DEVELOPMENT OF LEARNING OBJECTS FOR TEACHING SOFTWARE TESTING USING A CYBER LEARNING ENVIRONMENT

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Ashish Kumar Singh

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Software Engineering

November 2016

Fargo, North Dakota

North Dakota State University

Graduate School

Title

DEVELOPMENT OF LEARNING OBJECTS FOR TEACHING SOFTWARE TESTING USING A CYBER LEARNING ENVIRONMENT

By

Ashish Kumar Singh

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Kendall E. Nygard

Dr. Limin Zhang

Approved:

November 18, 2016	Brain M. Slator
Date	Department Chair

ABSTRACT

This paper is an attempt to extend the database of learning objects on WReSTT (Web Repository of Software Testing Tools) that students enrolled in programming courses can use to improve their understanding of testing concepts and testing skills. The first learning object describes Microsoft unit test on C# language. Second learning object describes data driven unit tests using different data sources, such as database, xml, and csv. Third learning object describes different assert classes of unit test. Finally, multiple quizzes are presented to assess student learning.

ACKNOWLEDGEMENTS

I would like to thank all the people who supported me during the process of writing this paper. My heart felt gratitude goes to Dr. Gursimran Walia. He not only guided me in writing this paper but also provided a great learning experience during my master's program. Furthermore, I thank my committee member's Dr. Kendall E. Nygard and Dr. Limin Zhang.

I would like to thank my family and friends for their continued support during last three years. I especially want to thank friends Maninder Singh Thind and Shweta Srivastava for their help.

Most importantly I am thankful for the blessings of my mother, Shobha Singh and support of my brother, Manish Singh.

DEDICATION

I dedicate this work to my father, Late Badri Prasad Singh.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDIX FIGURES

INTRODUCTION

Software testing is a major step of software development life cycle. It accounts for more than half of the total cost of developing software. As software becomes more ubiquitous and complex, it not only demand for in-depth knowledge of Software Engineering but also good and practical knowledge of validation techniques like testing. One of the major requirements necessary to fulfill these improvement is more and better trained professionals in the area of software testing. In today’s world, software testing knowledge is a crucial industry need hence more and more industries are expecting their employees to have relevant testing knowledge and expertise. New and growing technology areas (web applications, embedded software, secure systems, and object-oriented software) require software that is tested more thoroughly.

In order to fulfill the software testing requirement in industries, various universities make an effort to included software testing course in Computer Science / Software Engineering program especially during the introductory computer programming courses. These approaches range from the integration of testing into CS1 and CS2 using novel approaches, such as test-driven development (TDD) and test-driven learning (TDL), to the restructuring of more advanced CS/SE courses which include software testing component (Timothy C. Lethbridge, 2007). As software testing is a high level concept and due to the complex fundamental knowledge of software engineering, and system development, teaching software testing with the identified required courses is very challenging (Wikipedia). Also, current software testing curriculum is fails to ensure that students continues to use the technique that they learned in previous courses. The Web-Based Repository of Software Testing Tools - (WReSTT) is an online repository for learning software testing in pedagogical style. It provides adequate testing techniques in most effective, efficient and continuous manner.

One of the important software testing technique is unit test. Most of the industries are making sure their application code is covered by unit test. There are several benefits of having unit test like it helps in finding bugs early, significantly reduces production bugs, makes complex code easy to understand, provide form of documentation, save lot of development time, easier to save and refactor code, and developer becoming more confident. In short, unit testing is essential for application.

The objective of this paper is to create learning objects on different kind of unit testing. As Microsoft Unit Test is one of the most common framework used in industries, this paper has used Microsoft Unit Test as framework and C# as programming language. Our first objective is to familiarize students with fundamental concepts of unit test, naming conventions of unit test methods and classes that fallows in industries. Our second objective is to help students to create and implement data driven unit test using different data sources (database, xml, and csv) in easiest possible way. Our third objective is to show students the uses of different types of assert class provided by Microsoft Unit Test framework, so that students could implement them effectively while creating different unit tests. Our fourth and final objective is to assess impact on student's knowledge which they got while going through learning objects provided in this paper.

The remainder of the paper is organized as follows. Section 2 presents a brief overview of WReSTT and its features. In fallowing three sections we present our Learning Objects on Unit Test, Data Driven Unit Test, and Assert Classes of Unit Test respectively. In last section we discuss and conclude about presented Leaning Objects. Also, Appendix A includes several quizzes of all Learning Objects that were presented in this paper. In order to get good assessment of student's knowledge enhancement in software testing by using these learning objects, each quiz should be used as pre and post-test for any learning object.

INTRODUCTION TO WReSTT

Web-based Repository of Software Testing Tools (WReSTT) is a cyber enabled virtual learning environment that provides students and instructor with fruitful information of software testing, supports various type of teaching materials in the form Learning Objects, and facilitates with social and media networking and peer study environments (Wikipedia). WReSTT is a web based repository that includes pedagogical Learning Objects (tutorials) and Quizzes on the core concepts of software testing. It provides easy, interactive and chronical approach to students to learn software testing concepts along with tools.

The basic idea behind WReSTT was to provide online repository which contains tutorials on software testing tools and links to other learning resources on software testing, which bring WReSTT V1 (version 1) into existence. After using WReSTT V1, students and instructors provided feedback. Based on those feedback, as well as based on the results of several studies performed, WReSTT V1 enhanced to much bigger and boarder spectrum, WReSTT V2. It does not include software testing tools tutorials but also collaborative learning environment that contains broader array of tutorials on testing concepts and testing tools.

WReSTT V2 provides the minimally disruptive approach for integrating software testing into SE and high- level programming courses. WReSTT V2 is minimally disruptive because it do not expect from instructors to make major changes in current curriculum. This provides an advantage to instructor, making it easier for them to integrate WReSTT V2 into their curriculum, and thus making it more likely to be adopted and useful for a large number of institutions.

Using WReSTT students could form their virtual learning groups and compete for points. These virtual groups will receive quizzes and other knowledge assessment tasks from instructors. Points will be awards to groups upon the successful completion of these quizzes and knowledge assessment tasks. This social aspect of WReSTT also provide students with ability to create profile, monitor the learning progress of classmates, post comments to the discussion boards, and identify which students have acquired the most points.

WReSTT V2 Design

WReSTT V2 was developed from the feedback provided by students and instructors after using WReSTT V1. WReSTT V2 has four tier architecture (As shown in Figure 1, (Peter J. Clarke, October 2014)) which includes collaborative learning and classroom management features. The main components of WReSTT are as fallows,

- Authentication: Requires user's credentials for providing secure access to the system. It has three features – Log in/out, Registration and password reset. These features decide kind of access need to provide to users. There are four kinds of access that could be provided to user – unauthorized, students, instructor, and administrator.
- Social: It provide social networking features (like Facebook) to users. Here users can create their profile, monitor activity stream, post comments to the discussion boards and monitor virtual points assigned to users in his respective classes. It has four subcomponents – profile, activity stream, discussion board, and virtual point. Virtual point subcomponent is common between Social and Learning component.
- Learning: This module basically has Learning Objects on different topic of software testing like unit test, data driven testing etc. Learning Objects contains self-explanatory tutorials (no sound) from beginner level to advance level. Quizzes are also provided for each Learning Objects for assessment. Learning also has three subcomponents – Learning Strategy, Tutorials, and quizzes. Learning Strategy has one subcomponent also called virtual point.

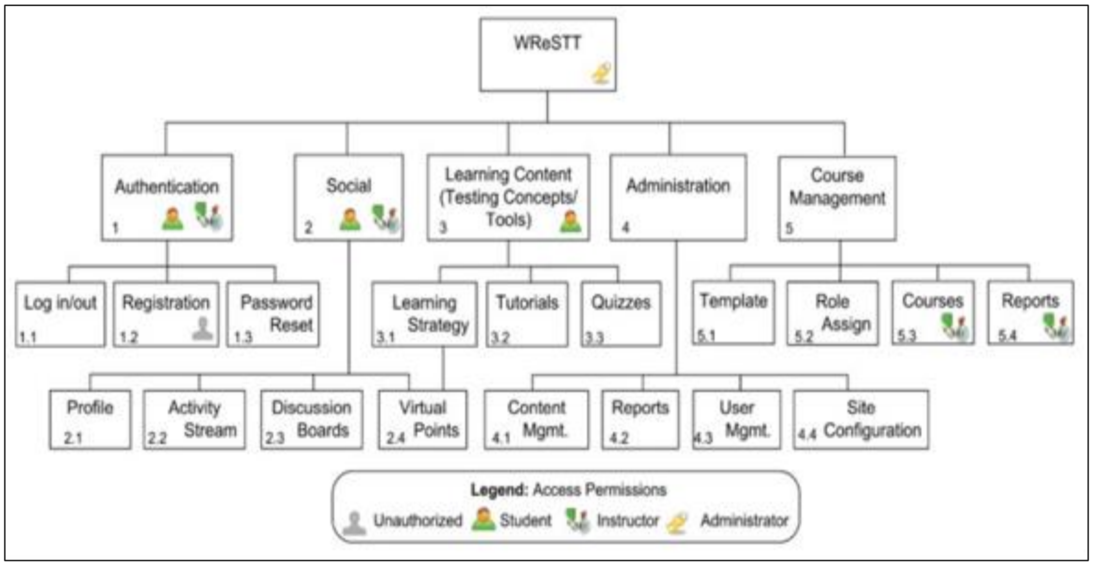


Figure 1. Block Diagram of WReSTT

- Administration: It provide administrative access to particular users which provide the ability to update content (Learning Objects, quizzes), generate system-side report, monitor and update user's access of the system, and configure the system (e.g. creating reports based on queries). Administration has four subcomponents – Content management, Report, User Management, and Site Configuration.
- Course Management: It provide access administrators to create new course and instructors to upload class rolls and generate reports. Course Management has four subcomponents – Template, Role Assign, Courses, and Reports.

