

INTRODUÇÃO AO CUCUMBER & CAPYBARA

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

Daniel Cordeiro

3 de outubro de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

- Não seria ótimo poder pegar os cartões 3x5 e mapear em testes para o usuário decidir se deve aceitar o app?
- Como você faria para mapear o texto em linguagem natural para código de teste?
- Como você pode fazer para executar os testes sem ter um ser humano para executar as ações?

- Testes a partir de histórias de usuário amigáveis pro cliente
 - Aceitação: garante que o cliente fique satisfeito
 - Integração: garante que as interfaces entre os módulos tenham hipóteses consistentes e que se comunicam corretamente
- Cucumber faz a ponte entre o usuário e o desenvolvedor
 - Foca em histórias de usuário, não em código. É algo que o cliente entende e que ele e desenvolvedor podem usar para chegar em um acordo
 - Mas não são escritos de qualquer jeito, assim podem gerar testes reais

Feature: User can manually add movie 1 funcionalidade

Scenario: Add a movie ≥ 1 cenários/funcionalidade
seguidos de 3--8 passos/cenário

Given I am on the RottenPotatoes home page

When I follow "Add new movie"

Then I should be on the Create New Movie page

When I fill in "Title" with "Men In Black"

And I select "PG-13" from "Rating"

And I press "Save Changes"

Then I should be on the RottenPotatoes home page

And I should see "Men In Black"

- **História de usuário**: tipicamente relacionados a uma funcionalidade
- **Funcionalidade**: ≥ 1 **cenários** que mostram jeitos diferentes de usar uma funcionalidade
 - as palavras chave **Feature** e **Scenario** identificam seus respectivos componentes
 - devem conter *caminhos felizes & tristes*
 - ficam em `features/*.feature`
- **Cenário**: tipicamente 3–8 **passos**
- **Definição dos passos**: código ruby para testar os passos; ficam em `features/step_definitions/*_steps.rb`

4 TIPOS DE PASSOS

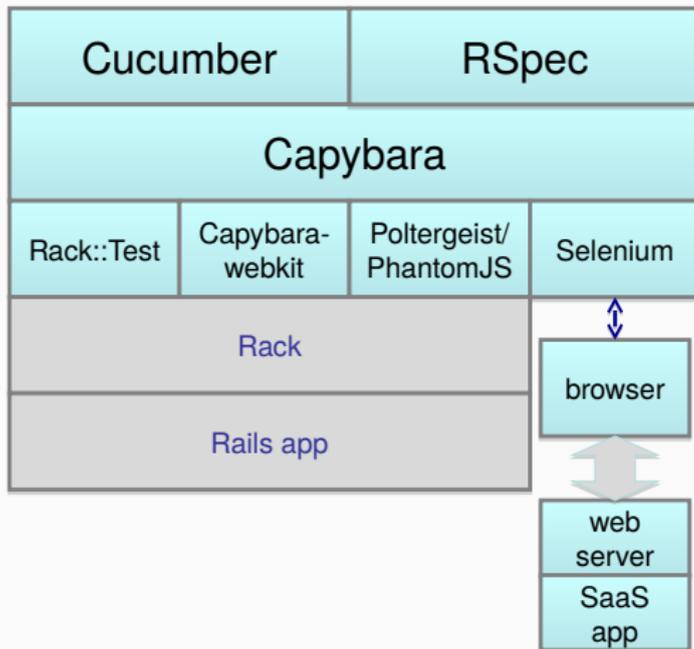
1. **Given:** precondições, representa o estado do mundo antes do evento
2. **When:** passos que representam o evento (ex: simular usuário apertando um botão)
3. **Then:** passos que representam as pós-condições esperadas; verifique se são verdadeiras
4. **And**
5. **But** estendem o passo anterior

Na prática, todos se referem a um mesmo método!

- **Expressões regulares** casam as frases em linguagem natural dos passos dos cenários para as definições dos passos
- Given `/^(?:|I)am on (.+)$/`
- “I am on the Rotten Potatoes home page”
- Definições dos passos (código Ruby) capturaria a string “the Rotten Potatoes home page”

- Ferramenta que finge ser um usuário para seguir os cenários da história de usuário
- **Capybara** simula ser o navegador
 - Pode interagir com o app para receber páginas
 - Analisa o HTML
 - Submete formulários como um usuário normal faria

PILHA DE TESTES DO CUCUMBER



Com o Selenium você pode criar scripts para testar todas as interações externas

- `cucumber` `nomedoarquivo` para executar uma funcionalidade, `rake cucumber` para rodar todas
- **Verde** se todos os passos passarem
- **Amarelo** se não foram implementados
- **Vermelho** para os que falham (os passos seguintes são marcados com **Azul**)
- Objetivo: fazer com que todos os passos fiquem **verdes** (daí vem o nome da ferramenta)

Qual afirmação (se houver) é falsa sobre Cucumber+Capbara?

