PUC Minas

Trabalho de Teste de Software

Plano de Teste

Versão 1.3

Grupo:

- Luiz Felipe
- Thaís Damásio
- Vinícius Pacheco

Histórico de Revisão

Data	Versão	Descrição	Autor
10/06/2019	1.0	Criação do template do documento	Thaís Damásio
15/06/2019	1.1	Adição da descrição do projeto	Vinícius Pacheco
16/06/2019	1.2	Prints dos resultados do projeto	Luiz Felipe
16/06/2019	1.3	Revisão e finalização do documento	Thaís Damásio

Sumário

Histórico de Revisão 2
1. Introdução 4
   1.1. Proposta 4
   1.2. Escopo 4
2. Itens de teste alvo 4
3. Visão geral dos testes planejados 4
4. Metodologia de teste 4
   4.1. Identificando e justificando testes 5
   4.2. Conduzindo os testes 6
5. Entregas 7
   5.1. Análise de cobertura de testes 7
   5.2. Resultados Obtidos dos testes 9
6. Necessidades Ambientais 9
   6.1. Elementos do software base no ambiente de teste 9
   6.2. Ferramentas de produtividade e suporte 10

1. Introdução

1.1. Proposta

A proposta deste plano de teste é:

- Apresentar um projeto simples feito em C#, o qual será realizado um teste de unidade;
- Apresentar os códigos e o teste de unidade realizado;
- Apresentar os resultados e conclusão obtida;

1.2. Escopo

O projeto utilizado para realização de teste é um aplicativo que simula um Banco chamado de *Bank*, mas sem o uso de *Backend* (Banco de Dados).

A ideia geral da aplicação é permitir que sejam criadas Contas Correntes e as mesmas são armazenadas em uma estrutura de dados de lista de forma que seja possível recuperar e manipular uma conta já criada. As contas possuem somente duas funções: sacar e depositar. E sobre essas duas funções serão realizados dois tipos de teste, o teste de classe de equivalência e o teste de valor limite.

Todo o teste será realizado com o apoio do framework **NUnit** e todo o projeto foi desenvolvido na linguagem de programação C#.

2. Itens de teste alvo

A lista abaixo identifica os itens de teste – software, hardware, e elementos de produtos de suporte – que foram identificados como alvos para testes. Esta lista representa quais itens serão testados.

- Software Bank
    - Testar a classe Conta Corrente, mais especificamente seus métodos Sacar e Depositar

3. Visão geral dos testes planejados

Pelo fato do projeto ser simples, escolhemos aplicar apenas um teste de valor limite e de classe de equivalência para garantir que o método sacar e depositar da classe conta corrente funcionem corretamente.

Desta forma o teste visa garantir a corretude da Classe Conta corrente, não permitindo:

- Saques negativos ou igual a zero;
- Saques maiores que o saldo;
- Depósitos negativos ou igual a zero;

4. Metodologia de teste

Como metodologia de teste foi aplicado o TDD (*Test Driven Development* ou Desenvolvimento Orientado por Testes). Desta forma foi implementado primeiramente os testes e depois a funcionalidade foi desenvolvida com o intuito de passar nos testes. Após alcançado este objetivo, toda a implementação foi refatorada.