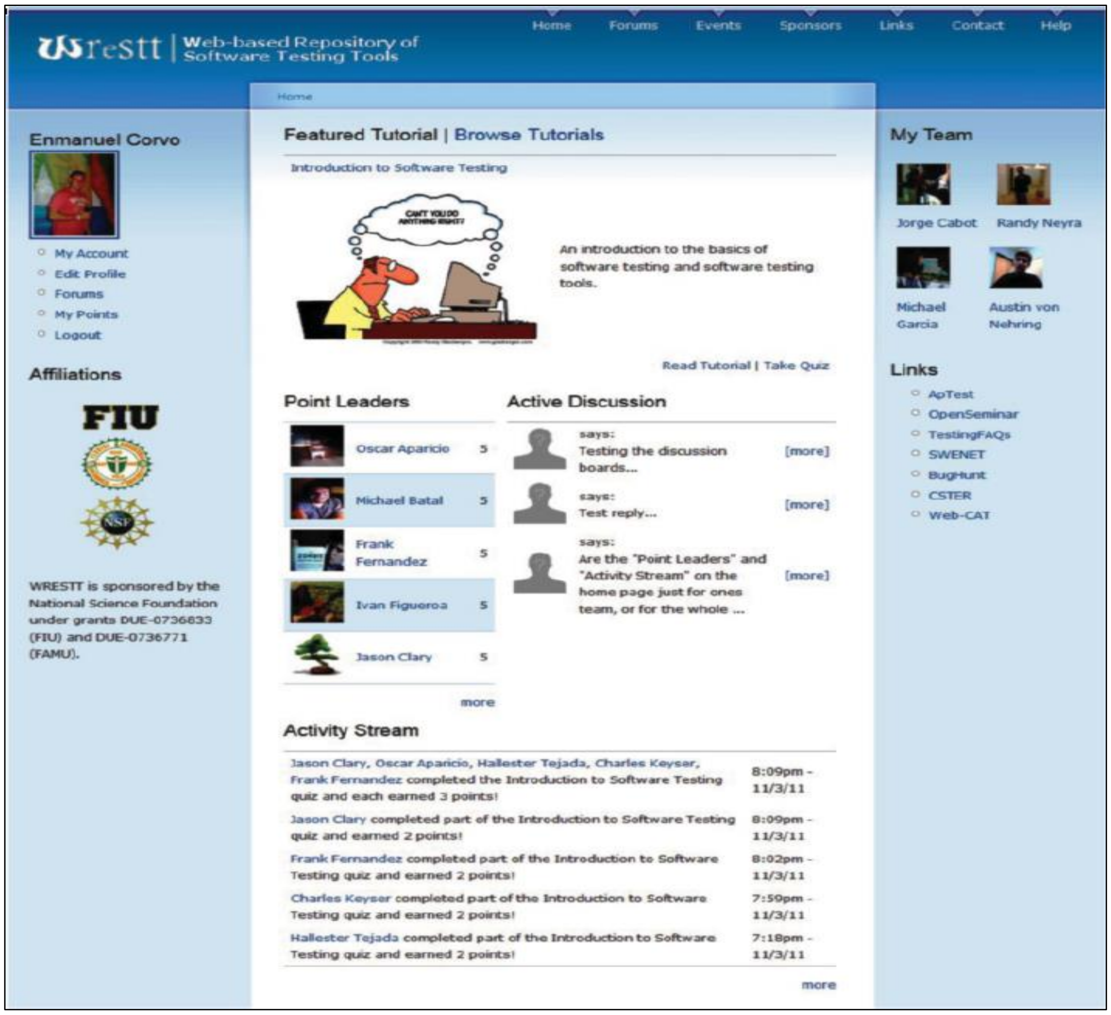
Collaborative Learning

The major change from WReSTT V1 to WReSTT V2 is incorporation of collaborative learning. The main focus of collaborative learning in WReSTT V2 is student involvement, cooperation, and teamwork throughout the learning process. WReSTT V2 primarily achieves student involvement, cooperation, and teamwork by rewarding students with virtual points, requiring team members to collaboratively participate in various activities in a timely manner, and providing opportunities for social engagement. Social engagement includes course discussion forums and activity streams, among other features. (Peter J. Clarke, October 2014).

To get effective collaborative learning, virtual points are awarded on individual basis when users participate in the various social engagement activities. These activities include creating a user profile, which include uploading a picture, posting questions, commenting on discussion forums, and answering questions posted to the forums. Collaborative learning occurs both at class level and team level. WReSTT V2 provide real-time access to its users so that student in the class can participate in the discussions that occurring in Forum. The other features of real-time access are Point Leaders, Active Discussion, and Activity Stream as shown in Figure 2. (Peter J. Clarke, October 2014). WReSTT V2 maps actual class project teams to virtual teams which is much easy to manage and collaborate.

Figure 2 shows one of the student's home pages from the fall 2011 undergraduate software testing class that used the collaborative learning strategy. Name of this student is "Emauel Corvo" from fall 2012 class. (Peter J. Clarke, October 2014). Here user created a profile, upload a picture, and commented on the discussion board.

Home section of WReSTT (Figure 2) consists of four sections – top section, left section, right section, and center section. Some sections also have subsection. The top section shows main component of the website which contain full name and logo of WReSTT. This section also provide many tabs like Home, Forum, Events, Sponsors, Links, Contact, and Help.



WReSTT | Web-based Repository of Software Testing Tools

Home Forums Events Sponsors Links Contact Help

Home

Enmanuel Corvo

- My Account
- Edit Profile
- Forums
- My Points
- Logout

Affiliations

FIU

WRESTT is sponsored by the National Science Foundation under grants DUE-0736833 (FIU) and DUE-0736771 (FAMU).

Featured Tutorial | Browse Tutorials

Introduction to Software Testing

An introduction to the basics of software testing and software testing tools.

Read Tutorial | Take Quiz

Point Leaders

Oscar Aparicio	5
Michael Batal	5
Frank Fernandez	5
Ivan Figueroa	5
Jason Clary	5

more

Active Discussion

says:
Testing the discussion boards... [more]

says:
Test reply... [more]

says:
Are the "Point Leaders" and "Activity Stream" on the home page just for ones team, or for the whole ... [more]

Activity Stream

Jason Clary, Oscar Aparicio, Hallester Tejada, Charles Keyser, Frank Fernandez completed the Introduction to Software Testing quiz and each earned 3 points!	8:09pm - 11/3/11
Jason Clary completed part of the Introduction to Software Testing quiz and earned 2 points!	8:09pm - 11/3/11
Frank Fernandez completed part of the Introduction to Software Testing quiz and earned 2 points!	8:02pm - 11/3/11
Charles Keyser completed part of the Introduction to Software Testing quiz and earned 2 points!	7:59pm - 11/3/11
Hallester Tejada completed part of the Introduction to Software Testing quiz and earned 2 points!	7:18pm - 11/3/11

more

My Team

Jorge Cabot Randy Neyra

Michael Garcia Austin von Nehring

Links

- ApTest
- OpenSeminar
- TestingFAQs
- SWENET
- BugHunt
- CSTER
- Web-CAT

Figure 2. A Student's Home Page in WReSTT V2

Left section shown the picture of logged user and WReSTT affiliations. Right section has two subsection – upper right section and lower right section. Upper right section shows the pictures of other members of the team with their name. Lower right section shows various important links. Also, center section has four subsection – center upper section, center left section, center right section, and center lower section. Center upper section provide student with the ability to browse the featured tutorial or browse other tutorials. Center left section has list of current point leaders in the class. Center right section shows active discussion forum with the most recent entries.

LEARNING OBJECT

Learning Object is part of pedagogical approach of WReSTT V2 which extensively covered software testing concepts and tools with quizzes. Learning Objects provides a more structured approach to students who plan to use WReSTT V2 as an independent learning approach. Learning Objects comes under the "Learning Content" category for WReSTT V2 which numbered as 3 in Figure 1. This Learning Content category has three subcategories also – Learning Strategy, Tutorials, and Quizzes. Learning strategies are collaborative which includes taking part in active discussion and answering to questions asked by other team members. Second category of Learning Content is Tutorials which is in the form of Learning Objects. WReSTT V2 has wide range of topics covered for software testing from very basic topics like using any particular IDE (NetBeans, visual studio) to complex topics like test driven development, and data driven development. Third and last sub category within Learning Contents is Quizzes. There are several quizzes for each topics of Learning Objects. These quizzes are either objective with multiple choice questions with one or more correct answers or subjective where student supposed to write correct answers for questions. Quizzes are created in such a way that it covers all the topics of current Learning Object and thus provides the good in-sight of progress. Virtual points are awarded on the successful completion of each quiz when quizzes are done either individually or collaboratively.

