

ENGENHARIA DE SOFTWARE

TESTES DE SOFTWARE

Profº Ms Gustavo Molina
msc.gustavo.unip@gmail.com

Prof. Ms Gustavo Molina



- Graduado em Sistemas de Informação pelo MACKENZIE.
- Licenciado em Matemática pela UNIP.
- Pós – Graduado em Plataforma de Desenvolvimento Web pelo CLARETIANO.
- Pós – Graduado em IA pela faculdade Serra Geral
- Pós – Graduado em Gestão e Governança de Tecnologia da Informação pela UNIP
- Mestre em Engenharia Elétrica pela FEI
- Doutorando em Ciências da Educação pela Ivy Enber Christian University

Prof. Ms Gustavo Molina



<https://www.linkedin.com/in/gustavo-molina-a2798418/>



<http://lattes.cnpq.br/8512452850609937>



msc.gustavo.unip@gmail.com



<https://github.com/gustavomolina17>

Testes de Software

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

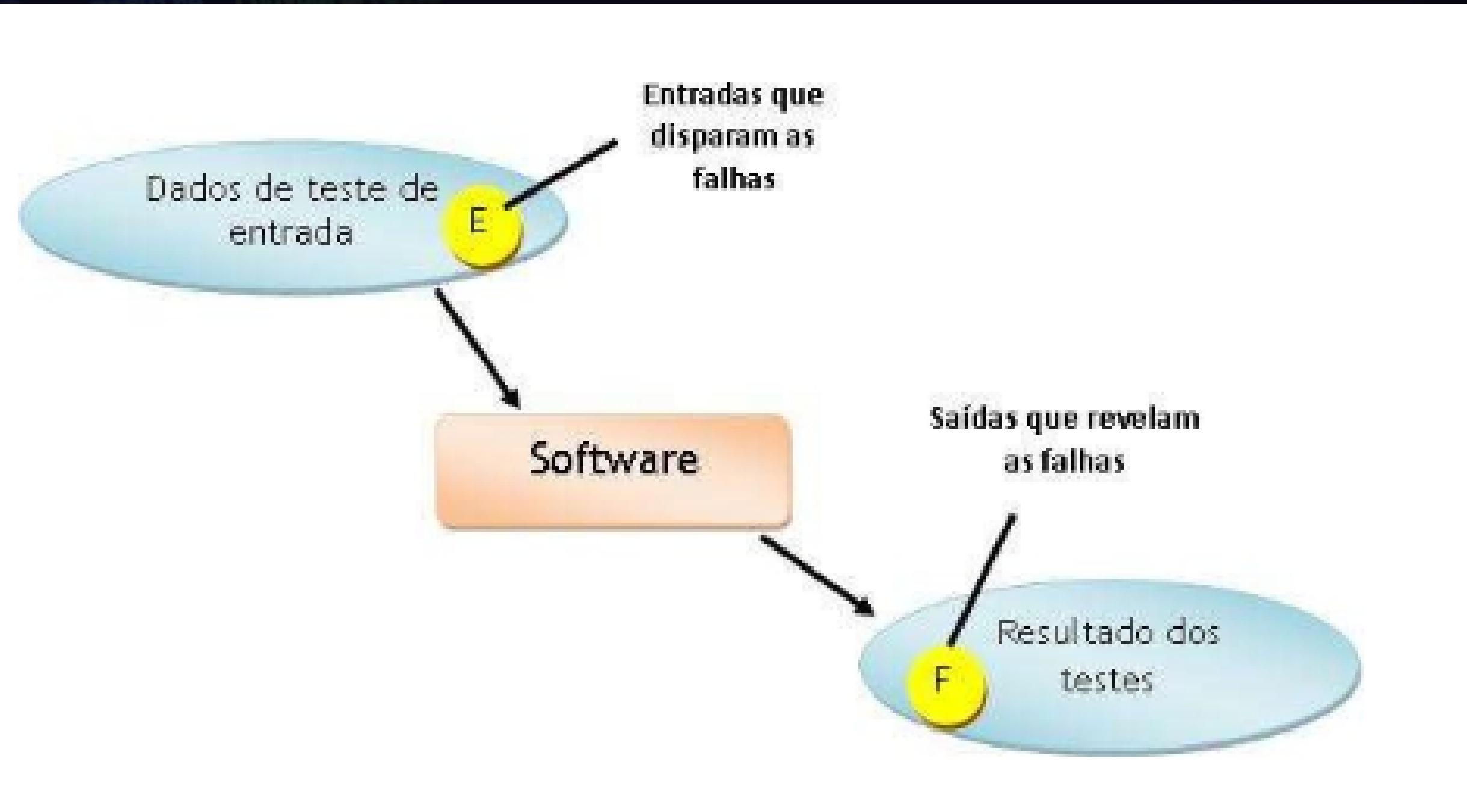
Hoje iniciaremos falando sobre os outros tipos de teste



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.
- 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

Testes 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 o 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.

