

# 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



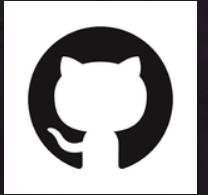
<https://www.linkedin.com/in/gustavo-molina-a2798418/>



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



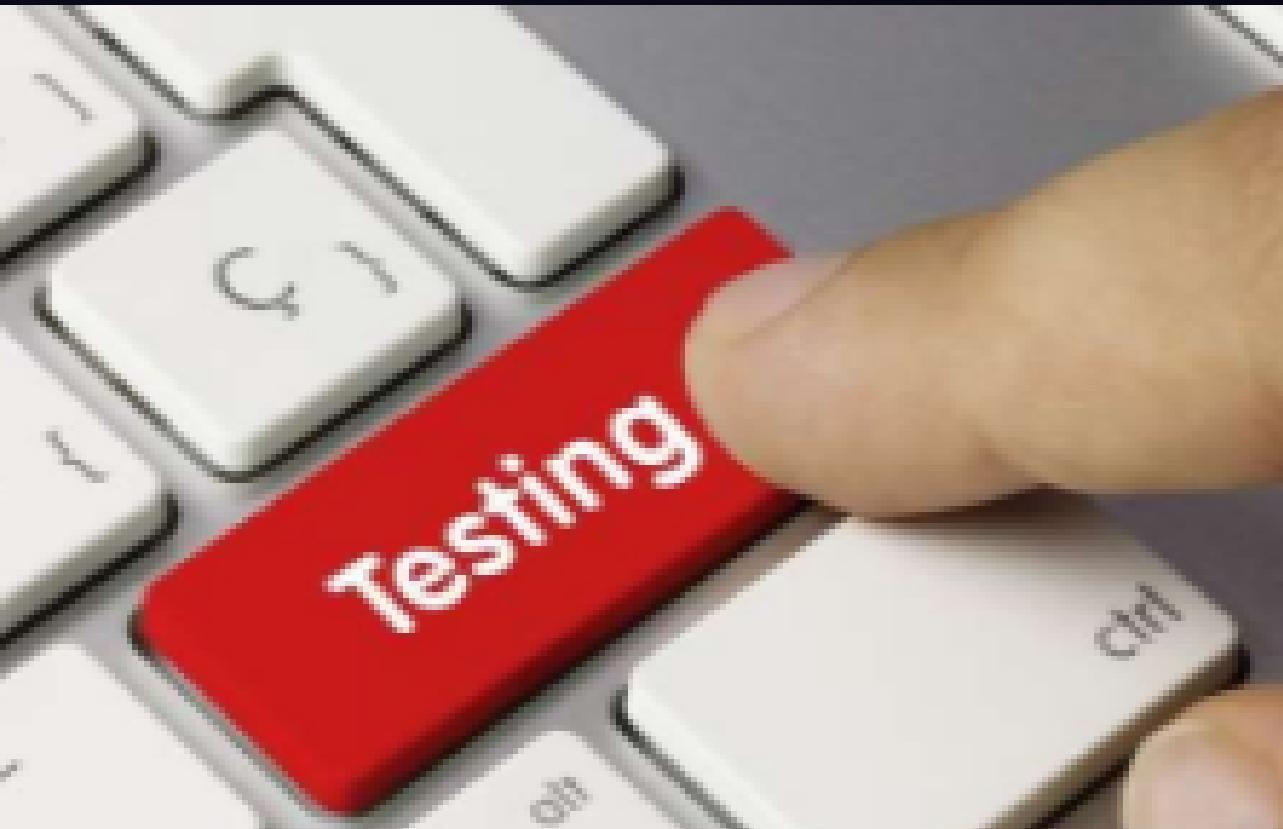
[msc.gustavo.unip@gmail.com](mailto:msc.gustavo.unip@gmail.com)