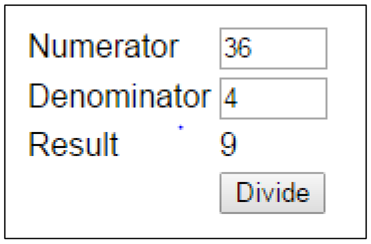
WReSTT V2 provides individual as well as collaborative learning strategies that are incorporated in the way student access the Learning Objects and quizzes. Apart from learning strategy, students are required to complete all Learning Objects assign to them via WReSTT V2. Virtual points are awarded on the successful completion of quizzes. Point would depends upon number of correct answer and time taken to complete particular quiz. All students still need to complete quizzes even if student is part of collaborative strategy. However, quizzes for each team member are generated by selecting appropriate questions from a test bank and point are awarded after last team member (with respect to time), completed his or her quiz.

In this paper we presented three Learning Objects - Unit Test, Data Driven Unit Test and Assert classes of Unit tests. All three Learning Objects is based on Microsoft Unit Test and uses C# as programming language and .Net as platform. Our first Learning Object describes how to create unit tests using Visual Studio (Integrated Development Environment), pattern of creating unit tests, different ways of running unit test, and naming conventions that fallowed in industries. Second Learning Object talk about data driven unit test. Using data driven unit test, user can test all cases just by one unit test by connecting and fetching data directly from data source. Here data source could either be any database or extensible markup language (xml) or comma separated values (csv). Our third and last Learning Objects is about assert classes of unit test. This unit test talk descriptively about all the methods of general assert classes, collection assert classes, and string assert class. Learning Objects are created in such a way that any beginner with no prior knowledge of unit tests could easy understand its concepts and could effectively implement it by its own.

Appendix A contains quizzes for each Learning Objects. All questions within each quiz are multiple choice questions with only one right answer. For references, correct answer with explanation are also given at the end of each question. These correct answers and explanation should be deleted before assigning quizzes to students and it should be used after taking the quizzes to confirm right answers with explanation.

LEARNING OBJECT 1 – UNIT TEST

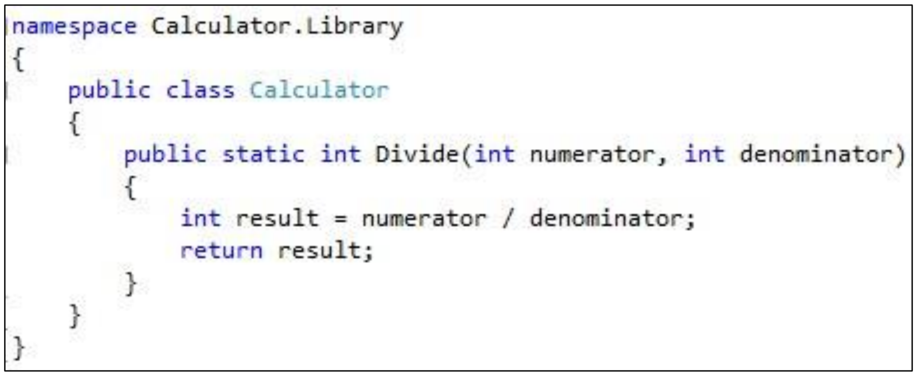
Definition of Unit test - A unit test is a function that tests a unit of work (KudVenkat, 2016) (Osherove, 2012). Let's understand this with the help of example. Right here in Figure 3, is a clipping from very simple web form that accept two numbers – Numerator and Denominator. At this point when we click the Divide Button the page will divide Numerator by Denominator and give the result which is quotient will display in Result.



Numerator	36
Denominator	4
Result	9

Figure 3. Web Form Clip

Here in this example, Figure 3, we given value of Numerator as 36 and Denominator as 4, which gives back the quotient value 9, displayed in Result. To achieve this, we might have divide function like given in Figure 4.



```
namespace Calculator.Library
{
    public class Calculator
    {
        public static int Divide(int numerator, int denominator)
        {
            int result = numerator / denominator;
            return result;
        }
    }
}
```

Figure 4. Divide Method

Here, the name of function is Divide. Notice that this static Divide function accepts two input parameters numerator and denominator of type integer. This function also returning an integer i.e. the quotient which get stored in result. Now to test this function we suppose to write unit tests.

Different Types of Unit Test Framework

There are several unit testing framework available for .net. Few of them are Microsoft (MS) Unit Test, N Unit, Mb Unit, and X unit.net. Each of these framework have their own pros and cons. Most of the organization uses MS Test which is integrated with Visual Studio. This means we don't have to install any third party tools. It is also worth to notice that once we learn to use one of these unit testing frameworks then it would not be too hard to learn the others. Only the way we implement them may be slightly different depending upon the framework we used.

Creating a Unit Test

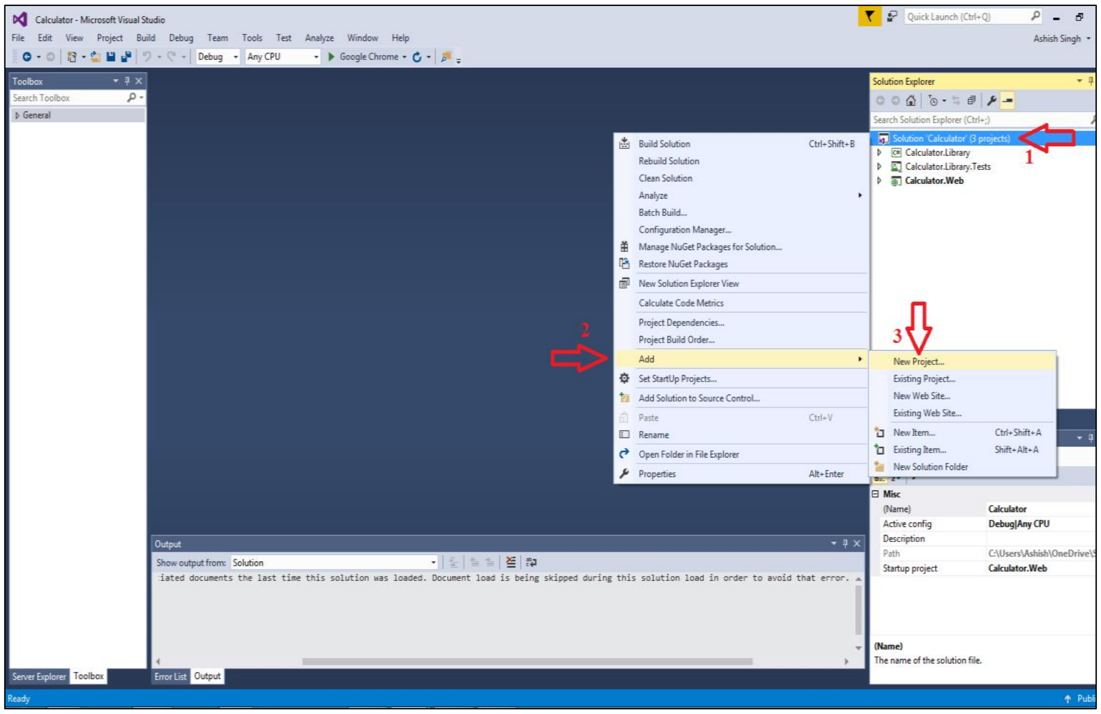
To create a unit test we require to fallow three steps. Firstly, add a unit test project to solution. Secondly, decorate the class that contains test method with [Test Class] attribute. Thirdly, decorate the test method with [Test Method] attribute. Let's talk about each of these step individually.

Add a Unit Test Project to Solution

Adding a unit test project our solution is quite simple. First open Visual Studio (versions 2010, 2012, 2013 or 2015) and then fallows these simple steps.

1. Right Click on solution. A new pop window would appear.
2. Click on "Add".
3. Click on "New Project".

Steps are shown on Figure 5.



```
iated documents the last time this solution was loaded. Document load is being skipped during this solution load in order to avoid that error.
```

Misc
(Name) Calculator
Active config Debug|Any CPU
Description
Path C:\Users\Ashish\OneDrive\
Startup project Calculator.Web

(Name)
The name of the solution file.

Figure 5. Visual Studio

This should open a new window of "Add New Project" as shown on below given figure 6. Now fallow these steps to add a unit test to your project.

1. Click on "Test" Under "Visual C#" on left side. It will give option of "Unit Test Project".
2. Click on Unit Test Project as shown in figure 6.
3. Click on browse to provide the location for unit test project. Although one can give any desired location but it is recommended to save this project on the same solution folder.
4. Provide some meaningful name to this project. We will talk more about naming conventions on the later section. For now, you can give project name as "Calculator.Test".
5. Click on "OK" button to create your unit test.

