Midterm Assignment
Software Verification and validation

Marks: **30**

Deadline:19-April-2020 12:00 pm

ID: 14646
NAME: Abdul Musawer

A. Choose any Testing tool from the list, and write downs its

- Choose any Testing tool from the list, and write downs its
    - pros/ cons
    - Functionality
    - Supporting Languages
    - Supporting Tests
    - Write a short(faulty) code, Test using this tool, and show the bugs in the code.

Solution:

Test tool Selected:

NUnit installed in C#

A: PROS AND CONS OF NUnit

- It is stand alone and easy to include for new developers which run smoothly and instantly.
- It is quick in running tests.
- Rapid response to bugs,
- It is open source.
- Dying a slow death in a few years.
- It never generates the solution but catches the problem quickly.
- No tightdy integration with visual studio.
- Automatic linking with bugs is needed to be introduced.
- C based extensions not available.

B: FUNCTIONALITIES

- Easy to Understand for developers and testers.
- It is object oriented based.
- It is flexible and can test any code in .NET language.
- It uses the attribute feature.
- Nunit provides the following Assertions:

1. Testing the equality that if it (condition) is equal or not.
2. Identification checking files which contain the same attributes or not in the files.
3. Condition checking weather empty, NULL, equal to, not equal to, isEmpty, isNotEmpty.
4. It can also do some comparison greater than, less than, or instance of, is AssignableFrom.
5. Custom Asserts such as that of reserved keywords like That, This.
6. Strings
7. Exceptions

C: SUPPORTING LANGUAGES:

1. C#
2. VC
3. VB.NET
4. J#

D: SUPPORTING TESTS:

1. Console Runner test.
2. GUI runner Test.
3. pNUNIT runner.
4. Simple Expression.
5. Namespace filtering
6. Property base filtering:
    a. Author
    b. Category
    c. Description
    d. SetCulture
    e. SetUICulture
    f. TestOf
    g. IgnoreUntilDate
7. Test ID filtration.
8. Compound expression.
9. Command Line usage.

E:

I made two classes by using the example of a bank account such as transfering funds and also showing the balance.
Note: these classes are build in C# and tested properly with NUnit.

The first class is by the name of **AccountTransaction.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace bank
{
        class AccountTransaction
        {
        private float balanceToTransfer;
        private float min_Balance = 10;

        public AccountTransaction()
        {
```

```
        }
        public AccountTransaction(int value)
        {
        balanceToTransfer = value;
        }
        public void Deposit(float amount)
        {
        balanceToTransfer += amount;
        }
        public void Withdraw(float amount)
        {
        balanceToTransfer -=amount;
        }
        public void TransferFunds(AccountTransaction destination, float amount)
        {
        destination.Deposit(amount);
        Withdraw(amount);
        }
        public AccountTransaction TransferMinFunds(AccountTransaction destination, float amount)
        {
        if (balanceToTransfer - amount > min_Balance)
        {
        destination.Deposit(amount);
        Withdraw(amount);
        }
        else
        {
        throw new NotEnoughFundsException();
        }
        return destination;
        }
        public float balanceToTransfer
        {
        get { return balanceToTransfer; }
        }
        public float min_Balance
        {
        get { return min_Balance; }
        }
        }
}
```

The other class is by the name of **Source_To_Destination**.cs

```
using NUnit.Framework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace bank
{
        [TestFixture]
        class Source_to_Destination
        {
        Account sourceAccount;
        Account DestinationAccount;

        [SetUp]
        public void InitSource_to_Destination()
        {
        //arranging
        sourceAccount = new AccountTransaction();
        sourceAccount.Deposit(200.00F);
        DestinationAccount = new AccountTransaction();
        DestinationAccount.Deposit(150.00F);
        }

        [Test]
        [Category("pass")]
        public void TransferFunds()
        {
        //action is happening in here
        sourceAccount.TransferFunds(DestinationAccount, 100.00F);

        //assert is used to verify
        Assert.AreEqual(250.00F, DestinationAccount.Balance);
        Assert.AreEqual(100.00F, sourceAccount.Balance);
        }


        [Test, Category("pass")]
        [TestCase(200,0,78)]
        [TestCase(200, 0, 189)]
        [TestCase(200, 0, 1)]
        public void TransferFund(int A, int B, int C)
        {
```

```
        Account sourceAccount = new AccountTransaction();
        sourceAccount.Deposit(A);
        Account DestinationAccount = new AccountTransaction();
        DestinationAccount.Deposit(B);

        sourceAccount.TransferFund(DestinationAccount, C);
        Assert.AreEqual(C,DestinationAccount.Balance );
        }

        [Test, ExpectedException(typeof(NotEnoughFundsException))]
        [Category("fail")]
        [TestCase(200,150,200)]
        [TestCase(200, 150, 300)]
        [TestCase(200, 150, 500)]
        [TestCase(200, 150, 1000)]
        public void TransferFundFail(int A, int B, int C)
        {
        Account sourceAccount = new AccountTransaction();
        sourceAccount.Deposit(A);
        Account DestinationAccount = new AccountTransaction();
        DestinationAccount.Deposit(B);

        DestinationAccount = sourceAccount.TransferFund(DestinationAccount, C);
        }


        [Test]//, The Exception handling part ExpectedException(typeof(NotEnoughFundsException))]
        [Category("fail")]
        [Combinatorial]
        public void TransferFundFailAll([Values (200,500)] int A, [Values(0,20)]int B,
        [Values(140,135)]int C)
        {
        Account sourceAccount = new AccountTransaction();
        sourceAccount.Deposit(A);
        Account DestinationAccount = new AccountTransaction();
        DestinationAccount.Deposit(B);

        DestinationAccount = sourceAccount.TransferFund(DestinationAccount, C);
        }
        }

        public class NotEnoughFundsException:ApplicationException
        {
        }
}
```

TEST ERRORS:

I intentionally left the errors for testing purpose in the method or function:
TransferFund( )

1. Source: Source_To_Destination.cs line 74
   TransferFunFailAll(200, 0, 135)
   Message: Bank.NotEnoughFunds Exception was Expected
   Elapsed time: 0:00:00:0.000001

2. Source: Source_To_Destination.cs line 74
   TransferFunFailAll(200, 20, 190)
   Message: Bank.NotEnoughFunds Exception: Error in the Application
   Elapsed time: 0:00:00:0.000001
   StackTrace:
   TransferFunFailAll(int32 A, int32 b, int32 C)

   2 tests failed!
   16 tests passed!