<https://github.com/gustavomolina17>

# Testes de Software

Testes de *software* são processos que fazem parte de um projeto de desenvolvimento de um *software*, com o objetivo de descobrir falhas no sistema, reportar erros e verificar se os mesmos foram corrigidos, garantindo uma qualidade maior na entrega do produto.



```
5 abort("The Rails environment is not supported by this generator")
6 require 'spec_helper'
7 require 'rspec/rails'
8
9 require 'factory_girl_rails'
10
11 Category.delete_all
12 Shoulda::Matchers::DSL::Category
13
14 Shoulda::Matchers::DSL::Category
15
16 Category.create!(name: "Electronics", description: "Electronics category")
17 Category.create!(name: "Clothing", description: "Clothing category")
18 Category.create!(name: "Books", description: "Books category")
19 Category.create!(name: "Sports", description: "Sports category")
20 Category.create!(name: "Home", description: "Home category")
21 Category.create!(name: "Food", description: "Food category")
22 Category.create!(name: "Tech", description: "Tech category")
23 Category.create!(name: "Automotive", description: "Automotive category")
24 # run as superuser
25 # in _spec.rb will run twice. It is
26
```

# Qual a importância dos testes de softwares para as empresas??



# Testes de Software

Existem 13 tipos principais de teste de *software*:

1. Teste de Configuração.
2. Teste de Instalação.
3. Teste de integridade.
4. Teste de Segurança.
5. Teste Funcional.
6. Teste de Unidade.
7. Teste de Integração.
8. Teste de Volume.



# Testes de Software

- Existem 13 tipos principais de teste de software:
    9. Teste de Performance.
    10. Teste de Usabilidade.
    11. Testes de Caixa Branca e Caixa Preta.
    12. Teste de Recessão.
    13. Teste de Manutenção.



# Testes de Software

- Teste de Unidade: Testa um componente ou classe isolada do sistema.
- Teste de Integração: Testa se um ou mais componentes combinados funcionam de maneira satisfatória.
- Teste de Volume: Testar o comportamento do sistema operando com um volume “normal” de dados e transações envolvendo o banco de dados durante um longo período de tempo.
- Teste de Usabilidade: Teste focado na experiência do usuário, consistência de interface, layout, acesso às funcionalidades, entre outros.
- Testes de Caixa Branca e Caixa Preta: Testes de caixa branca envolvem o código e o de caixa preta não.