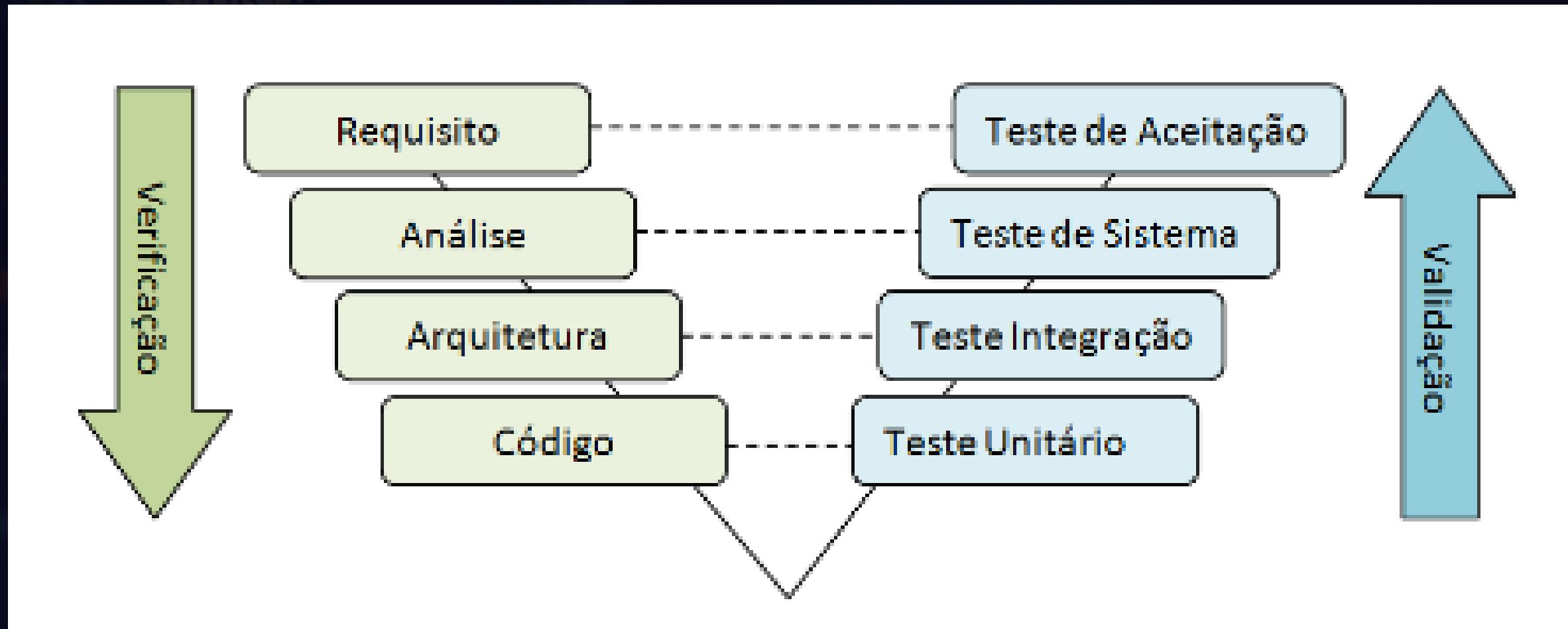


Teste de Aceitação

- 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.