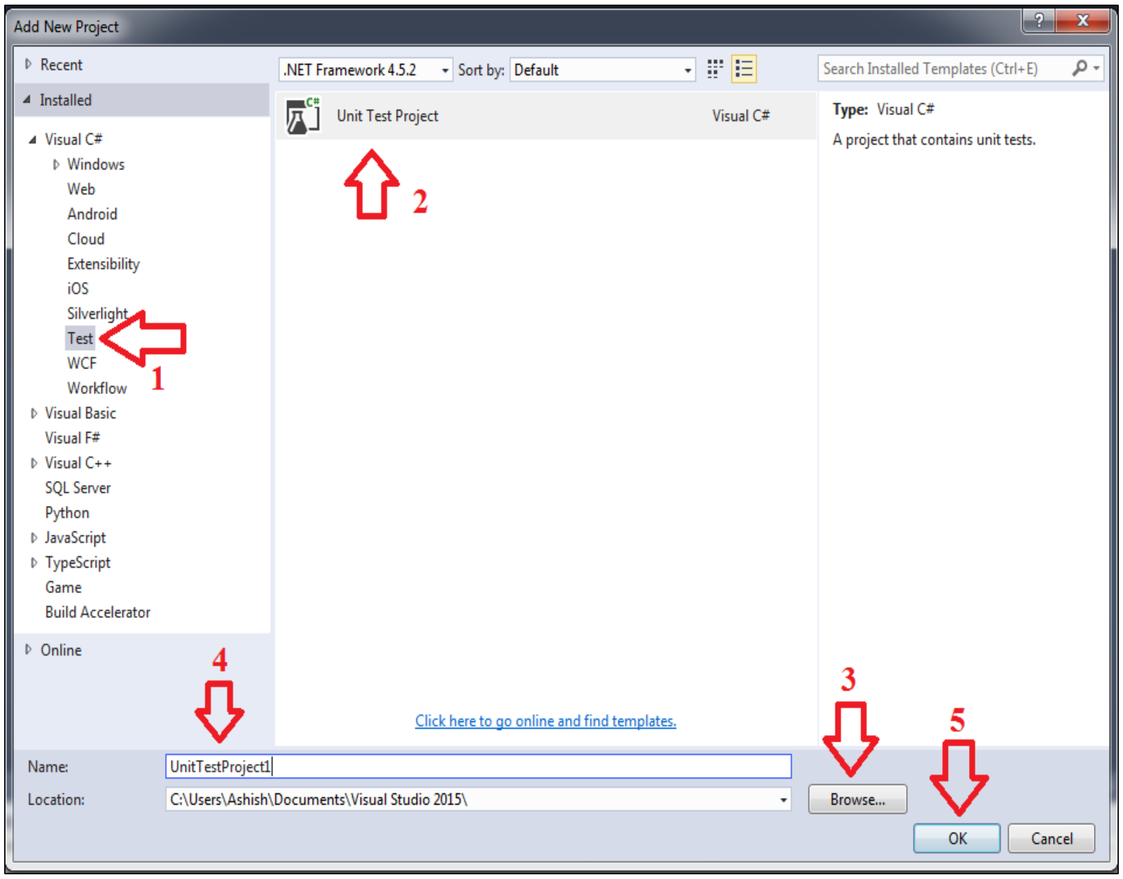


Figure 6. Add New Project

Look at what it has done, it creates a class file “Unit test 1”. Give this file a meaningful name like “CalculatorTest.cs”. It will give a pop -up window to rename reference - click” OK”.

Decorate the Class That Contains Test Method with [Test Class] Attribute

Above procedures should add a unit test project to our solution. If you look at “CalculatorTest.cs” class, visual studio decorated it with [Test Class] attribute. This attribute confirms to the compiler that this is a test class and it going to contain test method.

Decorate the Test Method with [Test Method] Attribute

Test Class contain test methods and all test method within it should be decorated with [Test Method] attribute. In this example, visual studio by default give the name of test method as TestMethod1() which is not a meaningful name in this case. Here we are going to use this method

(TestMethod1()) to test the divide method so for now to keep things simple we can name it as Test_Divide(). There are naming conventions that people fallow to name their unit test when develop in real world applications. We will discuss those naming conventions in later sections.

Pattern to Create Unit Test

There are certain patterns to create unit test and among them the most common, easy and effective pattern is AAA (Arrange Act Assert) (Osherove, 2012). This will divide unit test method into fallowing three sections.

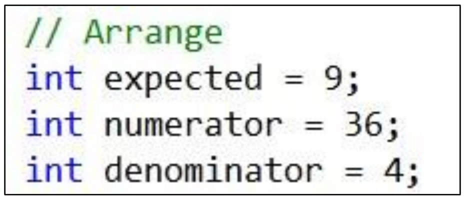
Arrange - Initializes objects and set the value of the data that is passed to the method being tested.

Act – Invokes the method being tested.

Assert – Verifies that the method being tested behaves as expected.

We will now learn about this pattern by implementing it in our “Test_Divide()” unit test.

Arrange - As we know that arrange section initializes the objects and variables that we want to pass that is being tested, so we created an integer variable named as “expected” and assign the value of 9. We also declare two more integer variable as “numerator” and “denominator” as initialize them the value of 36 and 4 respectively, as shown in Figure 7. That’s concludes us to arrange sections.



```
// Arrange
int expected = 9;
int numerator = 36;
int denominator = 4;
```

Figure 7. Arrange Section

Act - Now within our act section we are going to invoke our method that we want to test. Method that we want to test is actually present in Calculator.Library project and our unit test is present in Calculator.Test project. So at this moment, within Calculator.Test project we haven't added a reference of Calculator.Library project, which means divide method is not available in Calculator.Test. So to add reference to Calculator.Library project, go to Calculator.Test project, right click on References and select Add Reference. This should open a window named as Reference Manager (as shown in Figure 8).

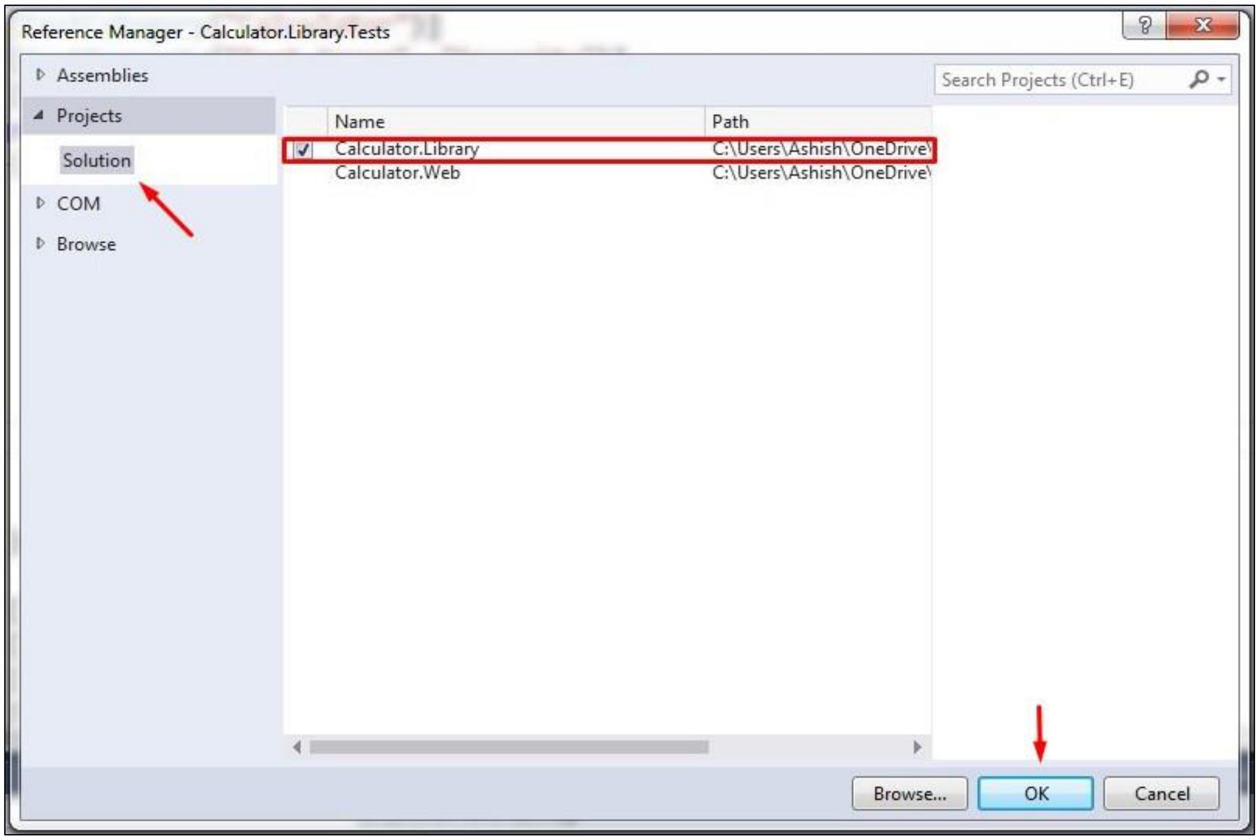
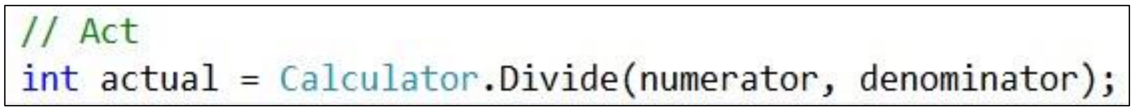


Figure 8. Reference Manager

Select Project then Solution and this should have Calculator.Library project, click on it and select OK. So that going to add a reference to our "Calculator.Library" project.

