# ACH 2028
# Qualidade de Software

## Aula 05 - Teste de Software
## Conceitos Básicos e Teste Funcional

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Escola de Artes, Ciências e Humanidades - Universidade de São Paulo

# Introdução

Como garantir que o código que eu escrevi está correto?

Ou melhor: como encontrar falhas no meu código?

**Solução: Teste de Software**

# Conceito de Teste de Software

*O **Teste de Software** consiste em executar um programa com o objetivo de revelar uma falha* (Myers, 1979)

Uma falha é qualquer evento do sistema que viola um objetivo de qualidade estabelecido

O teste de software não consegue provar a ausência de defeitos

# Exemplo bem simples de teste de software

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
if (resultadoEsperado == resultado)
    print (“resultado correto”)
else
    print (“resultado incorreto”)
```

# Exemplo bem simples de teste de software

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
if (resultadoEsperado == resultado)
    print (“resultado correto”)
else
    print (“resultado incorreto”)
```

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
assertEquals(resultadoEsperado,
resultado)
```

# Conceito de Teste de Software

Exemplos típicos de falhas (failures)

- Crash
- *Runtime error*
- Resultado errado
- Tempo de resposta excedido
- Formato de saída fora do padrão

# Alguns tipos de falhas

Visuais

- problemas de alinhamento de componentes
- sobreposição de componentes
- texto impossível de ler

# Alguns tipos de falhas

Funcionais

- O usuário não consegue fazer login
- O usuário não consegue fazer pagamento com dois cartões
- Não é possível atualizar o número de itens no carrinho de compras
- O usuário não consegue escolher outro endereço de entrega
- ...

# Alguns tipos de falhas

Não-funcionais

- Desempenho: a página demora 30 segundos para carregar
- Segurança: a senha digitada não aparece ofuscada

# Terminologia

Erro → Defeito → Falha

- **Erro**: item de informação ou estado inconsistente / engano do desenvolvedor
- **Defeito**: deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha
- **Falha**: evento notável em que o sistema viola suas especificações

# Quando começar a testar?

O quanto antes a atividade de teste de software for introduzida no processo, melhor

Uma falha encontrada na etapa de desenvolvimento/teste tem o custo de localizar e corrigir o defeito, e executar novamente os testes

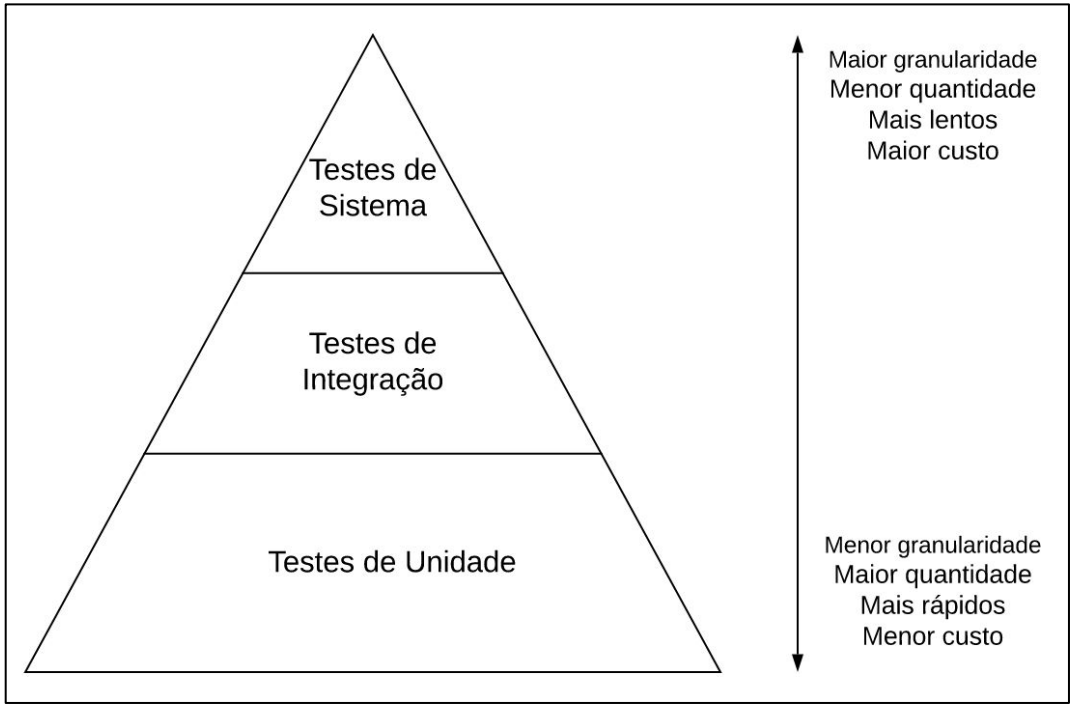
Uma falha encontrada em um software operação tem o custo de *help desk*, escalada dos problemas, alocação de equipe, localizar e corrigir o defeito, executar novos testes e fazer novamente o *deploy* da aplicação corrigida, sem contar a insatisfação do cliente