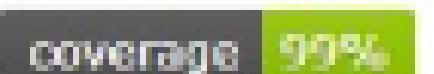


TDD e BDD na Prática

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

Arkade Project

Badges



Tipos de Teste de Integração

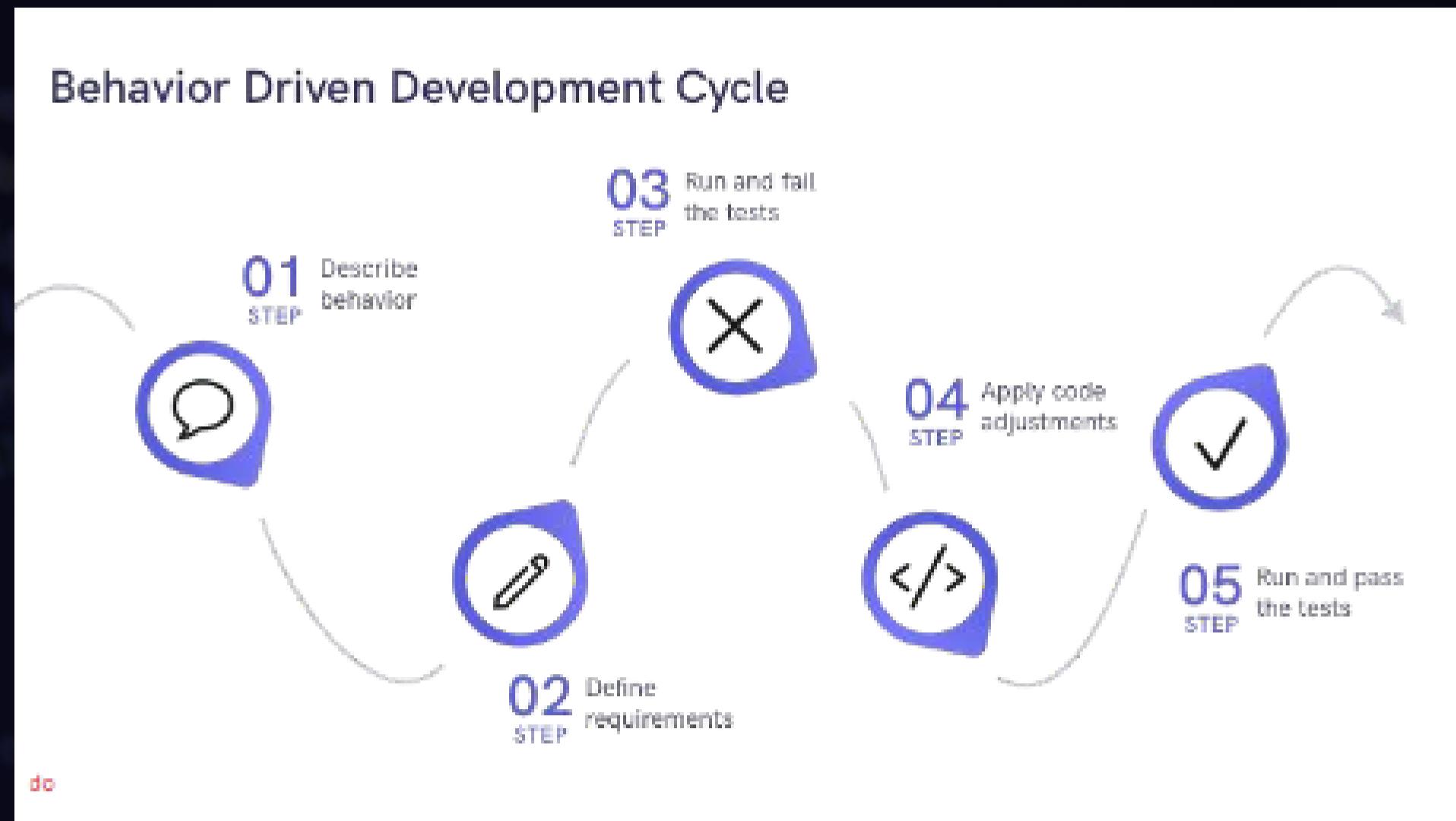
- RSpec é uma ferramenta de teste de linguagem específica de domínio de computador escrita na linguagem de programação Ruby para testar o código Ruby. É uma estrutura de desenvolvimento orientada por comportamento que é amplamente usada em aplicativos de produção.

```
1  require 'rails_helper'  
2  
3  RSpec.describe Appointment, types: :model do  
4    describe "validation" do  
5      before(:each) do  
6          @appointment = Appointment.new  
7      end  
8  
9      it "fails to create without sex" do  
10        @appointment.age = 42  
11        expect(@appointment).not_to be_valid  
12      end  
13  
14      it "fails to create without age" do  
15        @appointment.sex = "Female"  
16        expect(@appointment).not_to be_valid  
17      end  
18  
19      it "fails to create when age is not number" do  
20        @appointment.age = "quarenta e dois"  
21        @appointment.sex = "Female"  
22        expect(@appointment).not_to be_valid  
23      end
```

