WALL-E NA TERRA DOS BUGS

COMO OS TESTES DE SOFTWARE PODEM SALVAR O DIA

Aprenda o que são Testes de Software e como eles podem te ajudar no seu dia-a-dia como programador

Kássia Almeida Moura

O QUE SÃO TESTES DE SOFTWARE?

Os testes de software são um componente essencial do desenvolvimento de sistemas e aplicações. Eles envolvem a execução de um programa ou aplicação com a intenção de encontrar erros ou falhas. O principal objetivo dos testes de software é garantir que o produto final atenda aos requisitos especificados e funcione corretamente em diversos cenários.

Existem diversos tipos de testes de software, cada um com um propósito específico para garantir a qualidade e a funcionalidade do software.

Aqui estão alguns dos principais tipos de testes de software:

<u>Testes Funcionais</u>

- ***Teste Unitário:*** Verifica se unidades individuais do código (geralmente funções ou métodos) funcionam corretamente.
- ***Teste de Integração:*** Avalia se diferentes módulos ou serviços de um sistema interagem corretamente entre si.
- ***Teste de Sistema:*** Testa o sistema como um todo para garantir que ele atenda aos requisitos especificados.
- ***Teste de Aceitação***: Verifica se o sistema atende aos critérios de aceitação e é geralmente realizado pelo cliente ou usuário final.
- ***Teste de Regressão:*** Assegura que novas mudanças no código não introduzam erros em partes já testadas do sistema.
- ***Teste de Interface do Usuário (UI):*** Testa a interface do usuário para garantir que ela funcione conforme esperado.

Testes Não Funcionais

- ***Teste de Desempenho:*** Avalia a velocidade, capacidade de resposta e estabilidade do sistema sob uma carga de trabalho específica: Teste de Carga, Teste de Estresse, : Teste de Volume e Teste de Capacidade.
- ***Teste de Segurança:*** Avalia a capacidade do sistema de proteger dados e manter a funcionalidade conforme esperado, mesmo sob ataques maliciosos.
- ***Teste de Usabilidade:*** Verifica se o sistema é fácil de usar e intuitivo para o usuário final.
- ***Teste de Compatibilidade:*** Garante que o sistema funcione em diferentes dispositivos, sistemas operacionais, navegadores e outras plataformas.
- ***Teste de Confiabilidade:*** Avalia a capacidade do sistema de funcionar de forma consistente e correta durante um período prolongado.
- ***Teste de Manutenibilidade***: Verifica a facilidade de manutenção do sistema, incluindo a identificação de problemas e a implementação de correções.
- ***Teste de Portabilidade:*** Avalia a capacidade do sistema de ser transferido de um ambiente para outro.

Testing Software

Testes Baseados em Estrutura

- ***Teste de Caixa Branca:*** Testa a estrutura interna do código para garantir que todos os caminhos lógicos e condições sejam verificados.
- ***Teste de Caixa Preta:*** Avalia a funcionalidade do software sem considerar sua estrutura interna, focando apenas nas entradas e saídas.
- ***Teste de Caixa Cinza:*** Combina técnicas de teste de caixa branca e caixa preta.

Testes Automatizados

- **Teste Automatizado**: Utiliza scripts e ferramentas automatizadas para executar testes que verificam a funcionalidade, desempenho e outras características do software.
- **Teste de Regressão Automatizado**: Automação de testes de regressão para garantir que as alterações no código não introduzam novos bugs.

Testing Software

<u>Outros Tipos de Teste</u>

- **Teste Alpha**: Realizado pelos desenvolvedores ou por um grupo de usuários internos antes do lançamento do produto.
- **Teste Beta**: Executado por um grupo limitado de usuários finais após o teste alpha, mas antes do lançamento oficial.
- **Teste de Ad-hoc**: Testes não planejados, realizados sem um roteiro específico para encontrar falhas que não foram antecipadas.

Neste ebook vamos explorar mais detalhadamente alguns tipos de teste de software e fornecerei alguns exemplos de como eles podem ser executados, utilizando a linguagem C#, embora possam ser aplicados em outras linguagens também.

Testing Software

01 Testes Unitários

Os testes unitários verificam a menor parte testável do código, como funções ou métodos, garantindo que cada unidade funcione corretamente de forma isolada.

Exemplo de Teste Unitário

Vamos considerar uma função simples em C# que soma dois números:

```C#
public int Soma(int a, int b)
{
    return a + b;
}
```

Um teste unitário para essa função pode ser escrito usando a biblioteca NUnit:

```C#
using NUnit.Framework;

[TestFixture]
public class TestSoma
{
    [Test]
    public void TestaSoma()
    {
        Assert.AreEqual(5, Soma(2, 3));
        Assert.AreEqual(0, Soma(-1, 1));
        Assert.AreEqual(0, Soma(0, 0));
    }
}
```

Testing Software

02

Testes de Integração

Os testes de integração verificam a interação entre diferentes unidades ou módulos do software, assegurando que eles funcionem corretamente quando combinados.