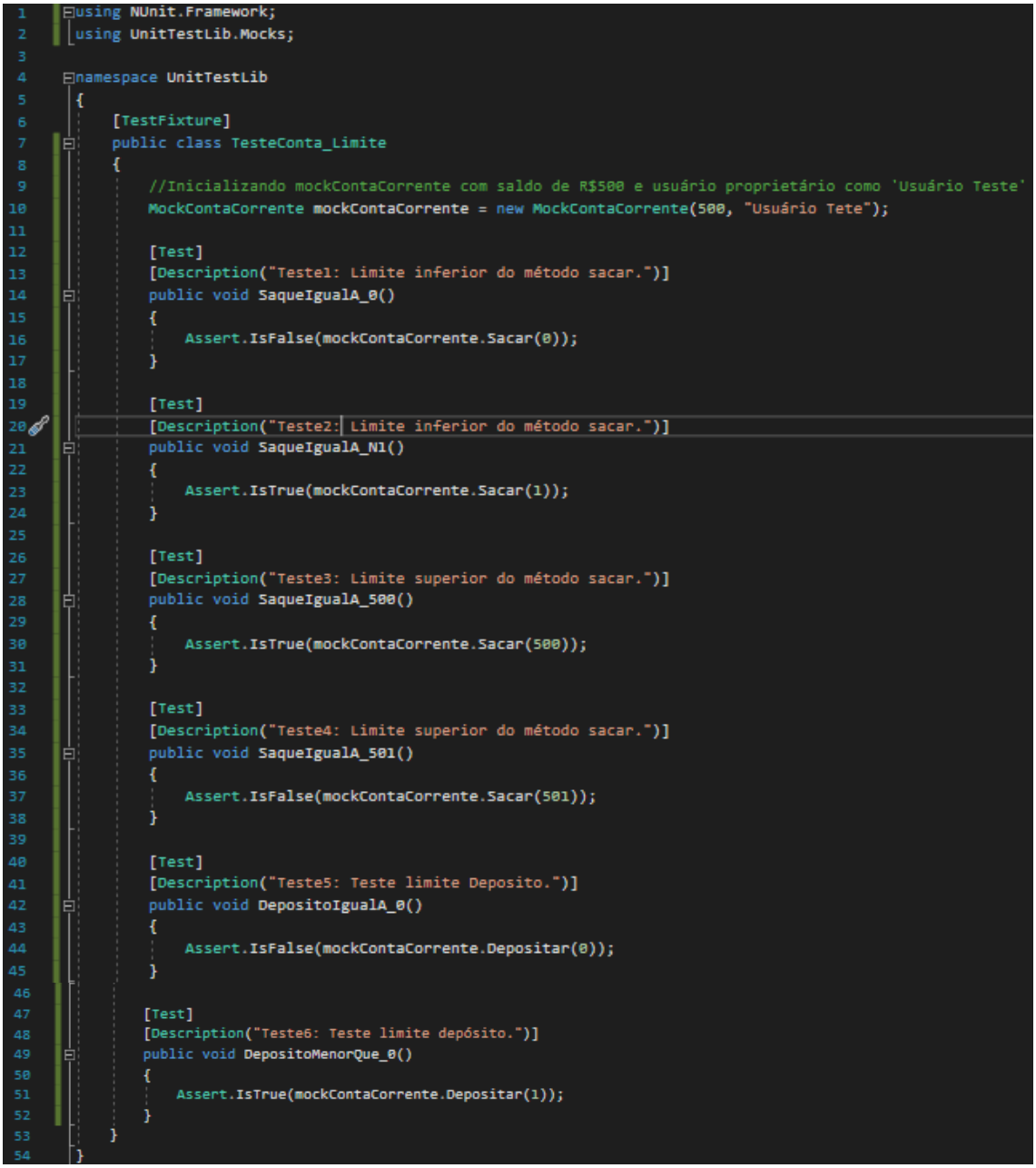
4.1. Identificando e justificando testes

Cada caso de teste foi implementado em um método da classe de teste, eles foram identificados a partir de seu nome de assinatura. Por exemplo, um caso de teste para testar o método de saque utilizando uma classe de equivalência inválida que seria “saque menores que zero” foi nomeado da seguinte forma: “*SaqueMenorQue_0”*, desta forma note que os métodos terão nomes descritivos sobre o que eles estão testando. Além disso cada método de teste possui um atributo de descrição (*Description*) que como o próprio nome sugere, armazena a descrição do que o método faz, isso é permitido graças aos recursos proporcionados pelo NUnit framework. Veja abaixo as imagens que apresentam o código de teste:

