
# TestNG

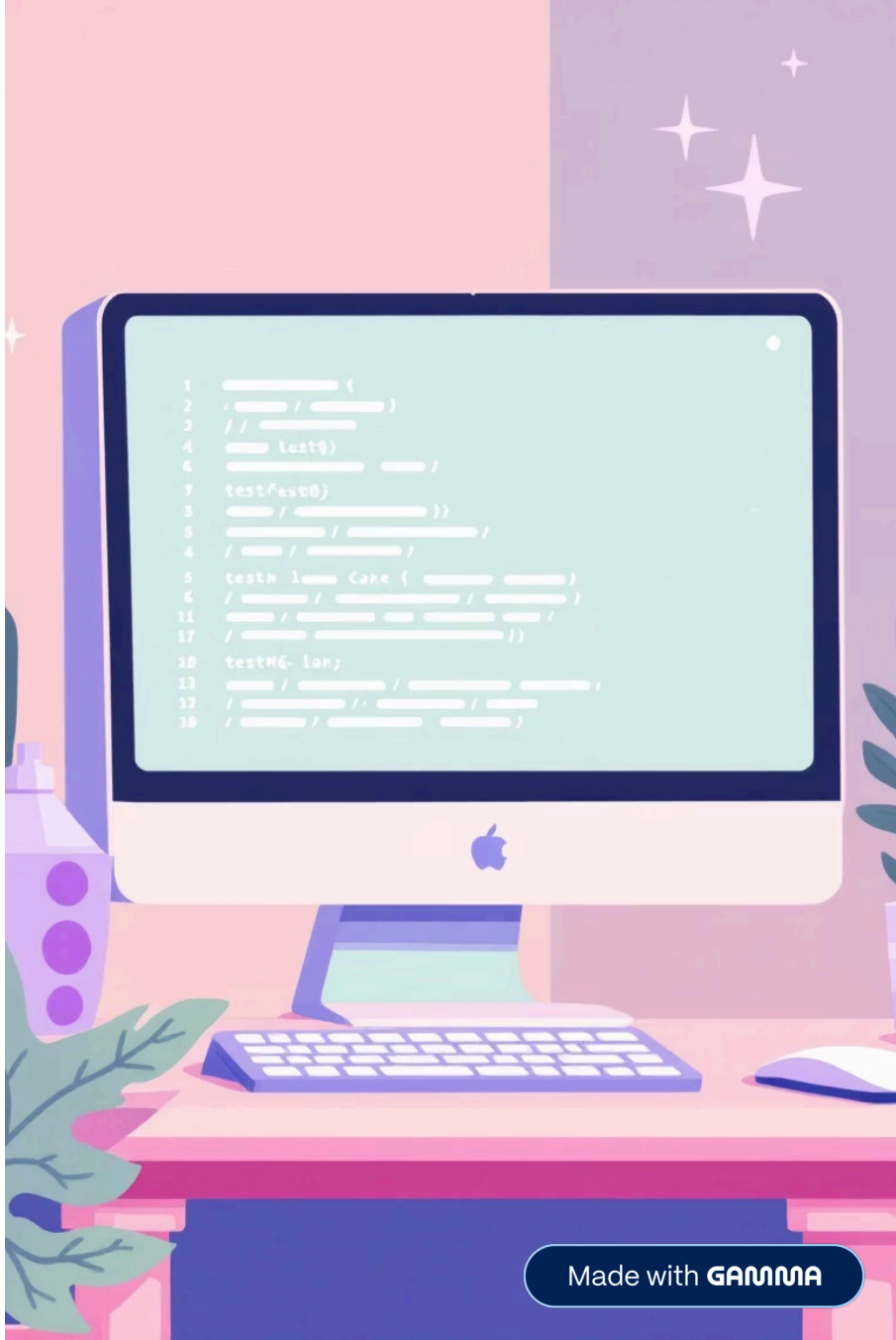
Lana e Larissa

# O que é?

É Framework de testes Java de código aberto para automação, construído com base no JUnit e NUnit.

Oferece recursos avançados para definir dependências, anotações, agrupamentos e sequenciamento de testes.