# Quando começar a testar?

Existem diversos tipos ou fases de teste:

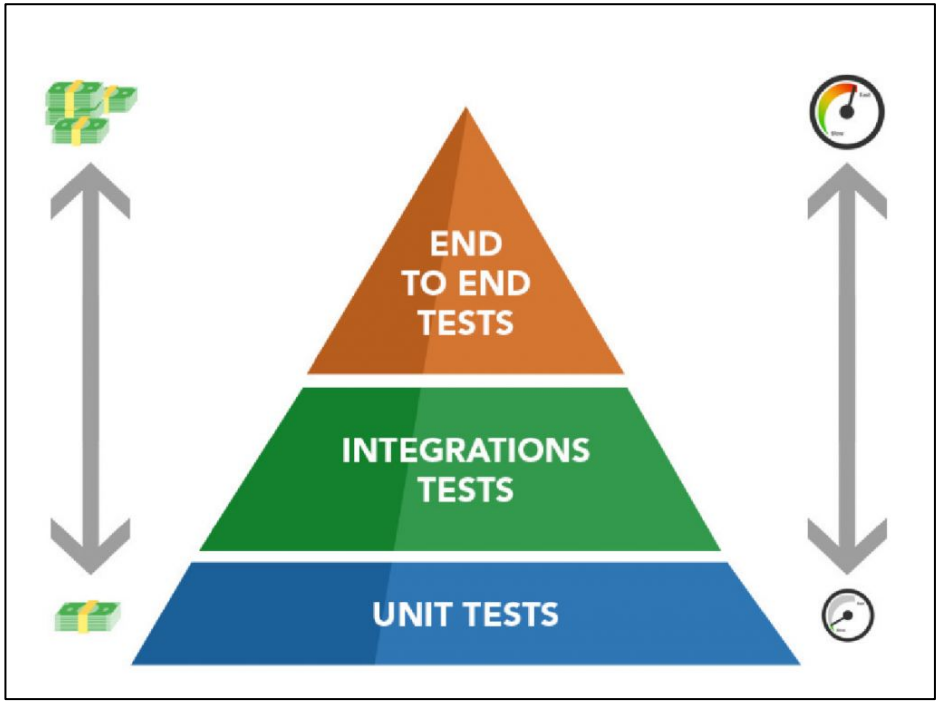
- Teste de Unidade (ou teste de desenvolvedor)
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação (e.g. end to end)
- Teste de Operação
- Teste de Regressão

# Pirâmide de testes



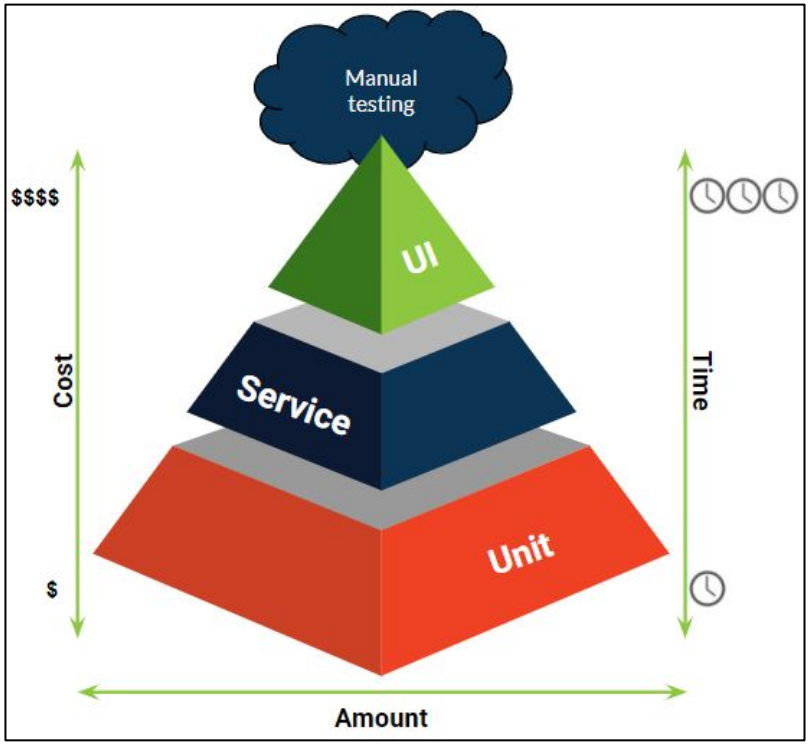

https://engsoftmoderna.info/cap8.html

# Pirâmide de testes



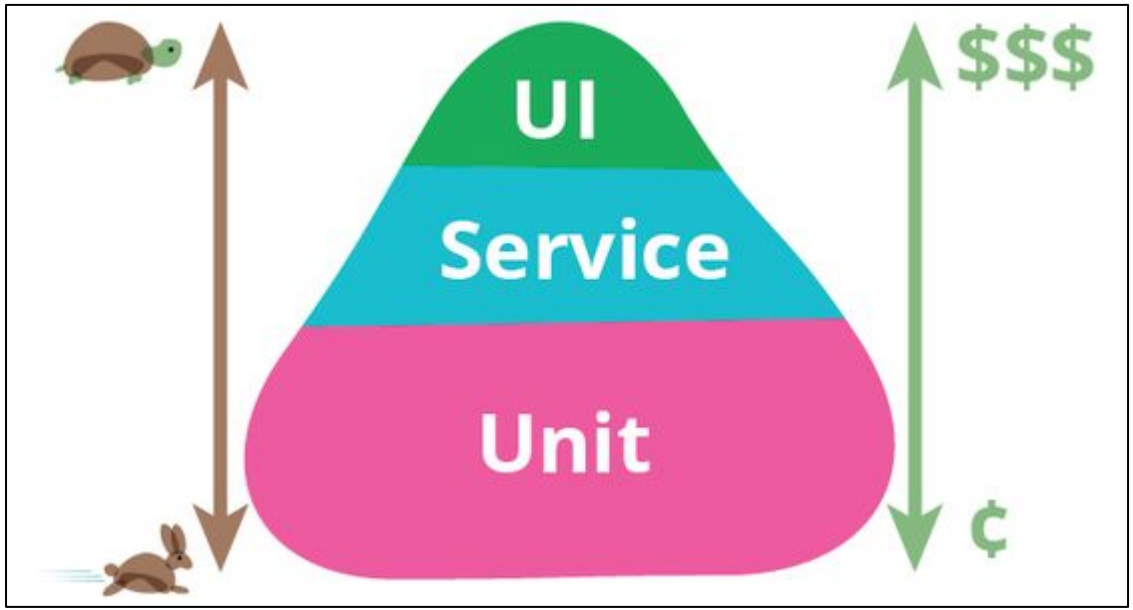

https://eximia.co/que-tipo-de-teste-escrever-para-reduzir-o-custo-total-de-um-projeto-de-software/

# Pirâmide de testes




https://www.zup.com.br/blog/fases-de-teste-tipos-de-teste

# Pirâmide de testes




https://martinfowler.com/bliki/TestPyramid.html

# Quando começar a testar?

Existem diversos tipos ou fases de teste:

- **Teste de Unidade (ou teste de desenvolvedor)**
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação (e.g. end to end)
- Teste de Operação
- Teste de Regressão

# Teste de Unidade

Também chamados de testes do desenvolvedor, ou micro-testes

São escritos para testar as menores unidades de desenvolvimento (método ou classe, por exemplo)

# Teste de Unidade

Também chamados de testes do desenvolvedor, ou micro-testes

São escritos para testar as menores unidades de desenvolvimento (método ou classe, por exemplo)