1. Funcionalidades devem incluir cenários tanto para os caminhos felizes, como para os caminhos tristes
2. C+C são apropriados para testes de integração (full-stack), mas não para testes de unidade/módulo
3. Alguns cenários do Cucumber que rodam usando Rack::Test podem não funcionar se usarmos Selenium
4. Todas as anteriores são verdadeiras; nenhuma é falsa

CENÁRIOS EXPLÍCITOS VS.
IMPLÍCITOS E IMPERATIVOS VS.
DECLARATIVOS

- Será que todos os requisitos vêm de histórias de usuário?
- Cenários devem ter de 3 a 8 passos; há algum modo de deixar esse número mais próximos de 3 do que de 8?

- Requisitos explícitos normalmente são parte dos testes de aceitação
 - provavelmente são uma história de usuário e cenários explícitos:
listar filmes
- Requisitos implícitos são consequências lógicas dos requisitos explícitos, normalmente verificados pelos testes de integração:
 - filmes devem ser listados em ordem cronológica ou alfabética?

- Imperativo: histórias de usuários iniciais, com muitos passos para especificar uma sequência lógica para obter o resultado desejado
- Declarativo: descreva o estado, não a sequência (menos passos)
- Funcionalidade de exemplo: filmes devem aparecer em ordem alfabética e não na ordem em que foram adicionados
- Cenário exemplo: ver lista de filmes após a adição de 2 filmes

EXEMPLO DE CENÁRIO IMPERATIVO

Feature: movies should appear in alphabetical order, not added order

Scenario: view movie list after adding 2 movies (imperativo e non-DRY)

```
Given I am on the RottenPotatoes home page
When I follow "Add new movie"
Then I should be on the Create New Movie page
When I fill in "Title" with "Zorro"
And I select "PG" from "Rating"
And I press "Save Changes"
Then I should be on the RottenPotatoes home page
When I follow "Add new movie"
Then I should be on the Create New Movie page
When I fill in "Title" with "Apocalypse Now"
And I select "R" from "Rating"
And I press "Save Changes"
Then I should be on the RottenPotatoes home page
Then I should see "Apocalypse Now" before "Zorro" on the RottenPotatoes
home page sorted by title
```

Linha em vermelha é a única que especifica comportamento, o resto é implementação. Mas BDD especifica comportamento e não implementação!

- Declarativa, define a linguagem de domínio
- Usa termos e conceitos do app
- Linguagem informal
- Passos declarativos descrevem o estado em que o app deveria estar:
 - vs. Imperativo: sequência de passos para mudar o estado atual até o estado desejado

EXEMPLO DE CENÁRIO DECLARATIVO

Feature: movies should appear in alphabetical order, not added order

Scenario: view movie list after adding movie (declarative and DRY)

Given I have added "Zorro" with rating "PG-13"

And I have added "Apocalypse Now" with rating "R"

Then I should see "Apocalypse Now" before "Zorro" on the RottenPotatoes
home page sorted by title

- 3 passos ao invés de 14; 2 passos para configurar o teste, 1 para o comportamento
- Cenários declarativos concentram a atenção na funcionalidade sendo descrita e testada, não nos passos necessários para testá-la

CENÁRIOS DECLARATIVOS PRECISAM DE NOVAS DEFINIÇÕES DE PASSOS

```
Given /I have added "(*)" with rating "(*)"/ do |title, rating|
  steps %Q{
    Given I am on the Create New Movie page
    When I fill in "Title" with "#{title}"
    And I select "#{rating}" from "Rating"
    And I press "Save Changes"
  }
end
```

```
Then /I should see "(*)" before "(*)" on (.)// do |string1, string2, path|
  step "I am on #{path}"
  regexp = /#{string1}.*#{string2}/m # /m means match across newlines
  page.body.should =~ regexp
end
```

- A medida que o app evoluir, reuse os passos dos primeiros cenários imperativos para criar cenários declarativos mais concisos e descritivos

Qual afirmação é verdadeira sobre cenários Explícitos vs. Implícitos e Imperativos vs. Declarativos?

1. Em geral, requisitos explícitos são definidos com cenários imperativos e requisitos implícitos são definidos com cenários declarativos
2. Cenários explícitos normalmente capturam testes de integração
3. Cenários declarativos tem como objetivo capturar tanto implementação como comportamento
4. Todas são falsas

A PERSPECTIVA PLANEJE-E-DOCUMENTE

- O que o Planeje-e-Documente usa ao invés de:
 - histórias de usuário?
 - pontos?
 - velocity?
- Como um gerente de projeto estima os custos? Como ele propõe um cronograma?

1. Levantamento de requisitos
2. Documentação de requisitos
3. Estimativa de custos
4. Planejamento e acompanhamento do progresso
5. Gestão de mudanças dos requisitos, custos e cronograma
6. Garantir que a implementação corresponda às funcionalidades descritas pelos requisitos
7. Análise de risco e gestão

Levantamento de requisitos **funcionais** e **não-funcionais**

1. **Entrevistas** – veja como é que realmente é feito atualmente
 - *stakeholders* respondem a um questionário predefinido
 - ou apenas baseado em discussões informais
2. Criação cooperativa de **cenários**
 - estado inicial, fluxos de execução dos caminhos felizes e tristes, o que é concorrente, estado final
3. Criação de **casos de uso**
 - lista de passos de usuário e sistema para atingir um objetivo; descrito usando linguagem UML

- Documentação de requisitos usando o **Software Requirement Specification (SRS)**
 - 100s de páginas; padrão IEEE para SRS!
- Faz os *stakeholders* lerem o SRS ou constroem o protótipo básico ou geram casos de testes para verificar:
 - Validade** todos os requisitos são realmente necessários?
 - Consistência** os requisitos são conflitantes?
 - Completude** todos os requisitos e restrições foram incluídos?
 - Viabilidade** os requisitos podem ser implementados?

- Gerente divide o SRS em tarefas
- Estima o número de semanas por tarefa ($1 \text{ semana} \leq \text{tarefas} \leq 8 \text{ semanas}$)
- Converte o valor de pessoas-semana em \$ usando os salários e sobrecusto
- Estima *antes & depois* do contrato:
 - adiciona margem de segurança: 1,3 a 1,5 \times
 - faz 3 estimativas (melhor caso, esperado, pior caso) e faz o melhor palpite

1. Estimativa experimental

- depende da experiência do gerente para ser precisa

2. Estimativa quantitativa

- estima as tarefas em linhas de código (LOC), divide LOC/pessoas-mês

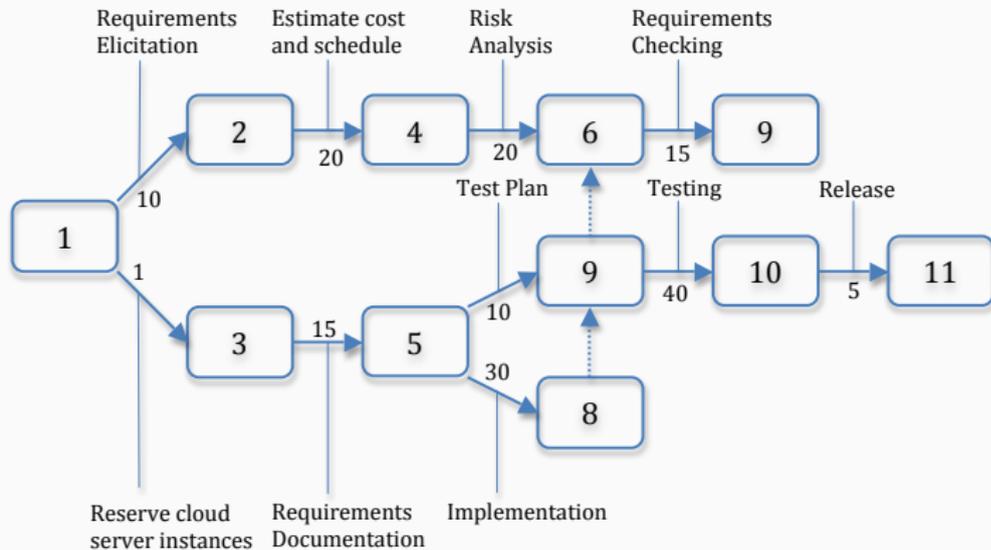
- COCOMO (*Constructive Cost Model*)

$$\text{Esforço} = \text{Fatores Organizacionais} \times \text{Tam. do Código}^{(\text{Penalidade pelo Tam.} \times \text{Fatores do Produto})}$$

- Fatores organizacionais = 2,94; $1,1 \leq \text{Penalidade pelo Tam.} \leq 1,24$; $0,9 \leq \text{Fatores do Produto} \leq 1,4$
- 92% usam experiência vs. fórmula

P-E-D: PLANEJAMENTO

- Usa um diagrama de PERT para mostrar o paralelismo das tarefas e o caminho crítico para fazer o cronograma
 - nós são os marcos, as arestas etiquetadas são as tarefas, os números são o esforço e as setas as dependências



- Compara o predito ao atual
 - tempo para realizar tarefas
 - despesas
- Marcos (*milestones*) intermediários ajudam os *stakeholders* a verem se o projeto está dentro do prazo e do orçamento

- Gerente usa ferramentas para a rastreabilidade de requisitos
- Ferramenta tem as referências cruzadas entre:
 - parte do SRS com o requisito
 - parte do código que implementa o requisito
 - testa e valida os requisitos

- Melhora a acurácia do orçamento/cronograma
- Identifica os riscos para tão cedo qto possível:
 - realizar trabalho extra que reduza o risco
 - mudar o plano para evitar o risco
- Técnico: “RDB não escala”
- Organizacional: “não estamos familiarizados com J2EE”
- Negócio: terminamos muito tarde para sermos competitivos no mercado
- Cria uma tabela de riscos: probabilidade x impacto (escala de 1-4)
 - tentamos resolver os 20% mais significantes na esperança que eles correspondam a 80% dos riscos potenciais

P-E-D VS. ÁGIL: REQUISITOS E ESTIMATIVA DE CUSTO/PRAZOS

<i>Tasks</i>	<i>In Plan and Document</i>	<i>In Agile</i>
Requirements Documentation	Software Requirements Specification such as IEEE Standard 830-1998	User stories, Cucumber, Points, Velocity
Requirements Elicitation	Interviews, Scenarios, Use Cases	
Change Management for Requirements, Schedule, and Budget	Version Control for Documentation and Code	
Ensuring Requirements Features	Traceability to link features to tests, reviews, and code	
Scheduling and Monitoring	Early in project, contracted delivery date based on cost estimation, using PERT charts. Milestones to monitor progress	Evaluate to pick range of effort for time and materials contract
Cost Estimation	Early in project, contracted cost based on manager experience or estimates of task size combined with productivity metrics	
Risk Management	Early in project, identify risks to budget and schedule, and take actions to overcome or avoid them	

Quais afirmações relacionadas ao levantamento de requisitos e estimativa de custo é falsa?

1. O conceito mais próximo a cronograma e tarefas de monitoramento de P-e-D são os pontos Ágeis e Velocity
2. O mais perto do documento *Software Requirements Specification* (SRS) em Ágil são as histórias de usuário
3. Métodos ágeis não tem um equivalente para garantir os requisitos, tal como rastreabilidade
4. Todas são verdadeiras, nenhuma é falsa

ARMADILHAS & FALÁCIAS, PRÓS & CONS DE BDD

- Entregar uma história como “pronta” quando apenas o caminho feliz foi testado
 - é preciso testar tanto o caminho feliz quanto o triste
- Comportamento correto do app; uma ação incorreta realizada acidentalmente por um usuário é tão importante quanto o comportamento correto, quando ele faz a coisa certa
 - Errar é humano

- Uso descuidado de expressões negativas
 - cuidado ao abusar de expressões como “Then I should not see...”
 - é difícil saber se a saída é o que você quer, ou se é o que você não quer
 - muitas saídas são incorretas
- Inclua verificações de resultado positivas
 - “Then I should see ...”

- Uso descuidado de expectativas positivas
 - `Then I should see "Emma"`, mas e se a string aparecer várias vezes na página?
 - Pode passar mesmo que a funcionalidade não esteja funcionando
- Use o auxiliar `within` do Capybara
 - restringe o escopo a um seletor CSS
 - `Then I should see "Emma" within "div#shopping_cart"`
 - Veja a documentação do Capybara