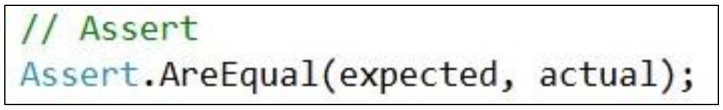
Now within act section we will call our static "Divide()" method which we want to check and pass the value of "numerator" and "denominator" as arguments. Now we store this value in another integer variable name as actual, as shown in Figure 9. We called this variable as actual because it going to store actual value when we divide numerator and denominator and we have the value that we are expecting in expected integer.



```
// Act
int actual = Calculator.Divide(numerator, denominator);
```

Figure 9. Act Section

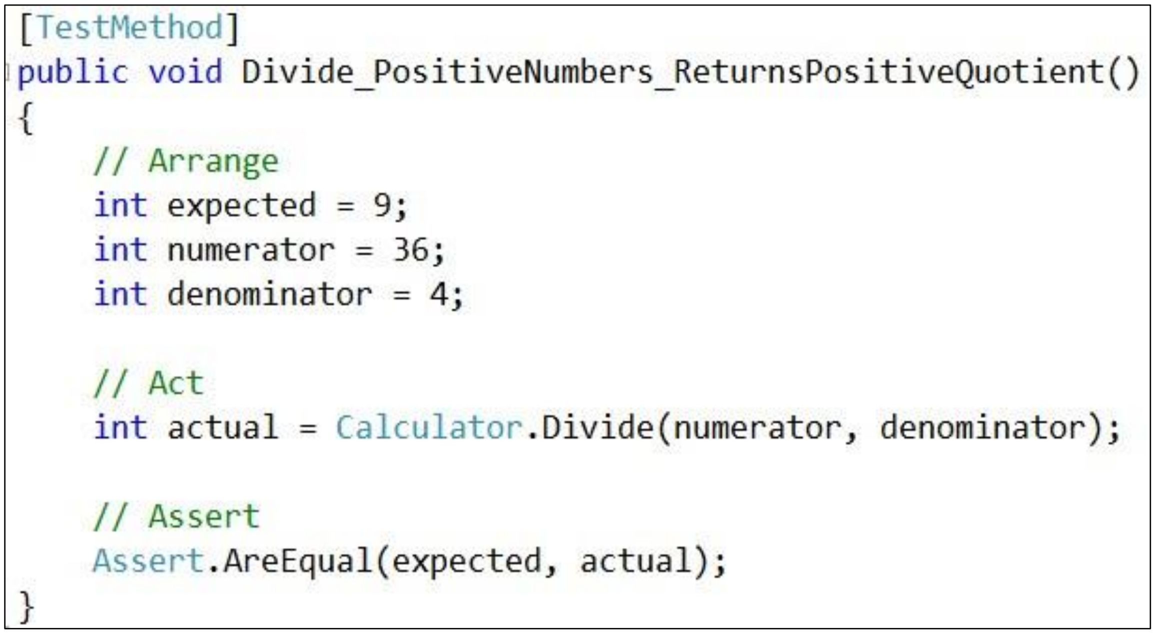
Assert - So finally within this section we are going to verify if the method behaves as expected. To do so we will use Assert class. Assert class have several static methods like Equals, “AreSame” and “AreNotSame” but here we are going to use “AreEqual” method which check expected value is equal to actual or not, as shown in Figure 10.



```
// Assert
Assert.AreEqual(expected, actual);
```

Figure 10. Assert Section

So overall our unit test will look same as Figure 11.



```
[TestMethod]
public void Divide_PositiveNumbers_ReturnsPositiveQuotient()
{
    // Arrange
    int expected = 9;
    int numerator = 36;
    int denominator = 4;

    // Act
    int actual = Calculator.Divide(numerator, denominator);

    // Assert
    Assert.AreEqual(expected, actual);
}
```

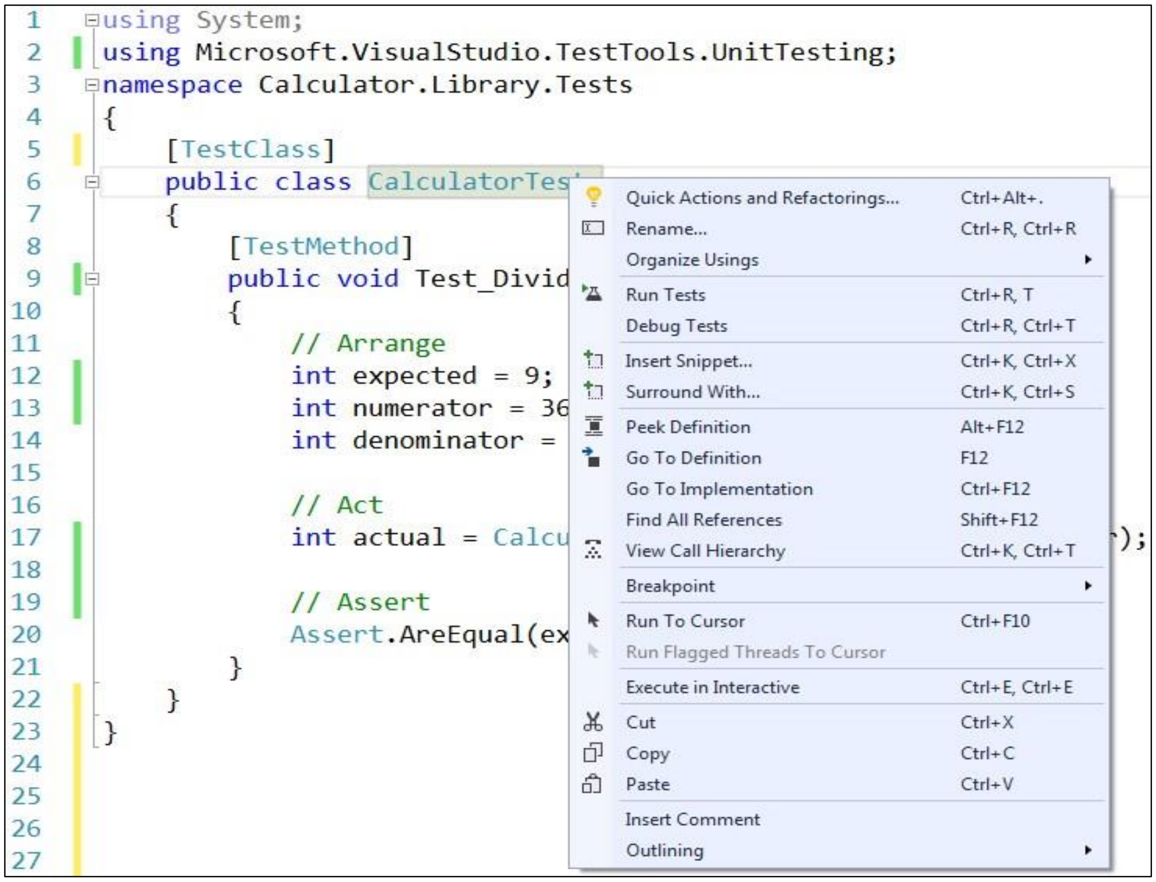
Figure 11. Test_Divide Unit Test

Running a Unit Test

There are different ways to run unit test within visual studio. We can run single unit test, multiple unit tests or all unit tests. Let's explorer those options now.