```
@Test
  public void testEmptyStack() {
    Stack<Integer> stack = new Stack<Integer>();
    boolean empty = stack.isEmpty();
    assertTrue(empty);
  }
```

# Teste de Unidade

Classicamente, os testes considerados de unidade são aqueles que não dependem de nenhuma outra unidade (por implementação, ou por uso de dublês de teste).

Ex: se estou testando um método, ele não pode chamar nenhum outro método (a não ser que seja um dublê)

Entretanto, essa é só uma questão de nomenclatura. No “mundo real”, os testes são classificados de acordo com questões mais práticas

# Teste de Unidade

Os testes de unidade devem:

- Ser escritos em grande quantidade (a maior parte dos testes de um software devem ser de unidade - pirâmide de testes)
- Rodar rapidamente (pois são executados muitas vezes durante o desenvolvimento)
- Ajudar a localizar defeitos com grande precisão (afinal, os testes são de partes muito pequenas do código)

# Teste de Unidade

É importante que os testes de unidade rodem rapidamente e sejam muitos porque eles são a base para o teste de regressão e refatoração

Após refatorar o código é importante executar casos de testes para verificar se a alteração no código não alterou seu comportamento

# Teste de Unidade

Os testes de unidade não devem, em sua execução:

- se comunicar com uma base de dados
- utilizar sistemas de arquivo
- se comunicar com alguma função na rede
- alterar configurações do ambiente

Esses são testes válidos, mas geralmente são testes de integração que vão levar mais tempo para serem executados.

# Teste de Unidade

Nos casos em que se deseja testar uma unidade que precisa se comunicar com outros módulos ou sistemas, a alternativa é usar os famosos dublês de teste:

- **Mock:** A ideia por trás dos mocks é de abstrair a lógica da classe dependente, ao mesmo tempo que garante que as interações ocorrem dentro do esperado.
- **Stub:** fornecem respostas prontas para as chamadas feitas durante o teste, geralmente não responde a nada fora do que está programado.
- **Fake:** Objetos que possuem implementação, porém com o objetivo de diminuir a complexidade e/ou tempo de execução de alguns processos para acelerar os testes, porém nunca usado em produção.

https://medium.com/@felippeduarte/dubl%C3%AAs-de-teste-mock-stub-dummy-fake-e-spy-80a5cb5fc3c2

# Teste de Integração

Testes que envolvem a execução de funções que exigem a integração entre mais de uma unidade/módulo/sistema

Na prática, o teste de integração vai ser aquele que vai testar funções que precisam acessar o banco de dados, escrever/ler dados para/de um arquivo, vão invocar um serviço na rede, etc, sem usar dublês de teste

Esses testes são mais lentos do que os testes de unidade

# Teste de Sistema

Envolve o teste do software como um todo (considerando suas operações globais, geralmente acessadas via interface gráfica ou API)

Podem ser chamados também de testes end-to-end

Podem envolver o teste de propriedades como:

- Desempenho
- Segurança
- Disponibilidade
- ...

# Test end-to-end

Como o próprio nome diz, tem o objetivo de testar uma interação de ponta a ponta em um software

Ex: testar todo o processo de um aluguel de carro:

- Usuário faz login
- Usuário faz uma busca pela categoria de carro desejado
- Usuário seleciona o carro desejado e adiciona no carrinho
- Usuário finaliza a compra
- Usuário faz o pagamento e recebe notificação

# Outros tipos de teste

Teste de aceitação:

- Às vezes também chamado de teste de sistema (tudo questão de nomenclatura)
- Teste que geralmente envolve a participação de clientes/usuários para “aceitar” o sistema (ou seja, implementa o que foi proposto)
- Baseado em regras de negócio/requisitos
- Pode também se referir a teste end-to-end

# Outros tipos de teste

Testes de regressão:

- Conjunto de testes executados para verificar se mudanças feitas no software introduziram algum defeito
- Dão segurança para a refatoração, correção e evolução do software
- Podem incluir testes de unidade, de integração e de sistema
- Quanto mais rápido executar, melhor

# Outros tipos de teste

Testes de operação:

- Testes realizados pelos próprios usuários
- Alfa: realizados no ambiente da organização que desenvolve
- Beta: realizados no ambiente dos usuários

# Forma de execução de testes

Testes manuais

Testes automatizados

# Forma de execução de testes

Testes manuais

**Testes automatizados:**

- Facilitam a reexecução dos testes

# Automação de testes

Ferramentas e Frameworks de teste

- Atendem a diversos tipos de teste (unidade, integração, sistema, end-to-end)
- Dependem da linguagem de programação, plataforma e tipo de teste

# Teste de unidade/integração

Framework JUnit (Java)

Método: classificaTriangulo(int LA, int LB, int LC) throws LadoInvalidoException

# Teste de unidade/integração

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class TrianguloTest {
...
}
```