# Testes de Software

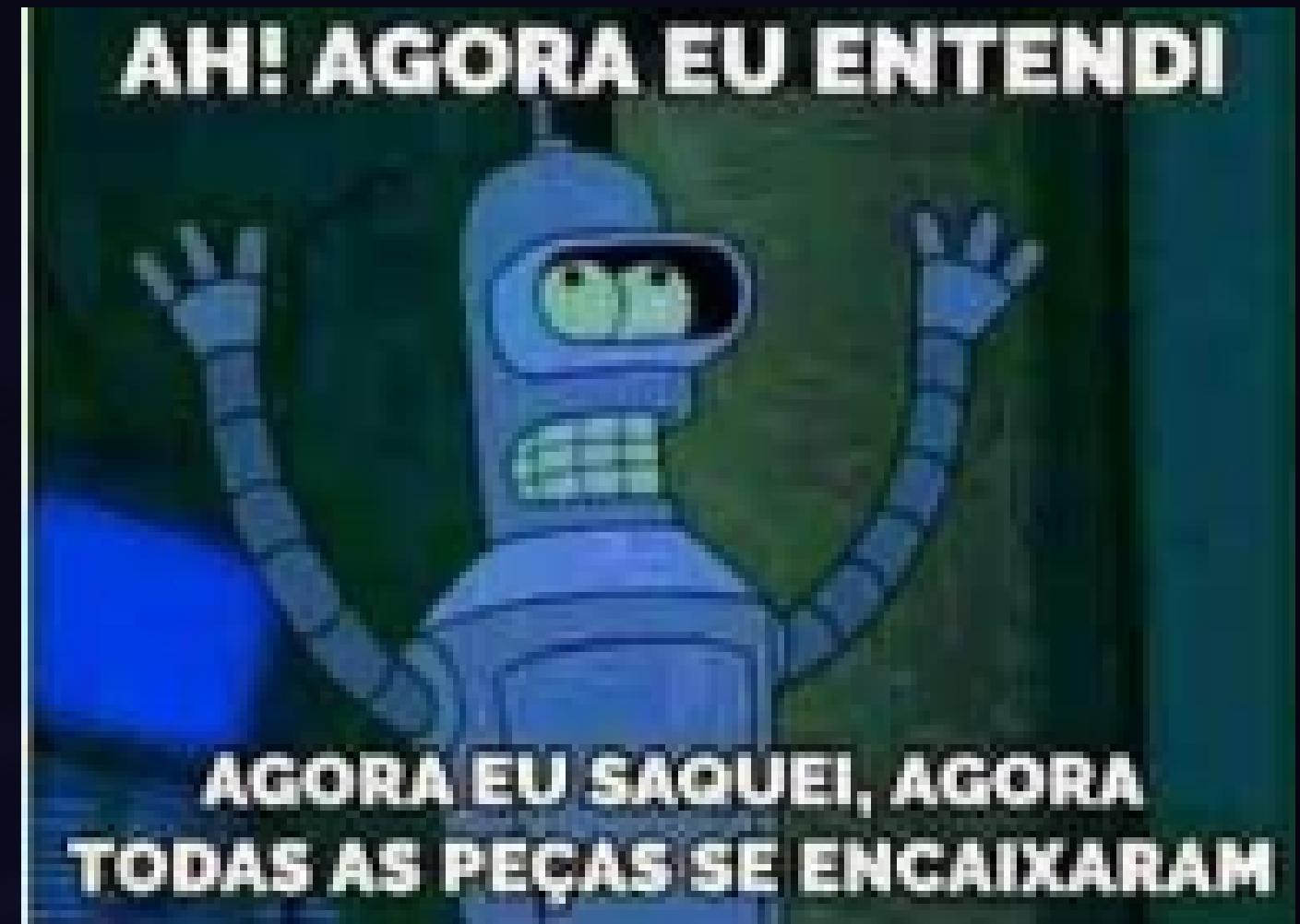
- Teste de Performance: É subdividido em três grupos:
- Teste de Carga: Testar o software sob as condições normais do uso. Ex: tempo de resposta, número de transações por minuto, usuários simultâneos.
- Teste de Stress: Testar o software sob condições extremas de uso. Grandes volumes de transações e usuários simultâneos. Picos excessivos de cargas em curtos períodos de tempo.
- Teste de Estabilidade: Testa se o sistema se mantém funcionando de maneira satisfatória após um período de uso.

# Testes de Software

- Teste de Performance: É subdividido em três grupos:
- Teste de Carga: Testar o software sob as condições normais do uso. Ex: tempo de resposta, número de transações por minuto, usuários simultâneos.
- Teste de Stress: Testar o software sob condições extremas de uso. Grandes volumes de transações e usuários simultâneos. Picos excessivos de cargas em curtos períodos de tempo.
- Teste de Estabilidade: Testa se o sistema se mantém funcionando de maneira satisfatória após um período de uso.