Exemplo de Teste de Integração

Considere um sistema simples com duas funções: uma que soma e outra que subtrai números.

```C#
public int Soma(int a, int b)
{
    return a + b;
}

public int Subtrai(int a, int b)
{
    return a - b;
}
```

Um teste de integração pode verificar se essas funções funcionam corretamente juntas:

```C#
using NUnit.Framework;

[TestFixture]
public class TestOperacoes
{
    [Test]
    public void TestaOperacoes()
    {
        int resultadoSoma = Soma(2, 3);
        int resultadoSubtrai = Subtrai(resultadoSoma, 1);
        Assert.AreEqual(4, resultadoSubtrai);
    }
}
```

03 Testes Funcionais

Os testes funcionais verificam se o software cumpre seus requisitos funcionais, focando no resultado final da aplicação.

Exemplo de Teste Funcional

Imagine uma função que valida se um número é par ou ímpar:

```C#
public bool EhPar(int numero)
{
    return numero % 2 == 0;
}
```

Um teste funcional pode ser escrito para garantir que a função está cumprindo seu propósito:

```C#
using NUnit.Framework;

[TestFixture]
public class TestParidade
{
    [Test]
    public void TestaEhPar()
    {
        Assert.IsTrue(EhPar(2));
        Assert.IsFalse(EhPar(3));
    }
}
```

04 Testes de Sistema

Os testes de sistema verificam o sistema como um todo, avaliando se o software atende aos requisitos especificados e se comporta corretamente em seu ambiente real.

Exemplo de Teste de Sistema

Considere um sistema de login simples:

C#

```
public class SistemaLogin
{
    private Dictionary<string, string> usuarios = new Dictionary<string, string>
    {
        { "user1", "senha1" },
        { "user2", "senha2" }
    };

    public bool Login(string usuario, string senha)
    {
        return usuarios.ContainsKey(usuario) && usuarios[usuario] == senha;
    }
}
```

Um teste de sistema pode verificar todo o processo de login:

```C#
using NUnit.Framework;

[TestFixture]
public class TestLogin
{
    private SistemaLogin sistemaLogin;

    [SetUp]
    public void Setup()
    {
        sistemaLogin = new SistemaLogin();
    }

    [Test]
    public void TestaLogin()
    {
        Assert.IsTrue(sistemaLogin.Login("user1", "senha1"));
        Assert.IsFalse(sistemaLogin.Login("user1", "senha_errada"));
        Assert.IsFalse(sistemaLogin.Login("user_inexistente", "senha1"));
    }
}
```

Testing Software

05

Testes de Aceitação

Os testes de aceitação são realizados para garantir que o software atenda aos critérios de aceitação e aos requisitos do cliente ou usuário final.

Exemplo de Teste de Aceitação

Vamos considerar um aplicativo que calcula o IMC (Índice de Massa Corporal):

```C#
public double CalcularIMC(double peso, double altura)
{
    return peso / (altura * altura);
}
```