**Curiosidade:** O NG vem de "Next Generation", para dar essa ideia que ele é a evolução do JUnit.



# Recursos Benéficos do TestNG

## Suporte para Anotações

Sistema de anotações para controle de execução

## Parametrização

Execução de testes com diferentes conjuntos de dados

## Configuração XML

Configuração flexível via arquivo testng.xml

## Gerenciamento de Dependências

Controle de ordem e dependências entre testes

## Execução Paralela

Execução simultânea para otimizar tempo

## Relatórios HTML

Relatórios detalhados e visuais integrados

# TestNG vs JUnit

Os dois principais frameworks de teste Java que se integram ao Selenium

# Diferenças Fundamentais

JUnit

- Suporta apenas testes unitários
- Sem anotações avançadas como @BeforeGroups
- Não suporta dependências entre casos de teste

TestNG

- Suporta testes unitários, funcionais, integração e ponta a ponta
- Suporta anotações avançadas como @BeforeGroups e @AfterGroups
- Suporta dependências usando dependsOn e dependsOnGroup

# Recursos Avançados do TestNG

### Agrupamento de Testes

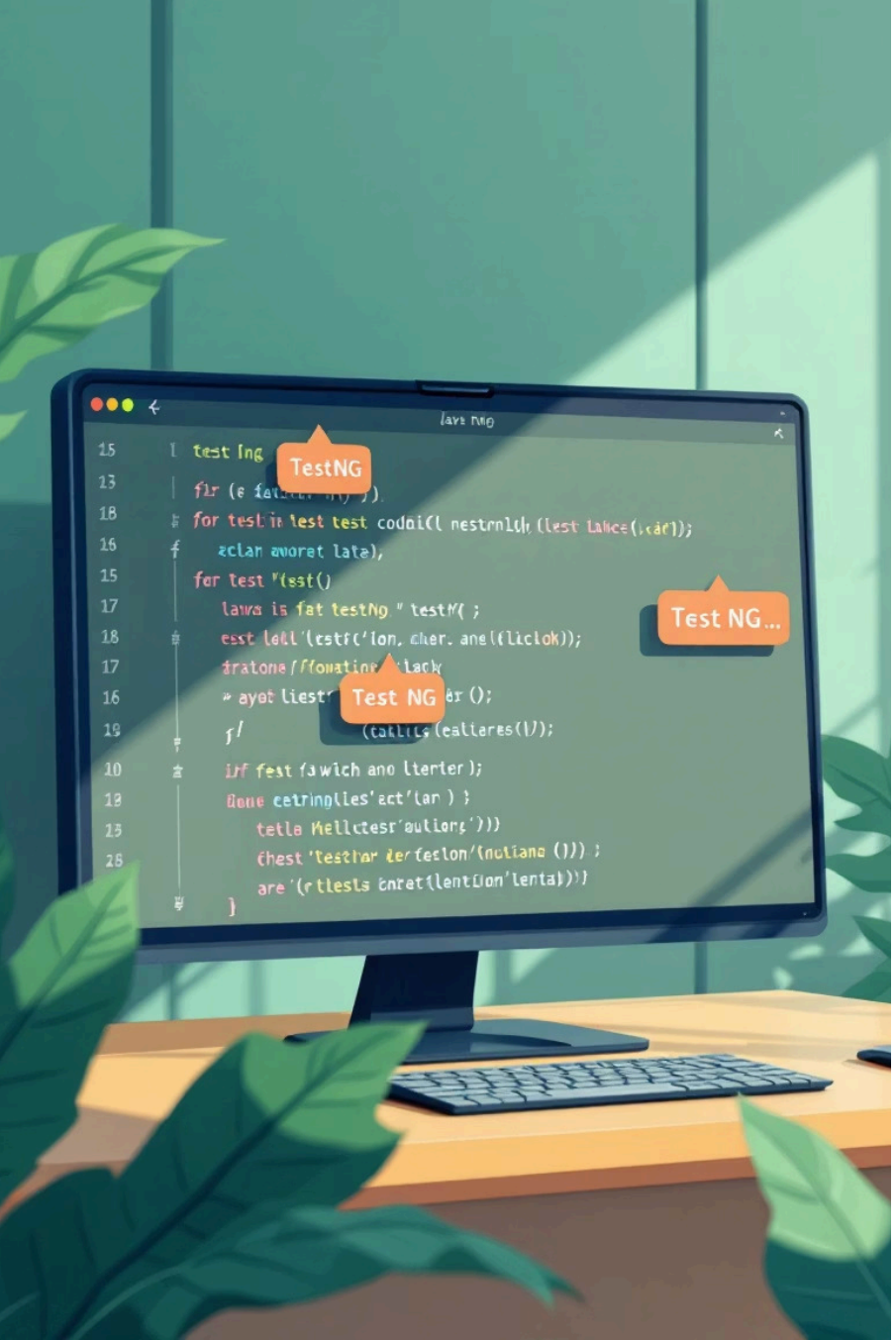
TestNG suporta agrupamento e execução conjunta de casos de teste, enquanto JUnit não oferece essa funcionalidade.