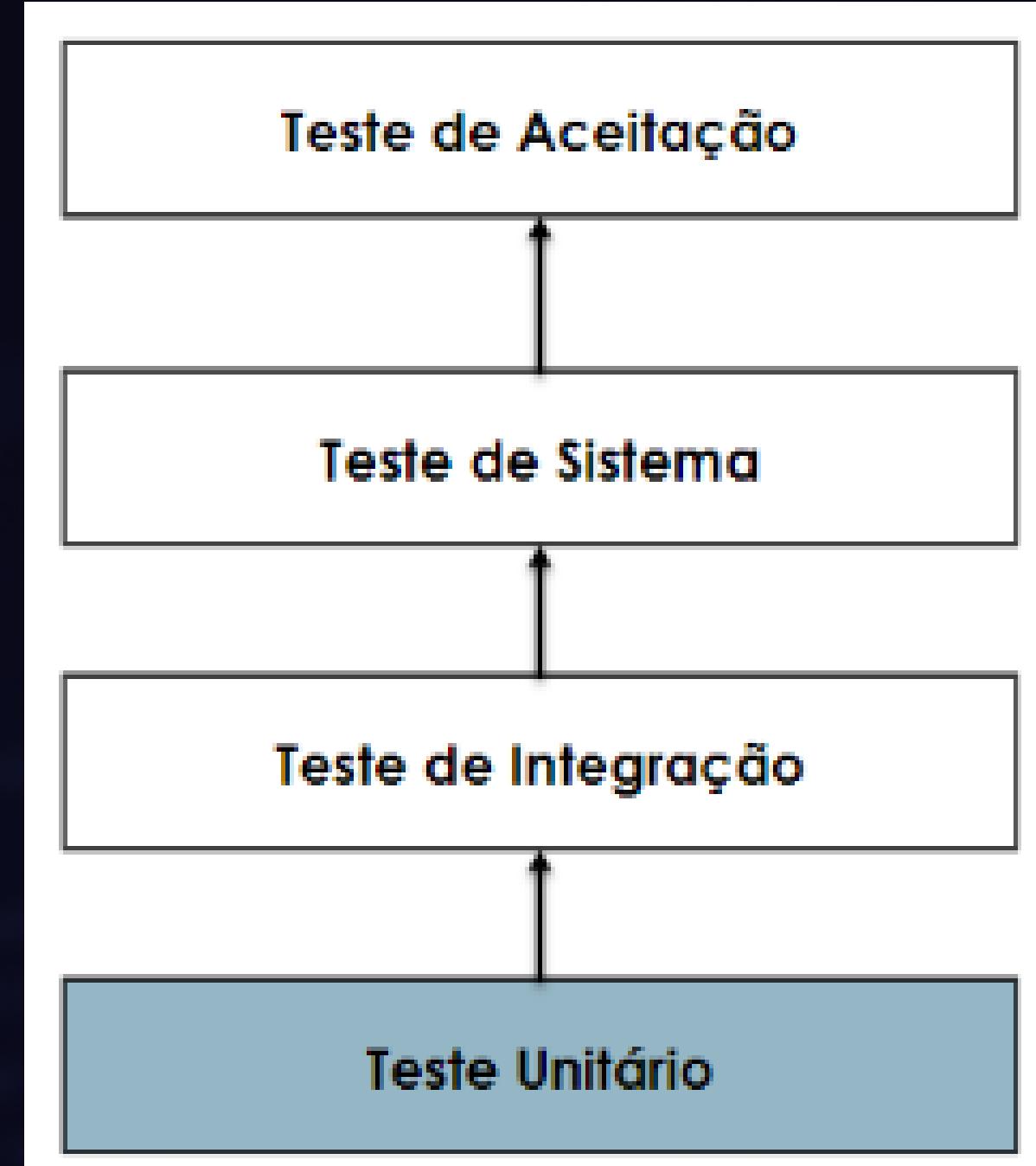
# Testes de Software

- Teste de Recessão: Reteste de um sistema ou componente para verificar se alguma modificação recente causou algum efeito indesejado, além de certificar se o sistema ainda atende os requisitos.
- Teste de Manutenção: Testa se a mudança de ambiente não interferiu no funcionamento do sistema.