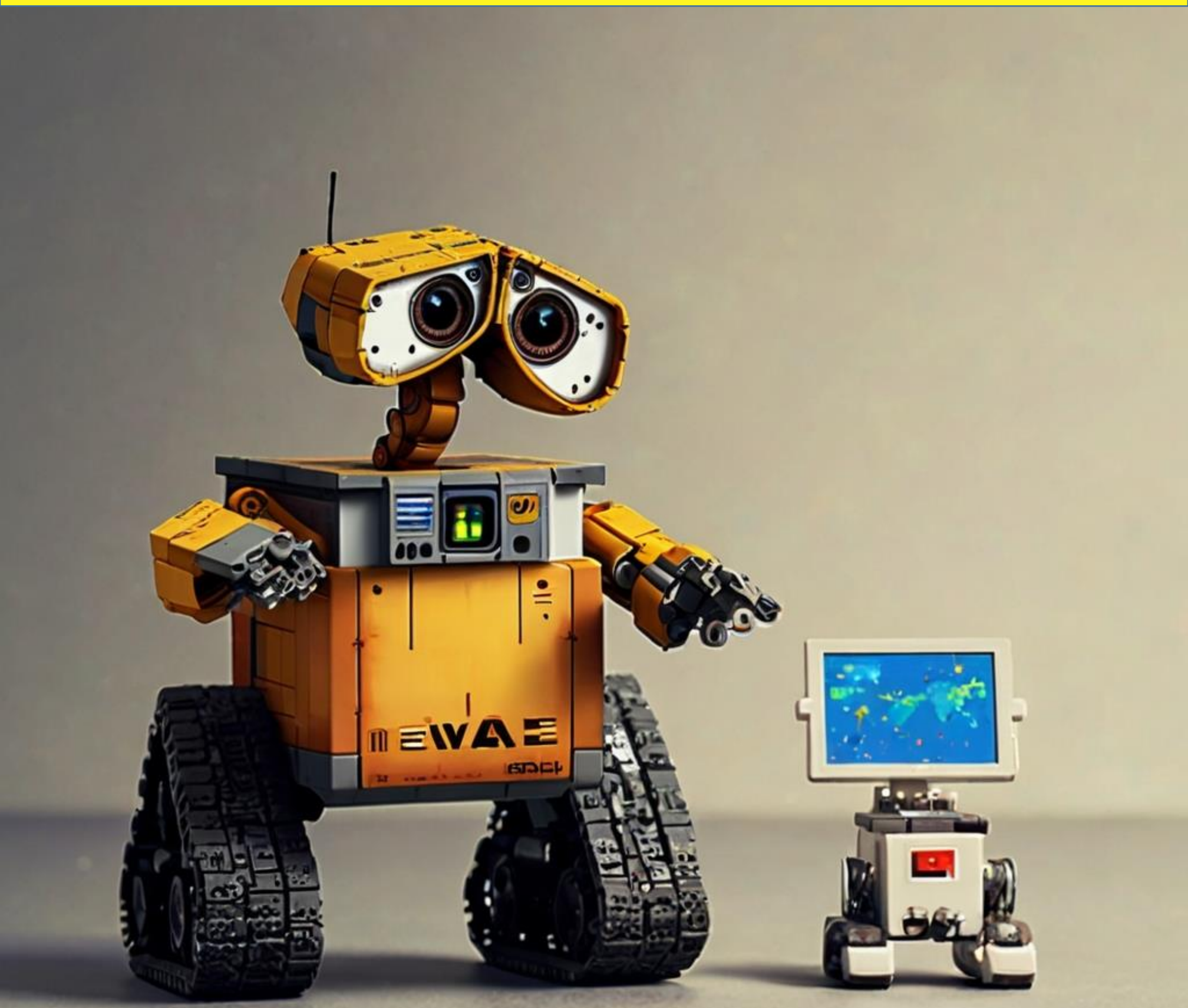
Um teste de aceitação pode ser:

```C#
using NUnit.Framework;

[TestFixture]
public class TestIMC
{
    [Test]
    public void TestaCalcularIMC()
    {
        double imc = CalcularIMC(70, 1.75);
        Assert.AreEqual(22.86, imc, 0.01);
    }
}
```

06

Testes de Regressão



Os testes de regressão garantem que as novas alterações no código não introduzam novos bugs e que o software continue funcionando como esperado após modificações.

Exemplo de Teste de Regressão

Considere uma função que converte temperatura de Celsius para Fahrenheit:

```C#
public double CelsiusParaFahrenheit(double celsius)
{
    return (celsius * 9/5) + 32;
}
```

Um teste de regressão pode garantir que a função ainda funcione após uma mudança:

```C#
using NUnit.Framework;

[TestFixture]
public class TestTemperatura
{
    [Test]
    public void TestaCelsiusParaFahrenheit()
    {
        Assert.AreEqual(32, CelsiusParaFahrenheit(0));
        Assert.AreEqual(212, CelsiusParaFahrenheit(100));
        Assert.AreEqual(-40, CelsiusParaFahrenheit(-40));
    }
}
```

07

Testes de Performance

Os testes de performance avaliam a velocidade, a capacidade de resposta e a estabilidade do sistema sob carga de trabalho.

Exemplo de Teste de Performance

Para medir o tempo de execução de um método que ordena uma lista:

C#

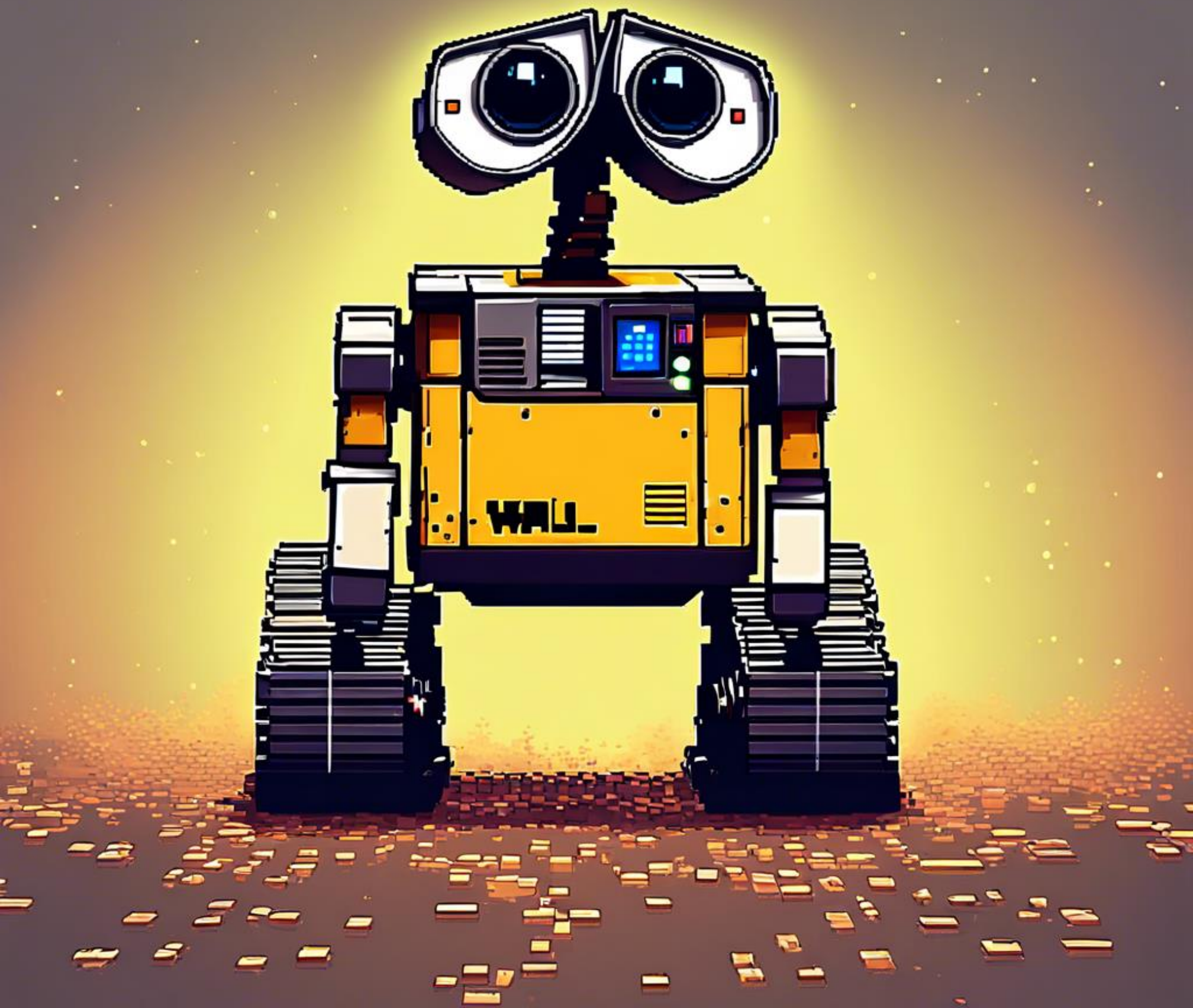
```
using System;
using System.Diagnostics;
using Xunit;

public class Testes
{
    [Fact]
    public void TestePerformance()
    {
        int[] lista = { 5, 3, 8, 1, 9, 2 };
        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();
        OrdenarLista(lista);
        stopwatch.Stop();
        double duracao = stopwatch.Elapsed.TotalSeconds;
        Assert.True(duracao < 0.1);  // Esperamos que a ordenação seja rápida
    }

    public int[] OrdenarLista(int[] lista)
    {
        Array.Sort(lista);
        return lista;
    }
}
```

Aqui, verificamos se o método ` OrdenarLista ` executa dentro de um tempo aceitável.:

Agradecimentos



OBRIGADO POR LER ATÉ AQUI !!!

Gostaria de expressar minha sincera gratidão a todos que dedicaram seu tempo para ler este ebook. Espero que ele tenha sido uma fonte útil e esclarecedora de conhecimento sobre testes de software.

Este trabalho foi realizado com a assistência de uma Inteligência Artificial (IA), que auxiliou na organização, criação e formatação dos conteúdos apresentados. No entanto, todo o material foi supervisionado e validado por mim.

Este ebook foi criado para proporcionar um aprendizado acessível sobre o que são os testes de software e sua importância na vida de um programador, com conteúdo direto e fácil de entender até para quem nunca programou antes.

Comece por esse ebook a sua jornada na programação, tenho certeza que você irá amar!

Autora: Kássia Almeida Moura