# Teste de unidade/integração

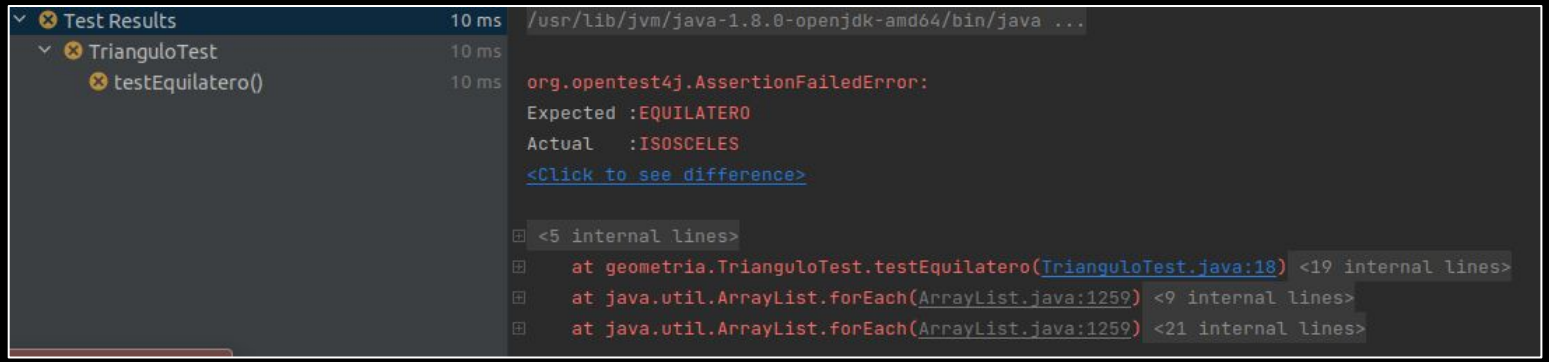
```java
@Test
public void testEquilatero() throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO",result);
}
```

# Teste de unidade/integração

```java
@Test
public void testEquilatero() throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO",result);
}
```

# Teste de unidade/integração

```java
@Test
public void testEquilatero() throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO",result);
}
```