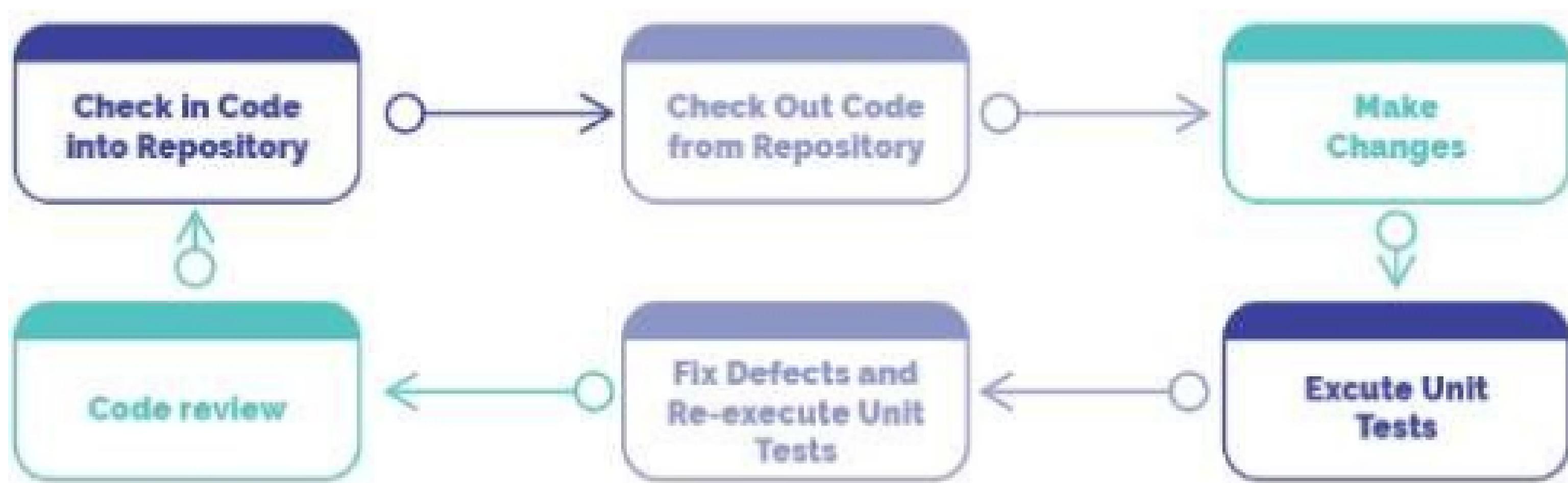


# Teste Unitário

- É o primeiro nível de teste de software, em que cada unidade de um sistema é testada individualmente. Com o propósito de validar a performance de cada uma para que o software funcione conforme o projetado.



# Teste Unitário



# Teste Unitário

- Geralmente o teste unitário é realizado na estrutura automatizada, porém, existe a possibilidade de ser realizado manualmente, com o auxílio de um documento instrucional sobre todos os tipos de aplicativos móveis e da web
- Após a implementação do aplicativo, o código pode ser removido.
- Após a finalização do teste, as dependências podem ser eliminadas.

A maioria dos desenvolvedores utilizam a estrutura automatizada para registrarem os casos de teste com falha.