Teste da Conta Corrente com Classe de Equivalência

```
using NUnit.Framework;
using UnitTestLib.Mocks;

namespace UnitTestLib.Teste_ContaCorrente
{
    class TesteConta_Classe
    {
        //Inicializando mockContaCorrente com saldo de R$500 e usuário proprietário como 'Usuário Teste'
        MockContaCorrente mockContaCorrente = new MockContaCorrente(500, "Usuário Tete");

        [Test]
        [Description("Teste1: O método depositar da conta corrente não deve aceitar valores negativos ou nulos.")]
        public void SaqueMenorQue_0()
        {
            Assert.IsFalse(mockContaCorrente.Sacar(-100));
        }

        [Test]
        [Description("Teste2: O método sacar deve funcionar corretamente.")]
        public void SaqueCorreto()
        {
            Assert.IsTrue(mockContaCorrente.Sacar(100));
        }

        [Test]
        [Description("Teste3: O método sacar da conta corrente não pode permitir saques maiores que o saldo.")]
        public void SaqueMaiorQue_Saldo()
        {
            Assert.IsFalse(mockContaCorrente.Sacar(700));
        }

        [Test]
        [Description("Teste4: O método depositar da conta corrente não deve aceitar valores negativos ou nulos.")]
        public void DepositoMenorQue_0()
        {
            Assert.IsFalse(mockContaCorrente.Depositar(-100));
        }

        [Test]
        [Description("Teste5: O método depositar deve funcionar corretamente.")]
        public void DepositoCorreto()
        {
            Assert.IsTrue(mockContaCorrente.Depositar(100));
        }
    }
}
```

Teste da Conta Corrente com Valor Limite



```
using NUnit.Framework;
using UnitTestLib.Mocks;

namespace UnitTestLib
{
    [TestFixture]
    public class TesteConta_Limite
    {
        //Inicializando mockContaCorrente com saldo de R$500 e usuário proprietário como 'Usuário Teste'
        MockContaCorrente mockContaCorrente = new MockContaCorrente(500, "Usuário Tete");

        [Test]
        [Description("Teste1: Limite inferior do método sacar.")]
        public void SaqueIgualA_0()
        {
            Assert.IsFalse(mockContaCorrente.Sacar(0));
        }

        [Test]
        [Description("Teste2: Limite inferior do método sacar.")]
        public void SaqueIgualA_N1()
        {
            Assert.IsTrue(mockContaCorrente.Sacar(1));
        }

        [Test]
        [Description("Teste3: Limite superior do método sacar.")]
        public void SaqueIgualA_500()
        {
            Assert.IsTrue(mockContaCorrente.Sacar(500));
        }

        [Test]
        [Description("Teste4: Limite superior do método sacar.")]
        public void SaqueIgualA_501()
        {
            Assert.IsFalse(mockContaCorrente.Sacar(501));
        }

        [Test]
        [Description("Teste5: Teste limite Deposito.")]
        public void DepositoIgualA_0()
        {
            Assert.IsFalse(mockContaCorrente.Depositar(0));
        }

        [Test]
        [Description("Teste6: Teste limite depósito.")]
        public void DepositoMenorQue_0()
        {
            Assert.IsTrue(mockContaCorrente.Depositar(1));
        }
    }
}
```

4.2. Conduzindo os testes

Para a condução dos testes é necessário possuir a pasta Bank e UnitTestLib e estar atento aos seguintes detalhes:

1. O programa testado está armazenado na pasta Bank;
2. Este programa possui na pasta Bank/Bank/bin/Debug um arquivo chamado Bank.dll;
3. O arquivo Bank.dll É usado pelo teste, ele é importado nas referências da pasta de teste;
4. O teste está armazenado na pasta UnitTestLib;
5. Dentro da pasta UnitTestLib/UnitTestLib/bin/Debug possui a biblioteca de testes UnitTestLib.dll;
6. Devemos usar UnitTestLib.dll em algum console ou GUI do Nunit para que os testes possam ser executados.