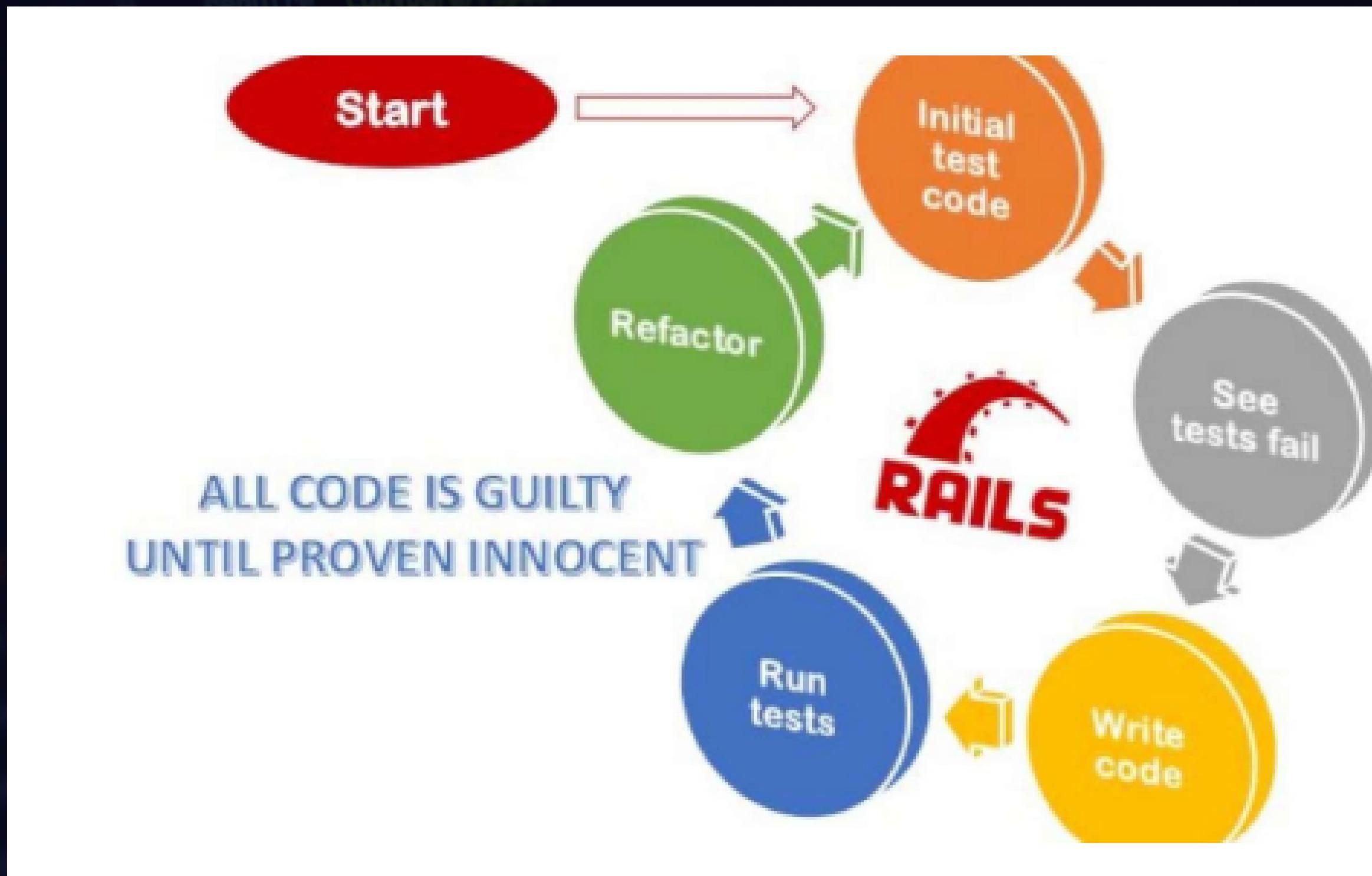


```
1  require 'rails_helper'  
2  
3  RSpec.describe Disease, type: :model do  
4    before(:each) do  
5      Disease.delete_all  
6    end  
7  
8    describe "instantiation" do  
9      before(:each) do  
10        @disease = Disease.new  
11      end  
12  
13      it "fails to create without name" do  
14        expect(@disease).not_to be_valid  
15      end  
16  
17      it "fails to create duplicated name" do  
18        dup_name = "Foo"  
19        Disease.create(name: dup_name)  
20        @disease.name = dup_name  
21        expect(@disease).not_to be_valid  
22      end  
23  
24      it "fails to create names that are too short" do  
25        @disease.name = "a"  
26        expect(@disease).not_to be_valid  
27      end  
28    end
```

Behavior



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???

```
5   abort("The Rails environment is missing. Please run 'rails server' to start a test server." unless ENV["RAILS_ENV"] == "test")
6
7   require 'spec_helper'
8
9   require 'capybara/rspec'
10  require 'capybara/rails'
11
12  Capybara.javascript_driver = :webkit
13  Category.delete_all; Category.create!(name: "Category 1")
14  Shoulda::Matchers.configure do |config|
15    config.integrate do |with|
16      with.test_framework :rspec
17      with.library :rails
18    end
19  end
20
21  # Add additional requires below this line.
22
23  # Requires supporting files within the same directory as this file or,
24  # if further up the tree, look for them in the `spec/support` directory
25  # run as spec files to be found in the `spec` directory
26  # in _spec.rb will both be loaded by `require` by default.
27  # It is recommended to keep all required code within this file, but feel
28  # free to move it to a test support file if it gets too complex. Support files
29  # can be shared between tests in the same directory with the same filename
```