# Teste Unitário - Vantagens

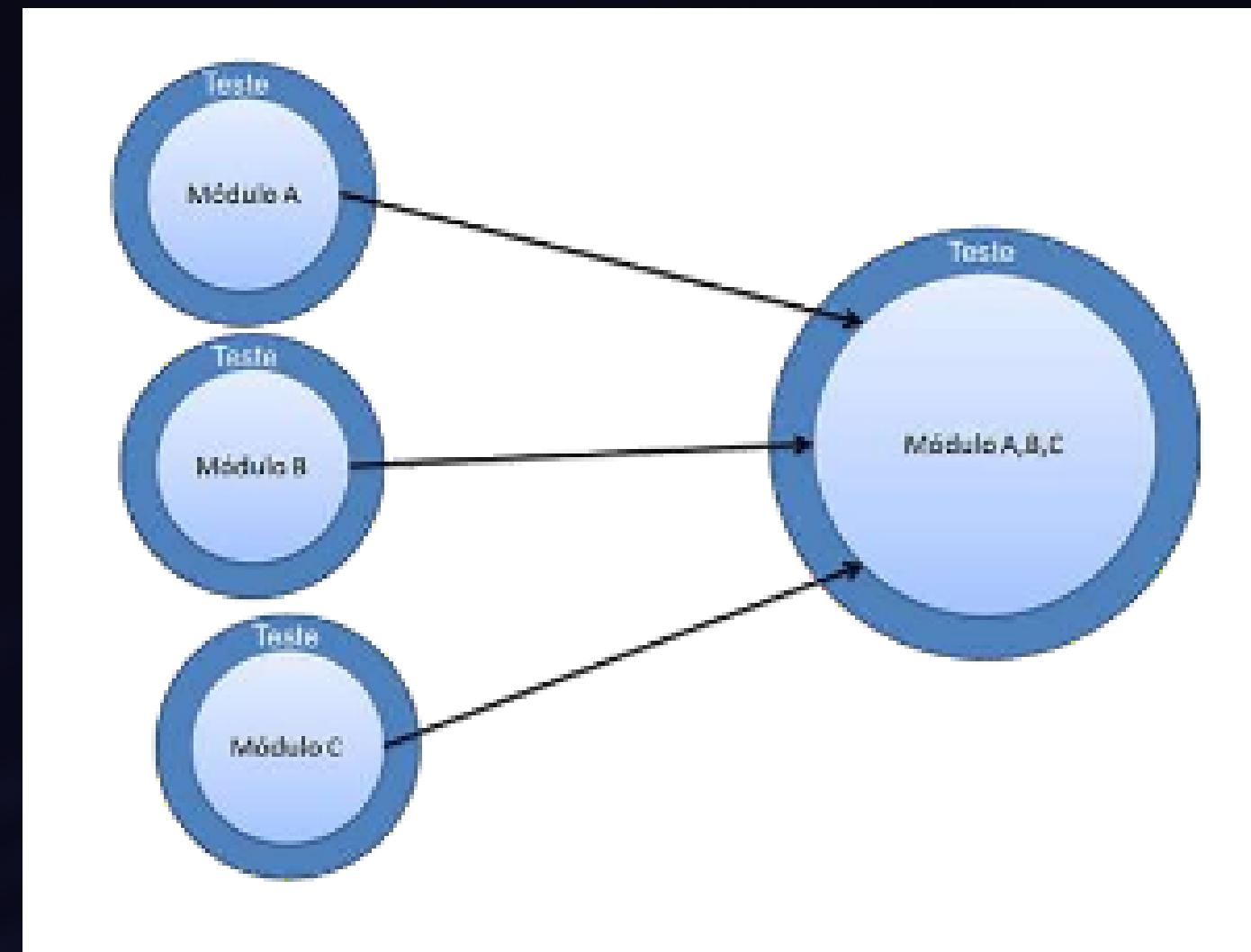
- Aumenta a confiança na alteração/manutenção do código.
- Códigos são mais reutilizáveis.
- Desenvolvimento é mais rápido.
- O custo, ao detectar algum defeito, em comparação com os outros testes de software
- é menor.
- Debugar é mais fácil.
- Os códigos são mais confiáveis.

# Teste Unitário - Desvantagens

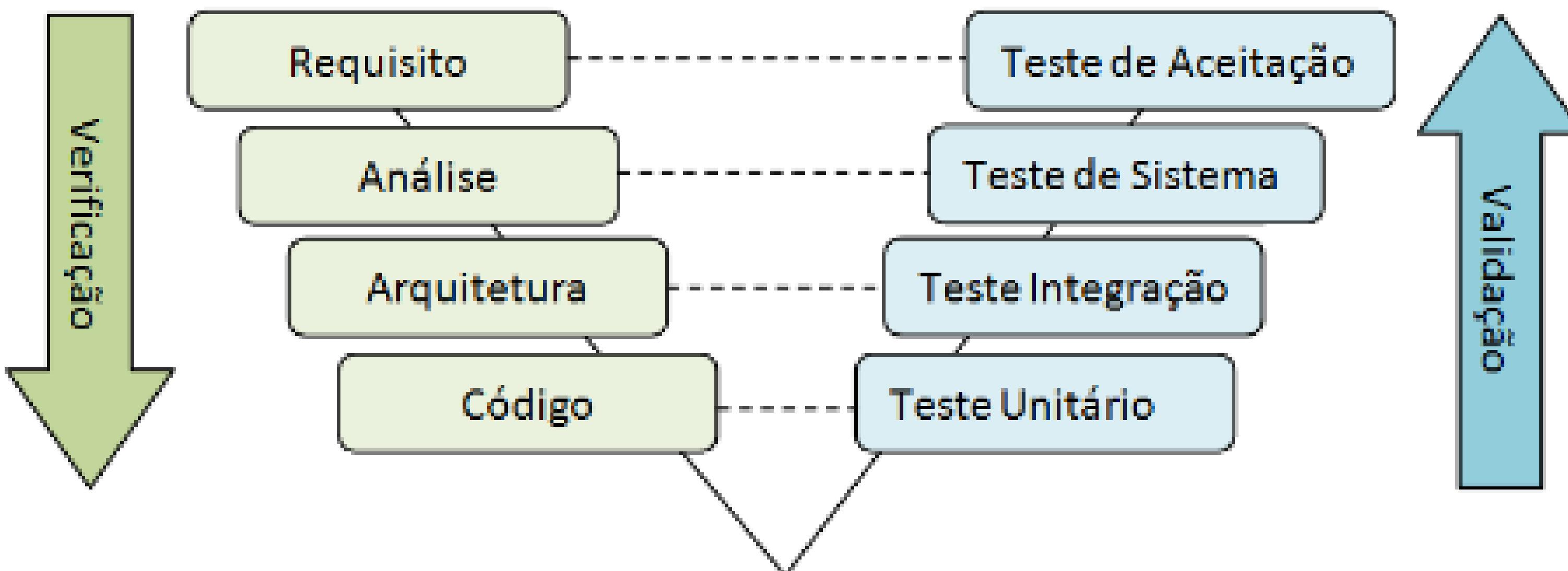
- Não é possível verificar todos os caminhos de execução em todos os aplicativos de *software*.
- Não é possível localizar todos os erros de um aplicativo.
- Existe um limite para a quantidade de cenários e dados de testes que pode ser utilizado pelo fornecedor para verificar o código-fonte.

