de unidade? (b) Qual o impacto negativo de realizar apenas os testes de integração?
- Explique brevemente a diferença entre testes de aceitação e testes de integração, e porque você acha que algumas empresas realizam os dois tipos de teste.

Software development

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba