Engenharia de Software

Prof. Ms. Gustavo Molina

Aula 08 – Testes de *Software II*

msc.gustavo.unip@gmail.com

Testes de *Software*

Na aula passada vimos os seguintes tipos de teste: **unitário, integração e caixa branca.**

Hoje iniciaremos falando sobre os outros tipos de teste .

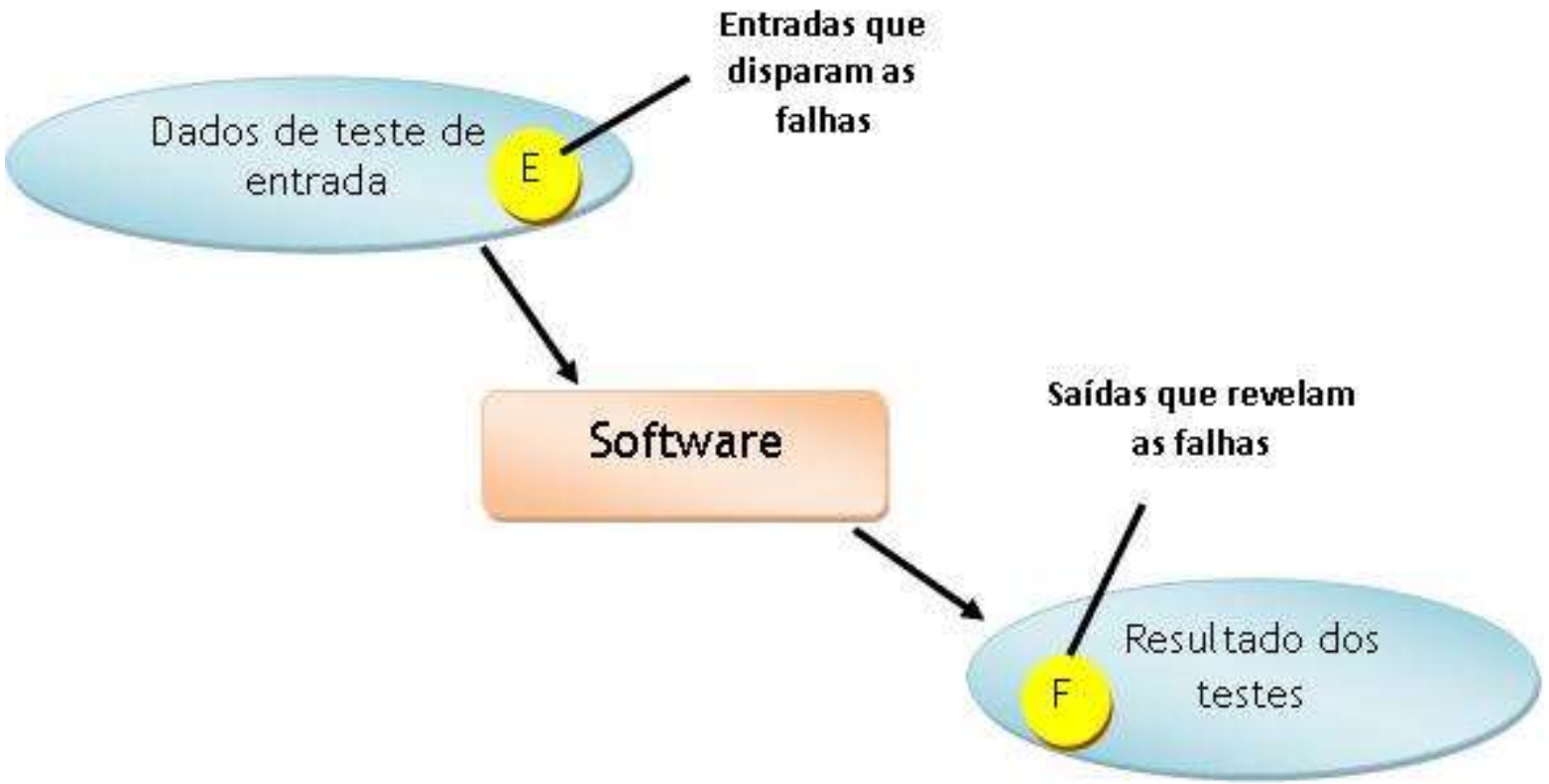
Teste de Caixa Preta

- É uma técnica de teste de *software* com o intuito de verificar a “qualidade” da nossa aplicação.
- Sua principal função é testar o programa de maneira funcional, ou seja, a caixa preta não olha para o código e sim para a funcionalidade do programa.

Teste de Caixa Preta

- O teste da Caixa Preta, baseia-se nos requisitos, ou seja, em quais passos a aplicação deve seguir para chegar em determinado resultado.
- O teste depende do fornecimento de dados de entrada da aplicação, e de um resultado esperado. Caso o resultado seja igual ao esperado o teste foi concluído com sucesso, caso contrário constata-se que há um problema funcional com a aplicação.

Teste de Caixa Preta



Teste de Caixa Preta

Teste de Regressão

- É o tipo de teste feito para verificar se uma mudança no código não impacta as funcionalidades já existentes do produto.
- Segundo Pressman (1997), "o teste de regressão é a reexecução de um subconjunto de testes que já foram conduzidos para garantir que as modificações não propagaram efeitos colaterais indesejáveis."

Teste de Regressão - Objetivo

- O objetivo de um teste de regressão é ter certeza de que nenhum defeito foi adicionado ao produto depois de alguma modificação. E que mesmo depois de uma modificação, o *software* continue cumprindo os requisitos do sistema.
- Por isso, o teste de regressão serve para testar novamente as funcionalidades que já foram testadas antes.

Teste de Regressão – Quando usar?

- O teste de regressão é normalmente feito depois de uma mudança significativa, que pode ser no código, no design ou qualquer outro aspecto que mexa com a estrutura do sistema, garantindo sua integridade.
- As situações mais comuns para se fazer testes de regressão são quando novas versões do código são criadas ou quando bugs são consertados.

Teste de Regressão – Tipos

• **Regressão Unitária:** teste é feito isoladamente, ou seja, a dependência do código que será testada é bloqueada para que essa unidade específica seja testada sem discrepância.

• **Regressão Parcial:** feita para verificar se o código funciona bem mesmo que mudanças tenham sido feitas no código e essa unidade esteja integrada com partes do código já existentes.

• **Regressão Completa:** é feito quando uma alteração no código é feita em vários módulos e também se o impacto de uma alteração em qualquer outro módulo for incerto. O *software* como um todo é regredido para verificar quaisquer alterações devido ao código alterado.

Teste de Regressão – Como Usar

Passo-a-passo:

• Entender que tipo alterações foram feitas no *software* e quais módulos do *software* podem ser impactados pelas mudanças.

• Priorizar as mudanças e os requisitos do produto.

• Definir ponto inicial antes da execução do teste.

Teste de Regressão – Como Usar

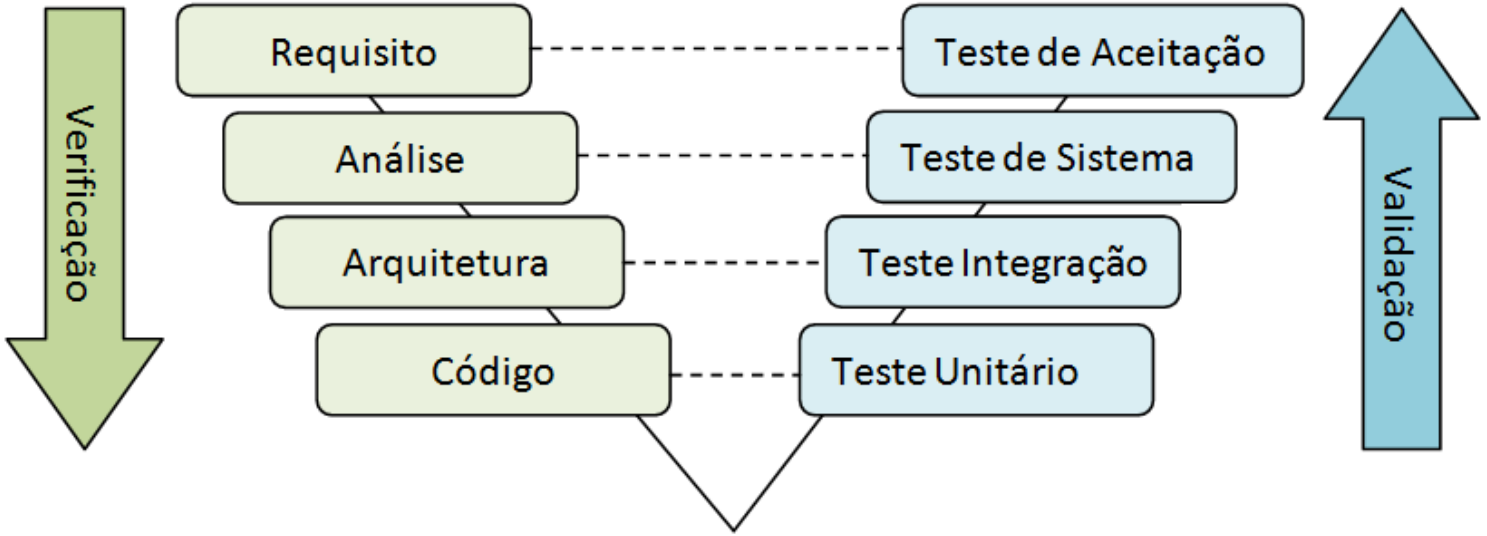
- Determinar um ponto final com as condições mínimas predefinidas no passo anterior.
- Executar o teste.

Teste de Aceitação

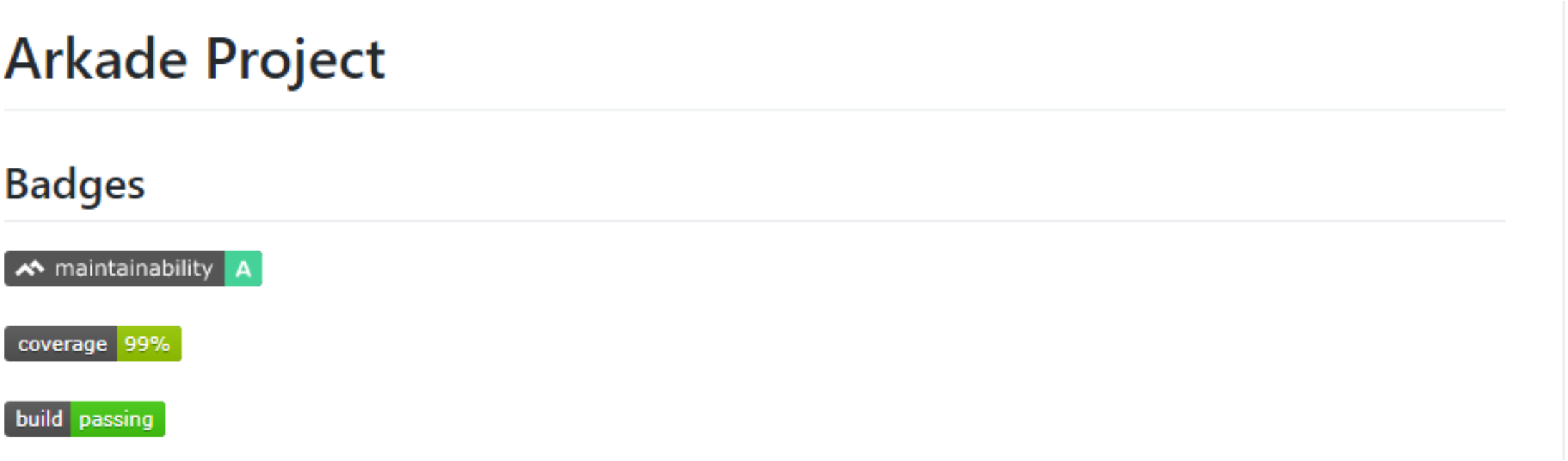
- O teste de aceitação é um tipo de teste aplicado capaz de verificar se um determinado *software* e suas funções e tarefas já estão prontos para serem usados.
- O teste de aceitação tem como objetivo verificar o sistema em relação aos requisitos originais e as necessidades do usuário. Consiste em um processo sistemático capaz de identificar prováveis defeitos. Ele verifica os *softwares* conforme os requisitos fornecidos.

Teste de Aceitação

- Antes de um *software* ser implementado, é dado o teste final, que por sua vez é dado pelo teste de aceitação.



TDD e BDD na Prática



https://github.com/Arkade-Team/Arkade

BDD e TDD na Prática

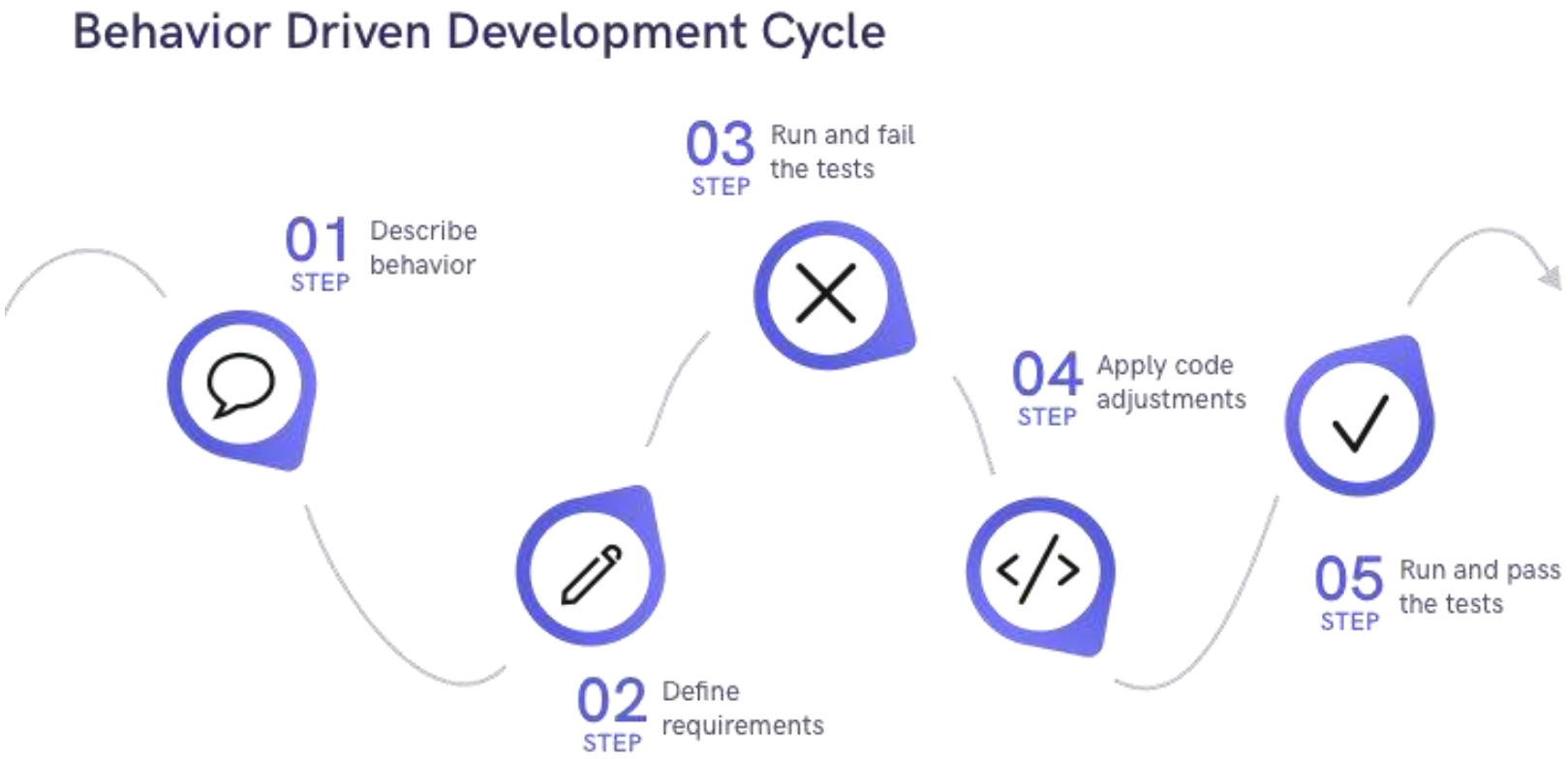
master Arkade / spec / models / appointment_spec.rb / <> Jump to

```ruby
require 'rails_helper'

RSpec.describe Appointment, type: :model do
  describe "validation" do
    before(:each) do
      @appointment = Appointment.new
    end

    it "fails to create without sex" do
      @appointment.age = 42
      expect(@appointment).not_to be_valid
    end

    it "fails to create without age" do
      @appointment.sex = "female"
      expect(@appointment).not_to be_valid
    end

    it "fails to create when age is not number" do
      @appointment.age = "quarenta e dois"
      @appointment.sex = "female"
      expect(@appointment).not_to be_valid
    end
```

RSpec é uma ferramenta de teste de linguagem específica de domínio de computador escrita na linguagem de programação Ruby para testar o código Ruby. É uma estrutura de desenvolvimento orientada por comportamento que é amplamente usada em aplicativos de produção.

```ruby
require 'rails_helper'

RSpec.describe Disease, type: :model do
  before(:each) do
    Disease.delete_all
  end

  describe "instanciation" do
    before(:each) do
      @disease = Disease.new
    end

    it "fails to create without name" do
      expect(@disease).not_to be_valid
    end

    it "fails to create duplicated name" do
      dup_name = "Foo"
      Disease.create(name: dup_name)
      @disease.name = dup_name
      expect(@disease).not_to be_valid
    end

    it "fails to create names that are too short" do
      @disease.name = "a"
      expect(@disease).not_to be_valid
    end
  end
```

master Arkade / spec / models / disease_spec.rb / Jump to



17 lines (14 sloc) | 731 Bytes

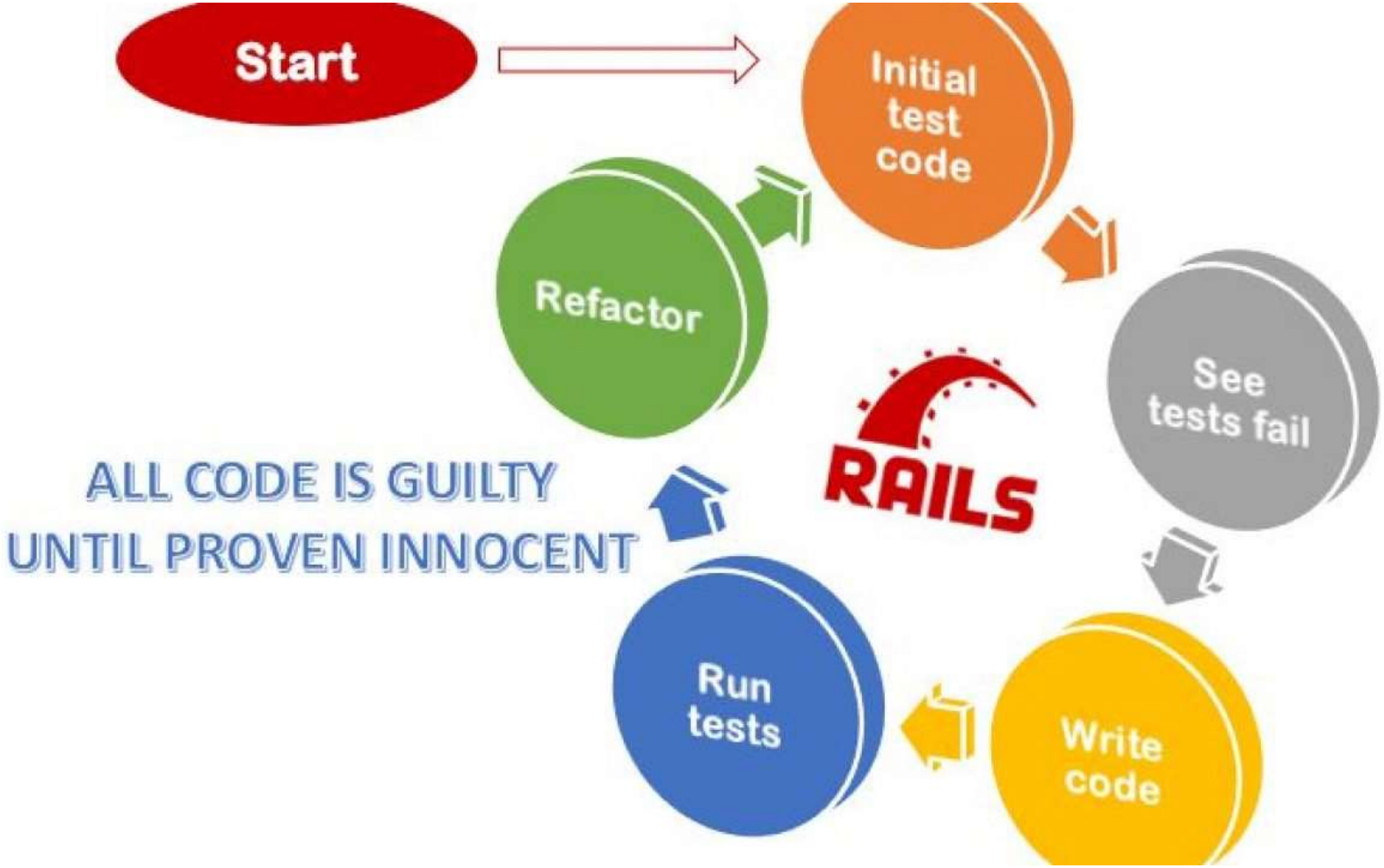
Raw Blame

```
#language: pt

Funcionalidade: Cadastro de Wiki
  COMO coordenador do HC
  PARA acompanhar os acesso às Wikis no aplicativo
  QUERO cadastrar Wikis e suas respectivas quantidades de leituras feitas no app

  Cenário: leitura de wiki no aplicativo
    Dado nenhum registro anterior de wiki
    Quando recebo uma requisição de leitura da Wiki "Calculadora FRAX"
    Então a resposta deve conter o registro da Wiki "Calculadora FRAX"
    E a resposta deve conter um contagem de leitura igual a 1

  Cenário: visualização de wiki
    Dado o usuário na página principal da aplicação
    Quando o cliente clica em "Wikis"
    Então ele visualiza uma tabela contendo os nomes e as contagens de leitura das wikis registradas
```

master Arkade / features / criacao_wikis.feature

Teste de Aceitação

BDD e TDD na Prática



BDD e TDD na Prática

BDD + TDD

Projeto guiado por comportamento (BDD)

- desenvolva histórias de usuários (as funcionalidades que você quer ter) para descrever como o app irá funcionar
- usando Cucumber, histórias de usuários viram *testes de aceitação* e *testes de integração*

Desenvolvimento guiado por testes (TDD)

- cada *definição de passo* para uma nova história pode precisar que se desenvolva novo código
- TDD advoca que: escreva os testes de unidade & funcionais primeiro, antes de escrever o código
- ou seja, escreva testes para **o código que você gostaria de ter**

BDD e TDD na Prática

- Pense naquilo que seu código *deveria* fazer
- Capture a ideia em um teste, que irá falhar
- Escreva o código mais simples possível que faria o código do teste passar
- Refatore. Simplifique o que for comum a vários testes
- Continue com a próxima coisa que o código deveria fazer

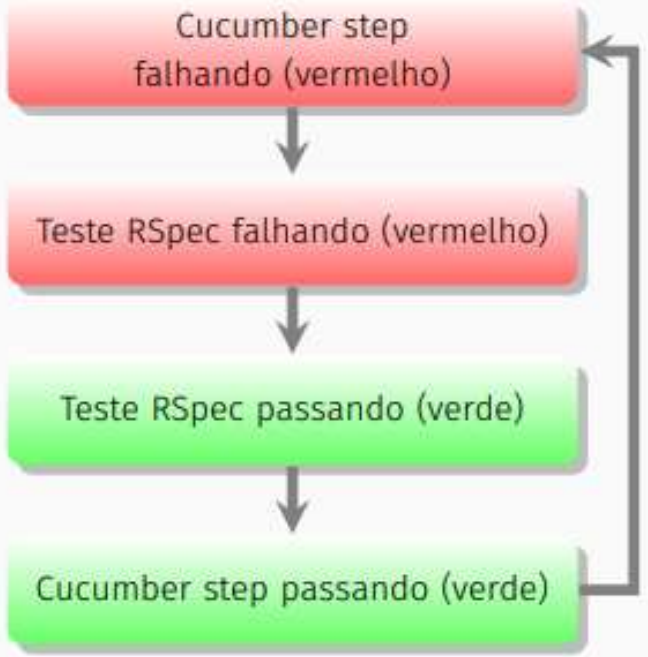
Vermelho–Verde–Refatore

Tente "sempre ter código funcionando"

BDD e TDD na Prática

CUCUMBER & RSPEC

- Cucumber descreve o *comportamento* com as funcionalidades & cenários (projeto guiado pelo comportamento)
- RSpec testa os módulos individuais que contribuem com esses comportamentos (desenvolvimento guiado por testes)



Dúvidas?