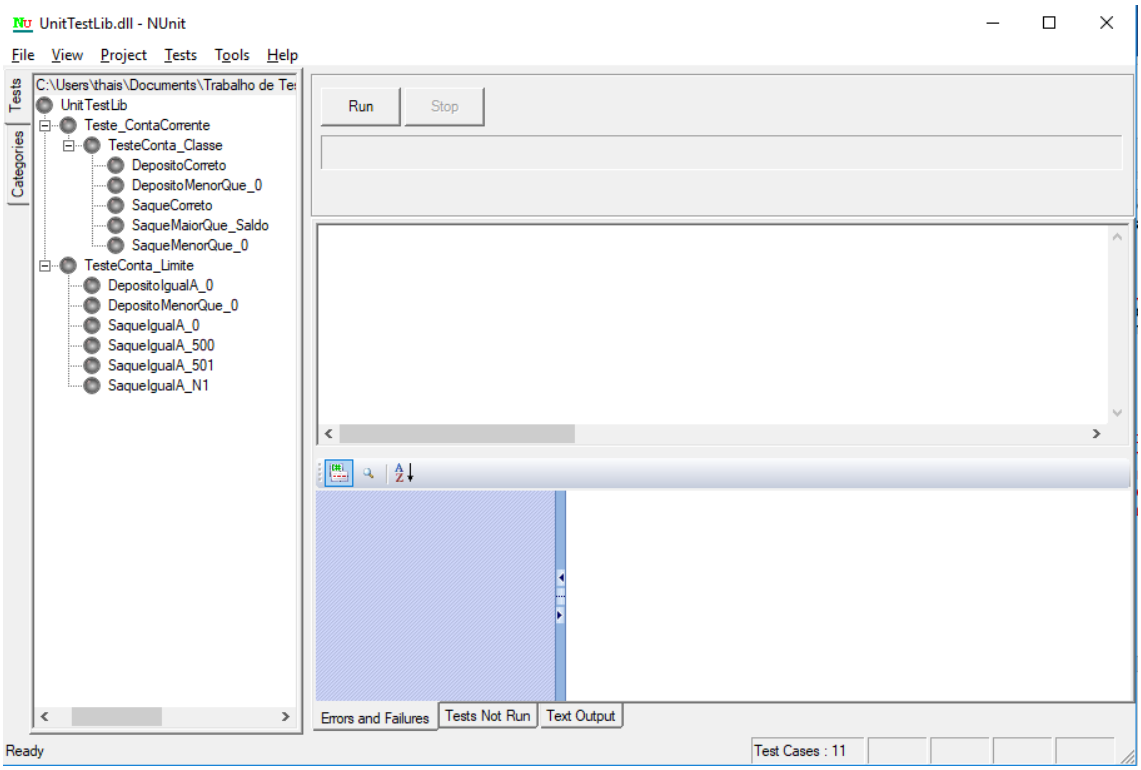
7. Antes é crucial verificar se o arquivo nunit.framework.dll e Bank.dll estão na pasta de Debug do projeto de UnitTestLib, para que sua execução ocorra corretamente.

5. Entregas

Junto a este documento está sendo entregue duas pastas de projeto em C#. Uma com o código fonte do teste e outra com o código fonte do projeto. Além disso o projeto também está vinculado ao GitHub o que permite seu acesso também por meio deste link: https://github.com/Thais-Damasio/bank_test.