Running Single Unit Test

Let's first look at the option available to execute single unit test. If we want to execute just one unit-test, we can right click anywhere within that unit test and select run tests to execute that unit test. For example if we want to execute our "Test_Divide()" unit test, we can right click anywhere within "Test_Divide()" and select run tests from the context menu, as shown in Figure 12. This will execute only "Test_Divide()" unit test.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace Calculator.Library.Tests
{
    [TestClass]
    public class CalculatorTes
    {
        [TestMethod]
        public void Test_Divid
        {
            // Arrange
            int expected = 9;
            int numerator = 36
            int denominator =

            // Act
            int actual = Calcu                                      );

            // Assert
            Assert.AreEqual(ex
        }
    }
}
```

Quick Actions and Refactorings... Ctrl+Alt+.
Rename... Ctrl+R, Ctrl+R
Organize Usings
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Execute in Interactive Ctrl+E, Ctrl+E
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Insert Comment
Outlining

Figure 12. Running Single Unit Test Option 1

This will open Test Explorer window which have Passed Tests, Failed Tests and Not Run Tests section. Here in your example our Test_Divide() unit test should come under Passed Tests section. If Test Explorer window does not open automatically then you can open it manually by selecting Run>>Windows>>Test Explorer.

This will open Test Explorer window which have Passed Tests, Failed Tests and Not Run Tests section. The “Passed Tests’ option is shown in Figure 13.

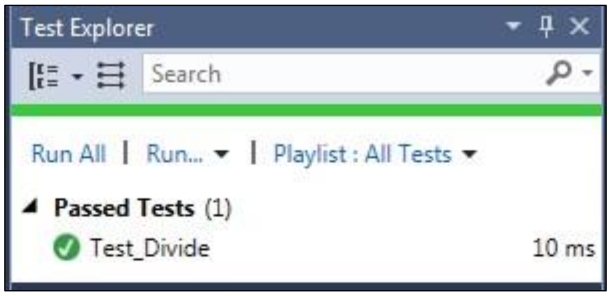
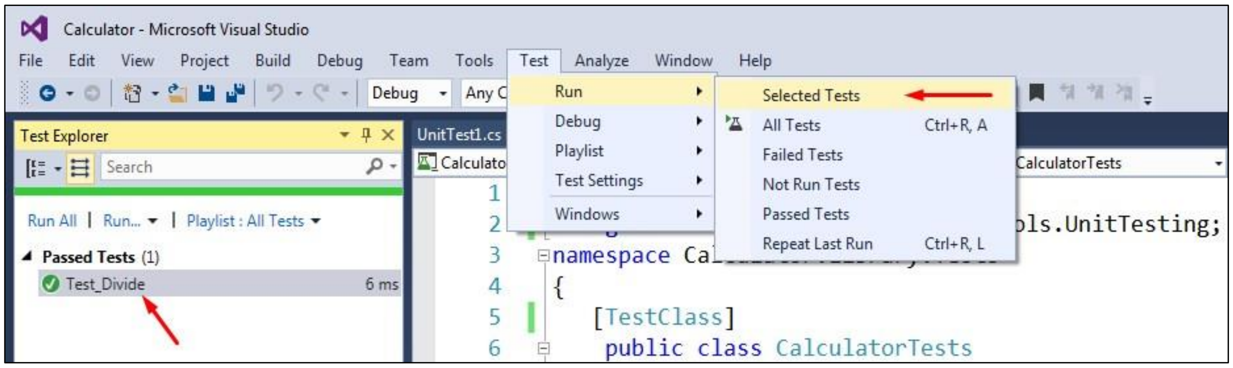


Figure 13. Test Explorer

The other option to run single unit test is to select the desired unit test within the Test Explorer window and then select Run (from main menu) then Selected Test as shown in Figure 14.



```
ols.UnitTesting;
namespace Ca
{
    [TestClass]
     public class CalculatorTests
```

Figure 14. Running Single Unit Test Option 2

The third option to run single unit test is to right click on single unit test within “Test Explorer” and select “Run Selected Tests” as given in Figure 15.

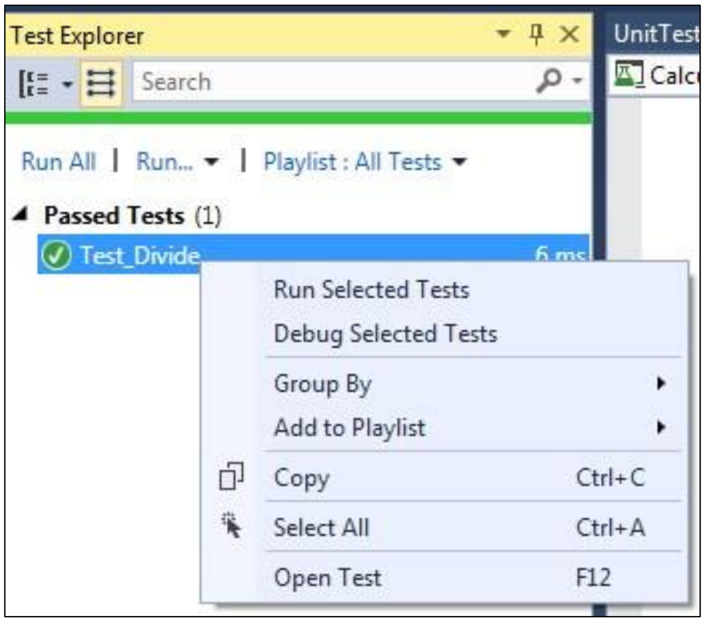
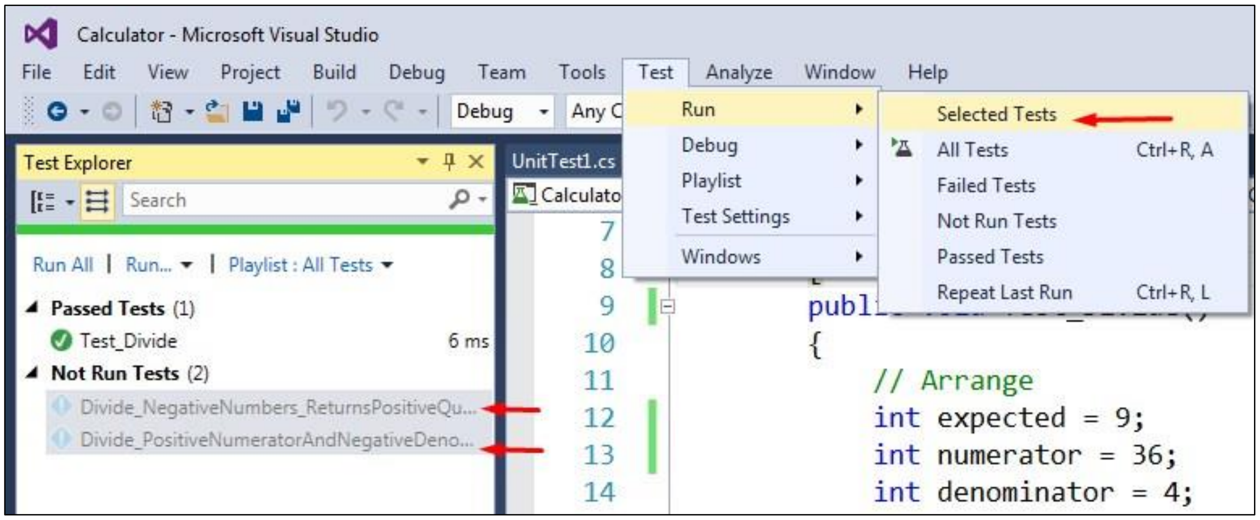


Figure 15. Running Single Unit Test Option 3

Running Multiple Unit Test

If you want to run multiple selected unit tests, then select multiple unit tests from "Test Explorer" (using ctrl button) then select "Run" (from main menu) then "Selected Test" as shown in Figure 16. This will execute multiple unit tests.


Calculator - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug Any C
Run
Debug
Playlist
Test Settings
Windows
Selected Tests
All Tests Ctrl+R, A
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run Ctrl+R, L
Test Explorer
Search
Run All | Run... | Playlist : All Tests
Passed Tests (1)
Test_Divide 6 ms
Not Run Tests (2)
Divide_NegativeNumbers_ReturnsPositiveQu...
Divide_PositiveNumeratorAndNegativeDeno...
UnitTest1.cs
Calculato

```
publ
{
    // Arrange
    int expected = 9;
    int numerator = 36;
    int denominator = 4;
```

Figure 16. Running Multiple Unit Test Option 1

Another option to run multiple unit tests is to select multiple unit tests (using ctrl button) from the “Test Explorer” then right click on it and select “Run Selected Tests” as given in Figure 17.

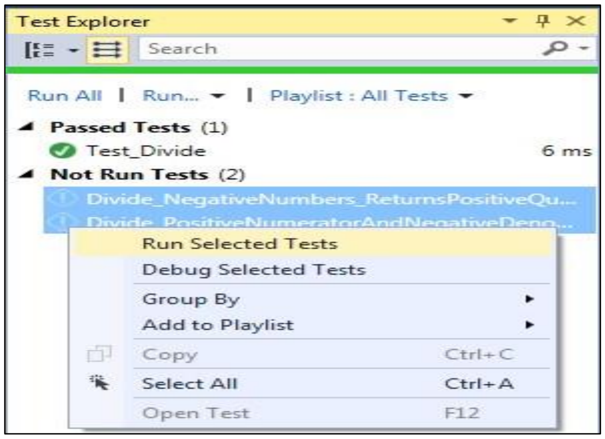
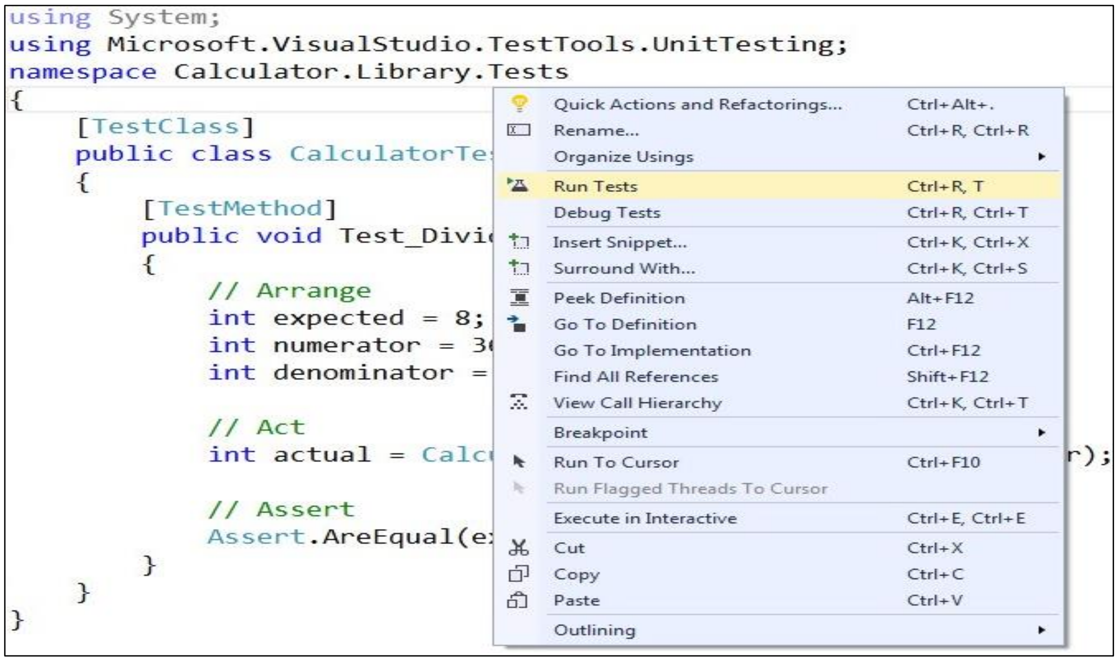


Figure 17. Running Multiple Unit Test Option 2

We also have an option to run all unit tests that are present in one specific test class. At the moment within our project we have all the unit test present within our single class: “CalculatorTests”.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace Calculator.Library.Tests
{
    [TestClass]
    public class CalculatorTe
    {
        [TestMethod]
        public void Test_Divi
        {
            // Arrange
            int expected = 8;
            int numerator = 3
            int denominator =

            // Act
            int actual = Calc                                    r);

            // Assert
            Assert.AreEqual(e
        }
    }
}
```