# Teste de Integração

- O teste de integração surge a partir do momento que ocorrem problemas específicos da integração entre as unidades de um software.
- Esse tipo de teste tem como objetivo garantir que as unidades funcionem juntas.



# Teste de Integração



# Tipos de Teste de Integração

- Integração Bottom-up
  - Essa estratégia começa construir o programa de baixo para cima, construindo primeiro suas dependências e depois realizando testes individualmente sem a necessidade de implementar o programa completo.
- Integração Top-down
  - Diferente do bottom-up, essa estratégia começa de cima pra baixo, onde a importância é rodar o programa sem ter suas reais implementação das dependências.

# Teste de Integração - Vantagens

## Integração Top-down

- Permite que seja feita uma verificação antecipada dos níveis superiores.
- Os módulos podem ser adicionados a cada passo sempre que necessário.

## Integração bottom-up

- A integração bottom-up é mais simples de formular os dados de entrada
- Faz com que seja possível ter uma prévia de como os níveis abaixo se comportam.

# Teste de Integração - Desvantagens

## Integração Top-down

- Os responsáveis por executar os testes não conseguem visualizar as funções do sistema já integradas.

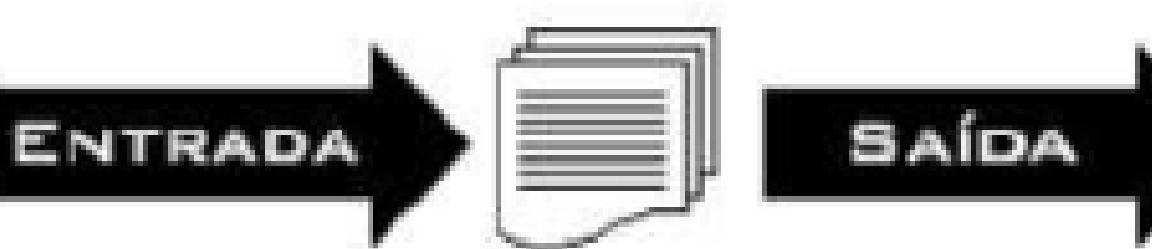
## Integração bottom-up

- Quem fica responsável por realizar os testes não consegue visualizar o nível do sistema até que a última parte do sistema esteja integrada.

# Teste de Caixa Branca

- É um teste de software que realiza testes utilizando código fonte.

