Diagrama

Descrição gerada automaticamente

**RED 1. Write a test that fails.**

A parte que escreve um teste que vai falhar, vai falhar porque vai ter o foco na funcionalidade já implementada, porém ela não está implementada. Como vai testar apontando para essas funcionalidades que ainda não foi implementada ela vai falhar.

**GREEN 2. Make the code work.**

Nesa etapa é onde vamos fazer o teste funcionar, vamos pegar os objetos que foi imaginado e fez referência e vai implementar as classes, os objetos, as informações que têm que fazer o teste funcionar.

**REFACTOR: 3. Eliminate redundancy**

Vai dar uma olhada, uma geral, nos objetos, classes que desenvolveu e ver o que pode melhorar, fica mais simples. Faça isso na classe do projeto, mas também pode fazer no teste, mas o foco é no projeto.

É recomendado que utilize framework de testes.

Nesse exemplo vamos trabalhar com o framework nativo que é Visual Studio Unit Test Framework e ele tem tudo o que precisa para trabalhar nesse projeto exemplo. Além do framework nativo que o Visual Studio tem, temos também a liberdade de trabalhar com outros frameworks que encontrar no mercado e atenda suas necessidades de fazer testes. Por exemplo o nUnit que é um código open Source de teste que tem uma interface própria para você trabalhar e tem todo uma ligação com Visual Studio para usar de forma integrada.

No VS2019 vamos cria um projeto de nome WebCoreTDD

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Adicionar o proejo de teste Unit

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

Para ambos os projeto .NET 5.0

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Quando tem data notation [TestClass] indica que é uma classe de teste, caso contrário é uma classe normal como por exemplo public class NaoClassTest

Texto

Descrição gerada automaticamente

Vamos deletar a classe ”UnitTest1.cs” e adicionar um nova classe com nome de “FibonacciTest.cs”

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCoreTDD.Tests

{

class FibonacciTest

{

}

}

Vamos deixar a class para teste deixando pública e colocando o data notation [TestClass]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCoreTDD.Tests

{

[TestClass]

public class FibonacciTest

{

//POSIÇÃO 0 1 2 3 4 5 6 7 8 9 10 ...

//NUMERO 0 1 1 2 3 5 8 13 21 34 55 ...

[TestMethod]

public void PrimeiroElementoDeveSerZero()

{

var resultado = Fibonacci.ObterElemento(0);

Assert.AreEqual(resultado, 0, "Valor náo é zero.");

}

}

}

Ao compilar/executar o teste vai dar erro porque estamos na fase **RED 1. Write a test that fails.**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Agora vamos criar nossa classe public static class Fibonacci com o método ObterElemento(0)

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCoreTDD.Tests

{

[TestClass]

public class FibonacciTest

{

//POSIÇÃO 0 1 2 3 4 5 6 7 8 9 10 ...

//NUMERO 0 1 1 2 3 5 8 13 21 34 55 ...

[TestMethod]

public void PrimeiroElementoDeveSerZero()

{

var resultado = Fibonacci.ObterElemento(0);

Assert.AreEqual(resultado, 0, "Valor não é zero")

}

// Para fins de teste vamos criar aqui mesmo, mas o correto seria no projeto

public static class Fibonacci

{

public static int ObterElemento(int posicao)

{

return -1;

}

}

}

}

Para executar o teste clique botão direito e em seguida “Run Test(s)”, antes exiba na barra lado esquerdo em “View”, “Other Windows” e “Test Results”. No menu “Test Result” tem como debugar o teste

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Teste falhou no primeiro ciclo do TDD

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Agora vamos implementar para que valide da forma correta alterado o retorno par zero

public static class Fibonacci

{

public static int ObterElemento(int posicao)

{

return 01;

}

}

Agora o teste passou

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Temos opção de executar só o teste que falhou, executar todos, só os que passou pra confirmar se continuam coretos, etc. Temos como usar o debug

Tela de computador com letras brancas

Descrição gerada automaticamente

Temos como fazer um play list selecionando quais queremos testar

Tela de jogo de vídeo game

Descrição gerada automaticamente

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Outras opções

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Ver log do erro

Tela de computador

Descrição gerada automaticamente

Assert.AreEqual(resultado, 0, "Valor náo é zero.");

As opções de **Assert** são pra validar o que está sendo retornado do método e ele é um objeto da classe de test que está utilizando

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Tem várias opções

Texto

Descrição gerada automaticamente