### Relatórios Integrados

Possui relatórios HTML integrados, enquanto JUnit precisa ser integrado ao Maven para gerar relatórios.

### Execução Paralela Avançada

Suporte integrado para execução paralela em múltiplos níveis: métodos, classes e testes.




# Anotações TestNG

As anotações TestNG fornecem controle sobre a execução dos testes, definindo comportamento e configuração. Utilizam o "@" seguido do nome da anotação.

# Suite de Teste no TestNG

No TestNG, uma **suíte** é um conjunto de testes que podemos agrupar e rodar juntos.
Isso é útil quando temos várias classes de teste e queremos organizar melhor a execução.

Coleção de Testes

Orquestração e Controle

Configuração Centralizada

# Suite

### @BeforeSuite

Executada uma única vez antes de todos os testes da suíte serem executados.

### @AfterSuite

Executada uma única vez depois que toda a suíte de testes terminou.

```java
@BeforeSuite
public void beforeSuite() {
    System.out.println("Início da Suíte de Testes do Triângulo");
}

@AfterSuite
public void afterSuite() {
    System.out.println("Fim da Suíte de Testes do Triângulo");
}
```

# Class

### @BeforeClass

Executa **uma vez antes** de todos os testes de uma classe.

### @AfterSuite

Executa **uma vez depois** que todos os testes de **uma classe** terminaram.

```java
@BeforeClass
public void beforeClass() {
    System.out.println("Preparando a classe de testes");
}

@AfterClass
public void afterClass() {
    System.out.println("Finalizando a classe de testes");
}
```

# Method

@BeforeMethod

Executa **antes de cada método de teste**.

@AfterMethod

Executa **depois de cada método de teste**.

```java
@BeforeClass
public void beforeClass() {
    System.out.println("Preparando a classe de testes");
}

@AfterClass
public void afterClass() {
    System.out.println("Finalizando a classe de testes");
}
```

# Estrutura do Arquivo testng.xml

```xml
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SuiteTriangulo">
    <test name="TestesBasicosTriangulo">
        <classes>
            <class name="example.TrianguloTest"/>
        </classes>
    </test>
</suite>
```

# Resumo do nosso código

`Triangulo.java` → recebe três lados e classifica como **Equilátero, Isósceles, Escaleno, Não é triângulo** ou **Lados inválidos**.

`TrianguloTest.java` → Usa as notações `@BeforeSuite` / `@AfterSuite` , `@BeforeClass` / `@AfterClass`, `@BeforeMethod` / `@AfterMethod`, `@Test`

**Arquivo** `testng.xml` → Permite rodar todos os testes juntos e organizar grupos.

**Maven**:

- Gerencia as dependências (TestNG).
- Automatiza o ciclo de build e testes com `mvn test`.

# Case de Sucesso

Automação de Testes com TestNG: Um Case de Sucesso na Redução do Ciclo de Regressão em 95%


# Stag Software

STAG Software para um cliente global do setor de tecnologia de simulação e serviços de engenharia.

O projeto de automação de testes de ponta a ponta (end-to-end) resultou em uma impressionante redução de 95% no tempo do ciclo de testes de regressão, um ganho de eficiência que demonstra o poder do TestNG quando combinado com outras ferramentas de automação.

# O desafio:

O cliente enfrentava um ciclo de testes de regressão manual que consumia 100 horas, um processo lento, propenso a erros e que se tornava um gargalo no ciclo de desenvolvimento de software. A necessidade era clara: automatizar o processo para acelerar a entrega, melhorar a qualidade e otimizar a alocação de recursos da equipe de testes.

# A Solução (com TestNG e Selenium):

A STAG Software desenvolveu uma robusta estrutura de automação utilizando Selenium WebDriver para a interação com os navegadores e o TestNG como o framework de execução e gerenciamento dos testes. A escolha do TestNG foi estratégica por suas funcionalidades avançadas, que foram cruciais para o sucesso do projeto:

**Execução Paralela**

**Gerenciamento e Organização dos Testes**

**Geração de Relatórios Detalhados**

**Integração Contínua (CI)**

# Resultados alcançados:

**Redução do Ciclo de Testes:**

→ De 100h para 4h

→ Redução de 95

**Aceleração do Time-to-Market:** Com a identificação de bugs no início do desenvolvimento, o ciclo de lançamento de novas versões acelerou.

**Aumento da Qualidade do Produto:**

A automação garantiu uma cobertura de testes mais consistente e abrangente.

**Melhoria na Eficiência da Equipe:**

Com a automação, a equipe de testes pôde se dedicar a atividades de maior valor, como a criação de novos cenários de teste e a análise de resultados.