## Caixa-branca (Estrutural)

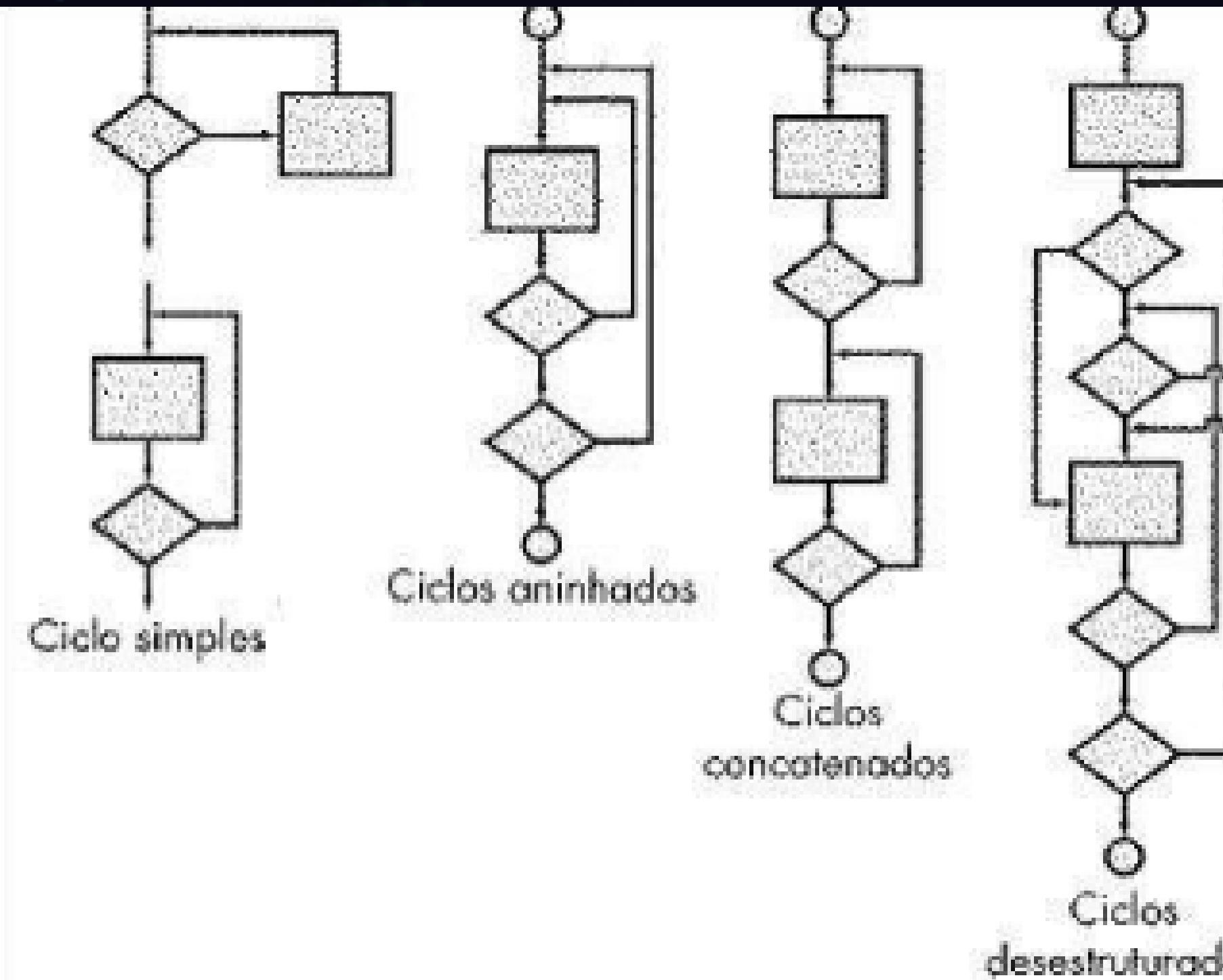


# Técnicas de Caixa Branca

1. Teste do Caminho Básico.
2. Teste de Estrutura do Controle.
3. Teste de Condição.
4. Teste de Ciclo.



# Testes de Ciclo



# 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 directory tree, look for them in the
25  # 'support' directory under the root 'spec' directory.
26  #
27  # in _spec.rb will both be required by default.
28  #
29  # You can also specify optional dependencies:
30  #
31  # require 'factory_girl_rails'
```