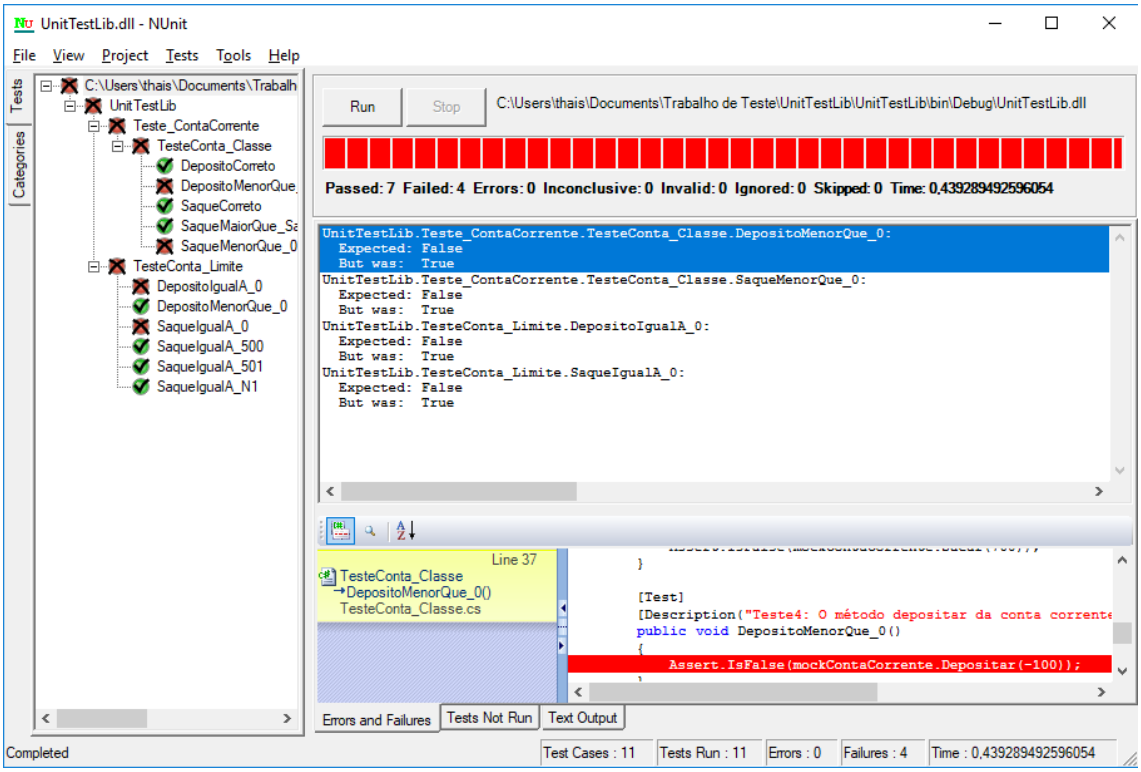
5.1. Análise de cobertura de testes

Os testes foram executados com apoio do framework NUnit 2.7 e sua GUI (Interface Gráfica) de versão 2.7 também. Foi importado a biblioteca de testes e o mesmo foi carregado dentro da GUI como mostra a imagem abaixo:



Considerando que primeiramente foi implementado os testes e depois as funcionalidades. Esta é a etapa em que as funcionalidades já foram implementadas e agora estão em fase de teste para que possam ser aprovadas e assim refatorada.

É interessante ressaltar que inicialmente encontramos uma falha na primeira execução como mostra a imagem abaixo:



Os testes mostraram que saques e depósitos negativos ou igual a zero retornaram o status *true* quando era esperado *false*. Então foi possível capturar estes erros e atualizar a versão do projeto para que garanta sua funcionalidade correta.

Assim alteramos o projeto:

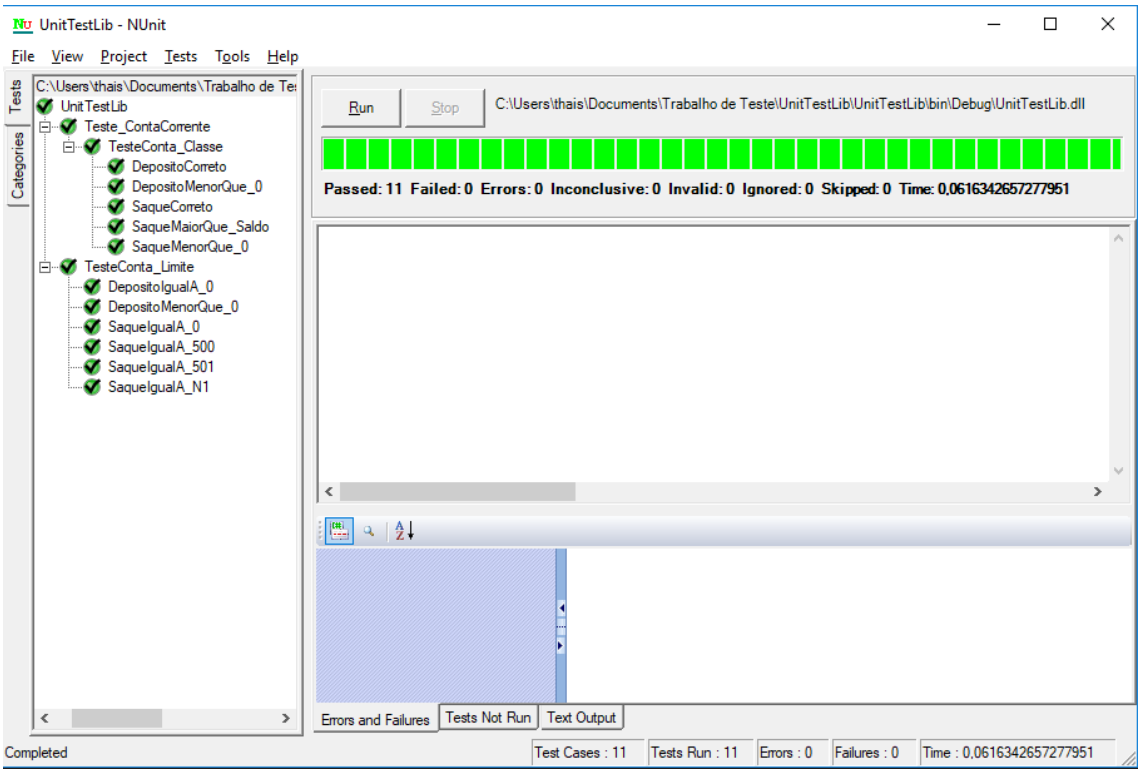
Antes

```
using System.Text;
using System.Threading.Tasks;

namespace Bank
{
    public class ContaCorrente : Conta
    {
        public ContaCorrente(string donoConta) : base(donoConta) { }

        public override bool Depositar(double valor)
        {
            this.saldo += valor;
            return true;
        }

        public override bool Sacar(double valor)
        {
            if (this.saldo >= valor)
            {
                this.saldo -= valor;
                return true;
            }
            else
                return false;
        }
    }
}
```

Depois

```
using System.Text;
using System.Threading.Tasks;

namespace Bank
{
    public class ContaCorrente : Conta
    {
        public ContaCorrente(string donoConta) : base(donoConta) { }

        public override bool Depositar(double valor)
        {
            if (valor <= 0)
                return false;
            this.saldo += valor;
            return true;
        }

        public override bool Sacar(double valor)
        {
            if (valor <= 0)
                return false;
            if (this.saldo >= valor)
            {
                this.saldo -= valor;
                return true;
            }
            else
                return false;
        }
    }
}
```

Com as funcionalidades corrigidas rodamos novamente o teste com o seguinte resultado:



Como podemos observar, todos os casos de teste passaram e agora foi possível a fase de refatoração do código e assim disponibilizar sua versão para uso.

5.2. Resultados Obtidos dos testes

Como mencionado anteriormente todo o teste foi realizado com apoio do framework NUnit e sua Interface gráfica para testes. Por meio dos resultados foi encontrado erros pequenos de implementação e a mudança pode ser realizada no decorrer do processo. Mas com o apoio do sistema de versionamento do GitHub foi possível controlar versões e realizar alterações. Como este é um projeto com intuito acadêmico, nesta fase não foi registrado as solicitações de alterações como previsto no modelo RUP e as boas práticas da Gestão de Configuração do Software.

6. Necessidades Ambientais

Esta sessão apresenta os recursos não humanos necessários para a realização do teste:

6.1. Elementos do software base no ambiente de teste

Os seguintes elementos de software base que foram necessários no ambiente de teste para este Plano de Teste.

Nome do Elemento do Software	Versão	Tipo e Outras Notas
.NET framework	3.5^	Nesta versão e superiores está incluída o LinQ.

NUnit framework	2.7	-

6.2. Ferramentas de produtividade e suporte

As seguintes ferramentas foram empregadas para suportar o processo de teste deste Plano de Teste.

Tipo ou Categoria da Ferramenta	Nome da Ferramenta	Vendedor
Controle de versão	GitHub	Microsoft
Interface de visualização dos resultados do teste	NUnit 2.7	Software livre