Quick Actions and Refactorings... Ctrl+Alt+.
Rename... Ctrl+R, Ctrl+R
Organize Usings
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Execute in Interactive Ctrl+E, Ctrl+E
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining

Figure 18. Running Unit Test within Specific Class

So if we want to run all our unit test that present only within “CalculatorTests” class then right click on “CalculatorTests” and from within context menu select Run Tests as shown in figure 18. This will ensure that it will execute only the unit tests that are present in CalculatorTest class but not in any other test class.

Running All Unit Tests

If we want to execute all unit test one of the option is to navigate Test>>Run>>All Tests (as shown in Figure 19).

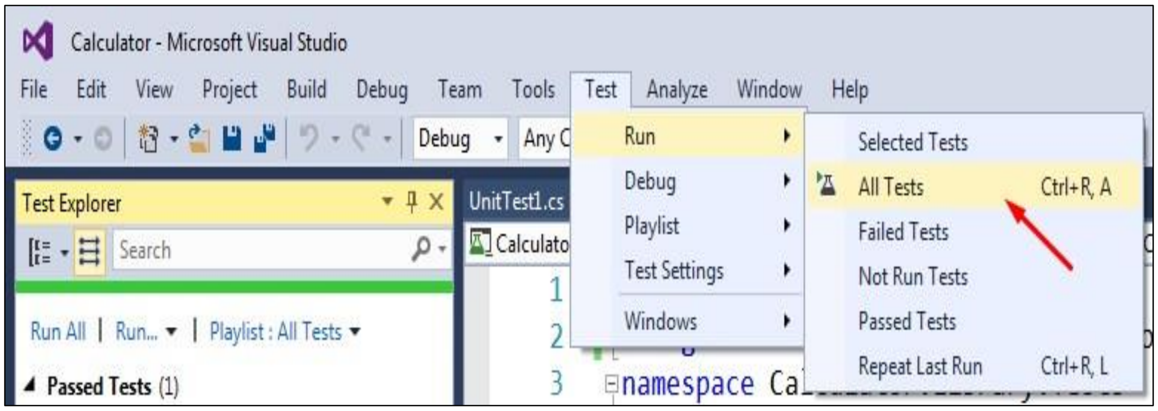


Figure 19. Running All Unit Tests Option 1

Another option is to select “Run All” option within “Test Explorer” window as shown as Figure 20. Fallow these steps to run all unit test at the same time. This will help if you have more than single unit test.

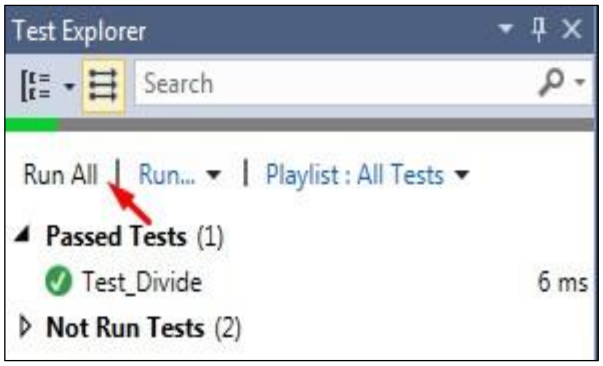
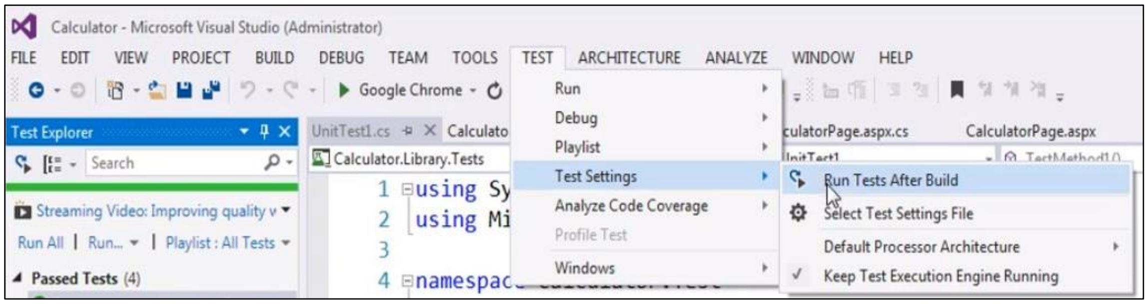


Figure 20. Running All Unit Test Option 2

Running Unit Test Automatically

At the moment we have to manually execute these tests. Now let’s say whenever we build the solution and if the build complete successfully, automatically unit test executes. We can do so by configuring that with visual studio. Navigate as given in Figure 21.


Calculator - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Google Chrome
Run
Debug
Playlist
Test Settings
Analyze Code Coverage
Profile Test
Windows
Run Tests After Build
Select Test Settings File
Default Processor Architecture
Keep Test Execution Engine Running
Test Explorer
Search
Streaming Video: Improving quality
Run All | Run... | Playlist : All Tests
Passed Tests (4)
UnitTest1.cs
Calculator.Library.Tests
culatorPage.aspx.cs
CalculatorPage.aspx

```
using Sy
using Mi

namespac
```

Figure 21. Running Unit Test Automatically

This will ensure that as soon as unit test build successfully unit test get execute automatically.

Naming Conventions

The naming conventions that we used to name unit tests are very important. We should make sure that we are using same naming conventions for all unit tests. A good unit test name provides all the important information about that tests. There are various naming conventions used by programmers and one of the most efficient is “Roy Osherove’s” naming convention for unit tests, which define as fallow

[UnitOfWork_StateUnderTest_ExpectedBehaviour]

“UnitOfWork” is name of the method being tested, “StateUnderTest” represent the input values for the method and ExpectedBehaviour is what the method returns for the specified input. Let’s understand it by renaming our “Test_Divide()” unit test method to stick to Roy Osherove’s naming strategy. According to “Roy Osherove’s” naming convention, first part of unit test name is “UnitOfWork” which is nothing but the name of the method that we testing, here we are testing divide method so “UnitOfWork” is “Divide”. Second part of Roy Osherove’s naming conventions is “StateUnderTest” which represent input value the method that we testing (Osherove, 2012). So if you at the “Test_Divide” method described in Figure 3 it has got two input parameters numerator and denominator and if you look the values of these two parameters both of them are positive numbers, so “StateUnderTest” will be “PositiveNumbers”. The final part of unit test name is “ExpectedBhaviour”, so when we divide two numbers we will get positive quotient, so “ExpectedBhaviour” will be “ReturnsPositiveQuotient”. So the name of “Test_Divide()” unit test will be, “Divide_PositiveNumbers_ReturnsPositiveQuotient()” as shown in Figure 22.