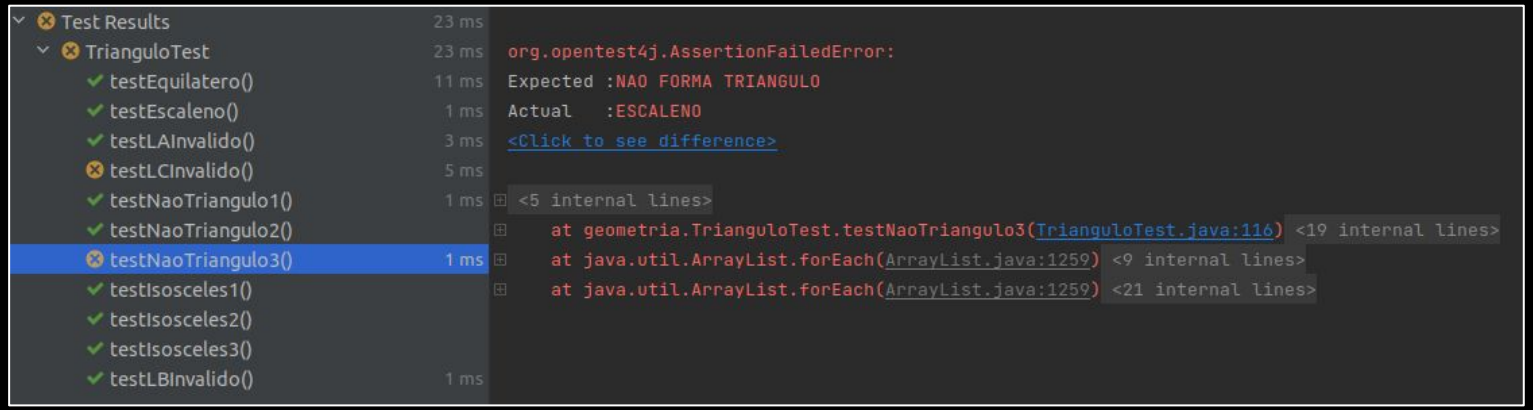
# Teste de unidade/integração

```java
@Test
public void testIsosceles1() throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 6;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("ISOSCELES",result);
}
```

# Teste de unidade/integração

```java
@Test
public void testLAInvalido() throws LadoInvalidoException{
    int LA = -2;
    int LB= 4;
    int LC = 5;
    assertThrows(LadoInvalidoException.class, () -> Triangulo.classificaTriangulo(LA, LB, LC));
}
```

# Teste de unidade/integração



# Teste end-to-end

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumTest {

    public static void main(String[] args) {
        // declaration and instantiation of objects/variables
        System.setProperty("webdriver.chrome.driver", "../ChromeDriver/chromedriver");
        WebDriver driver= new ChromeDriver();
        // Launch website
        driver.navigate().to("http://www.google.com/");
        // Click on the search text box and send value
        driver.findElement(By.id("lst-ib")).sendKeys("javatpoint tutorials");
        // Click on the search button
        driver.findElement(By.name("btnK")).click();
    }
}
```

https://www.javatpoint.com/selenium-webdriver-first-test-case

# Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Wait Until Page Contains Element  edit_text_username
 Input Text   edit_text_username  "User12345"
 Input Text   edit_text_passwd    "Mypassword12345"
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_search
 Click Element  btn_search
 Page Should Contain Text  "Micro USB Charger Cable"
 Click Text  "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_buy
 Click Element  btn_buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
 ...
```

# Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Log User
 Search for product
 Buy product

*** Keywords ***
Log User
 Wait Until Page Contains Element  edit text username
 Input Text   edit text username  "User12345"
 Input Text   edit_text_passwd    "Mypassword12345"
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login

Search for product
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_search
 Click Element  btn search
 Page Should Contain Text  "Micro USB Charger Cable"

By product
 Click Text  "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_buy
 Click Element  btn buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
```

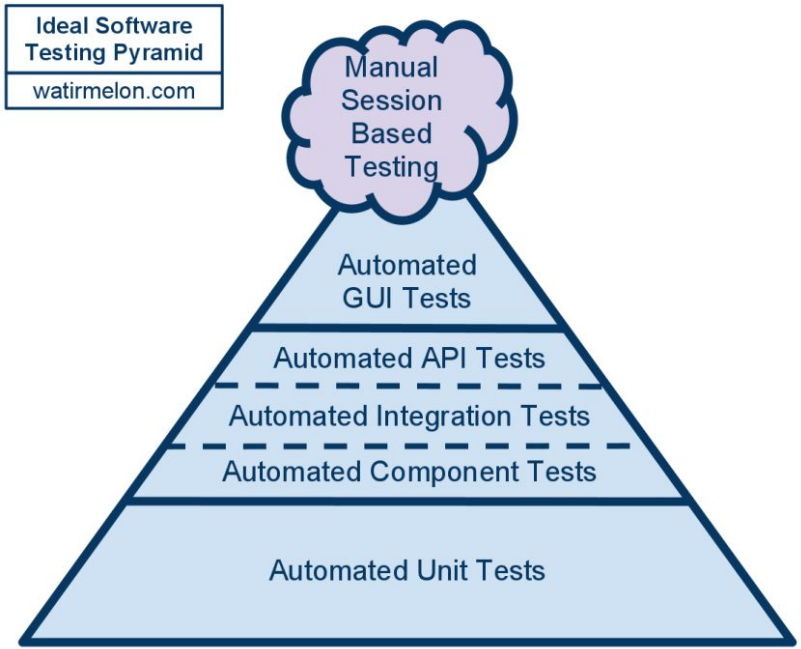
# Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Log User  "User12345" "Mypassword12345"
 Search for product  "Micro USB Charger Cable"
 Buy product  "Micro USB Charger Cable"

*** Keywords ***
Log User  {$username}  {$password}
 Wait Until Page Contains Element  edit text username
 Input Text   edit text username  {$username}
 Input Text   edit_text_passwd    {$password}}
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login

Search for product  {$product}
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search {$product}
 Wait Until Page Contains Element  btn_search
 Click Element  btn search
 Page Should Contain Text  {$product}

By product  {$product}
 Click Text  {$product}
 Wait Until Page Contains Element  btn_buy
 Click Element  btn buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
```

# Automação de testes

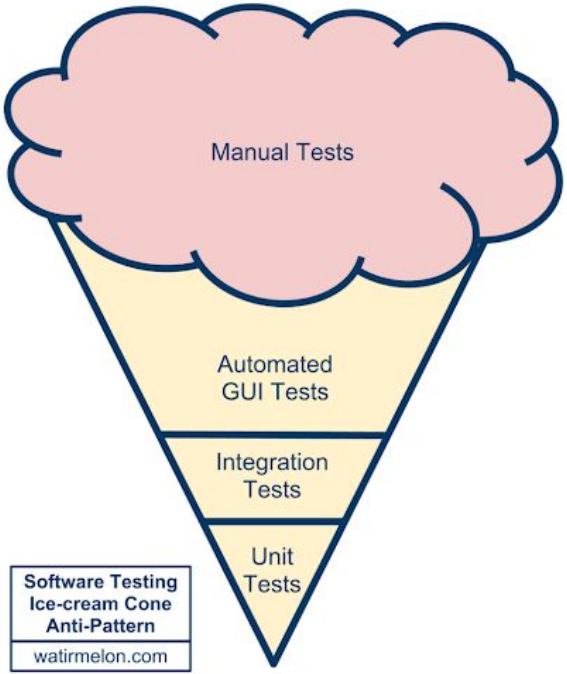
É preciso estudar cada framework/ferramenta de testes para entender os detalhes sobre como automatizar os testes que se pretende criar

# Cuidado com o cone de teste (é comum, mas perigoso)




IDEAL

**VS**




MUITO COMUM

# Cuidado com o cone de teste (é comum, mas perigoso)

Testes de unidade ajudam a achar o defeito mais rapidamente (teste localizado)

Os testes end-to-end revelam falhas cujos defeitos associados são mais difíceis de localizar a origem

Portanto, invistam em muitos testes de unidade

# Leituras recomendadas

Leitura recomendada:

- **Autor**: Ham Vocke
- The Practical Test Pyramid (https://martinfowler.com/articles/practical-test-pyramid.html)

# Leituras recomendadas

Leitura recomendada:

- **Livro**: Engenharia de Software Moderna - Princípios e Práticas para Desenvolvimento de Software com Produtividade
- **Autor**: Marco Tulio Valente
- Capítulo 8 - Testes (https://engsoftmoderna.info/cap8.html)