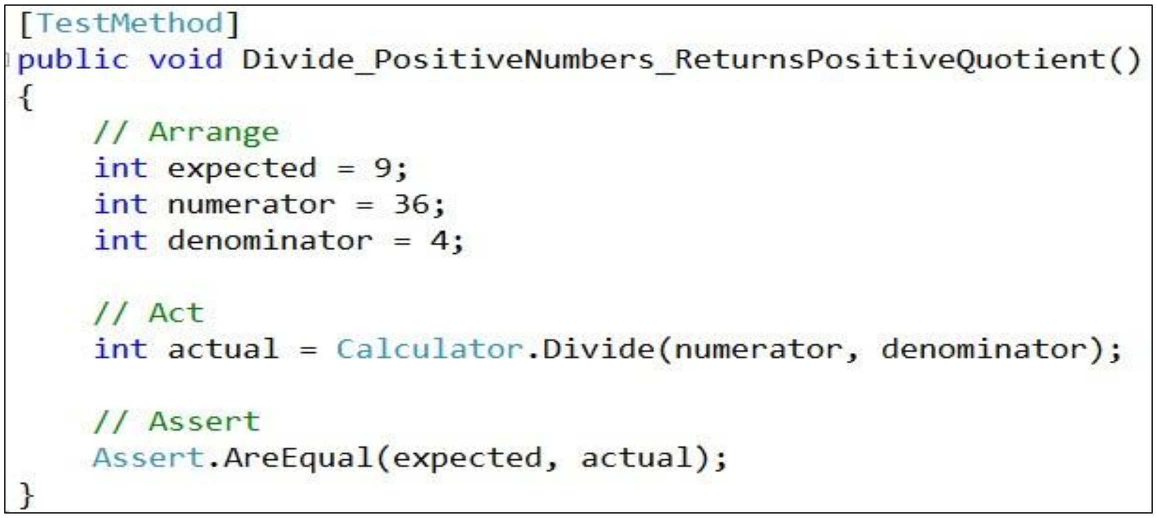
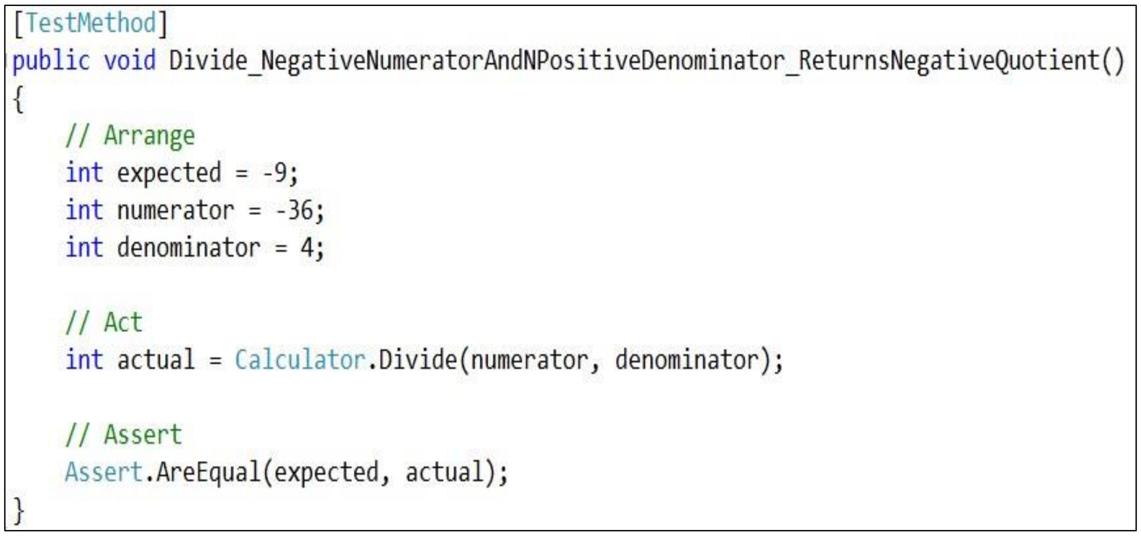

```
[TestMethod]
public void Divide_PositiveNumbers_ReturnsPositiveQuotient()
{
    // Arrange
    int expected = 9;
    int numerator = 36;
    int denominator = 4;

    // Act
    int actual = Calculator.Divide(numerator, denominator);

    // Assert
    Assert.AreEqual(expected, actual);
}
```

Figure 22. Roy Osherove's Naming Convention Example 1

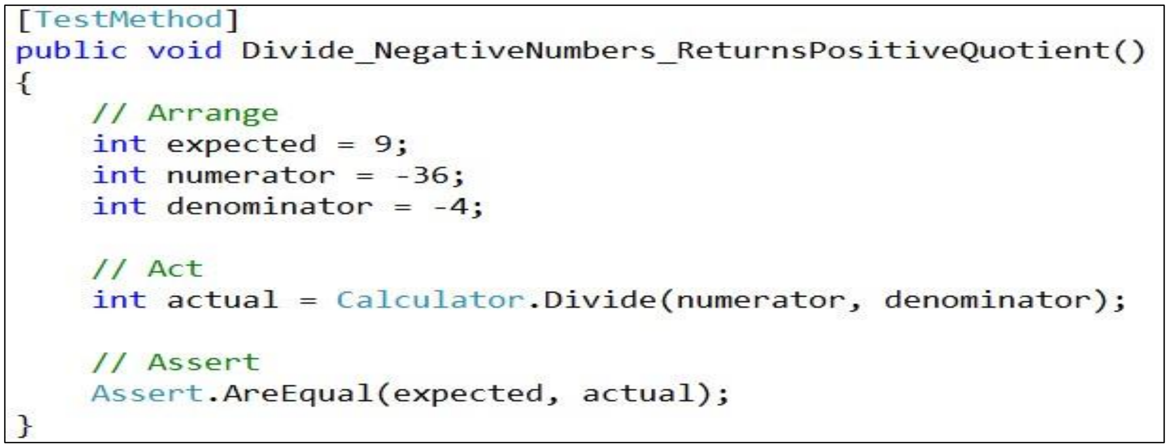
Let’s create another unit test and this time let’s divide a negative numerator with positive denominator so that we get a negative quotient. So this time in accordance with Roy Osherove’s naming convention name of our unit test would be same as shown in Figure 23, Divide_NegativeNumeratorAndPositiveDenominator_ReturnsNegativeQuotient.



```
[TestMethod]
public void Divide_NegativeNumeratorAndNPositiveDenominator_ReturnsNegativeQuotient()
{
    // Arrange
    int expected = -9;
    int numerator = -36;
    int denominator = 4;

    // Act
    int actual = Calculator.Divide(numerator, denominator);

    // Assert
    Assert.AreEqual(expected, actual);
}
```

Figure 23. Roy Osherove's Naming Convention Example 2

Let’s create one more unit test and this time we divide negative numerator with negative denominator which returns positive quotient which make name of our unit test as Divide_NegativeNumbers_ReturnsPostiveQuotient, shown in Figure 24.



```
[TestMethod]
public void Divide_NegativeNumbers_ReturnsPositiveQuotient()
{
    // Arrange
    int expected = 9;
    int numerator = -36;
    int denominator = -4;

    // Act
    int actual = Calculator.Divide(numerator, denominator);

    // Assert
    Assert.AreEqual(expected, actual);
}
```

Figure 24. Roy Osherove's Naming Convention Example 3

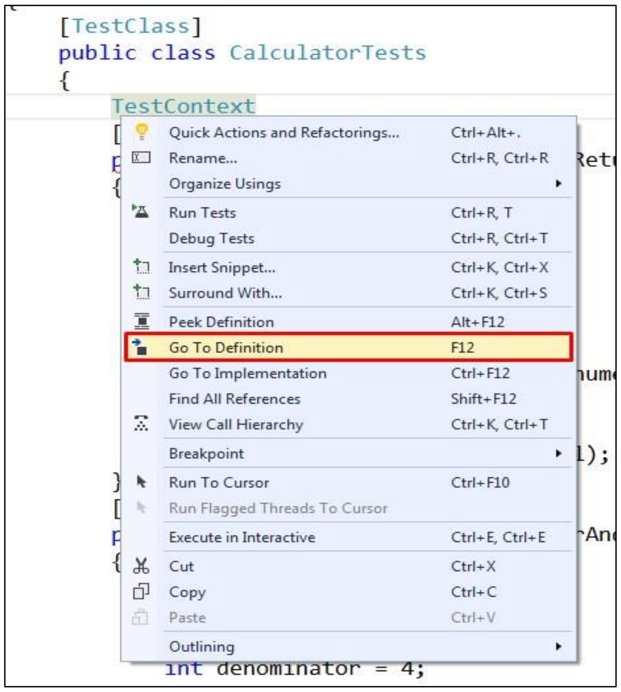
Now for naming conventions in Classes and Projects we simply suffix Test word with it. For example, in our project Calculator class which have “Divide()” we already given the name of class as “CalculatorTest” when we “Test_Divide()” (Netwrok, 2016). Roy Osherove’s naming conventions are quite helpful in providing meaningful name to our method.

LEARNING OBJECT 2 – DATA DRIVEN UNIT TEST

To learn about Data Driven unit test we first need to know about Text Context. Text Context is abstract class which present in “System.Data” namespace. In data-driven unit tests, the TestContext class is required because it provides access to the data row (Network, 2005) (Phlaz, 2012). We define Text context attribute in unit test as [TextContext].

Test Context

If we type [TestContext] in our unit test and right click on it and go to definition (as given in Figure 25), we find that [TextContext] abstract class exposes several class that provide useful information about the current text outcome.

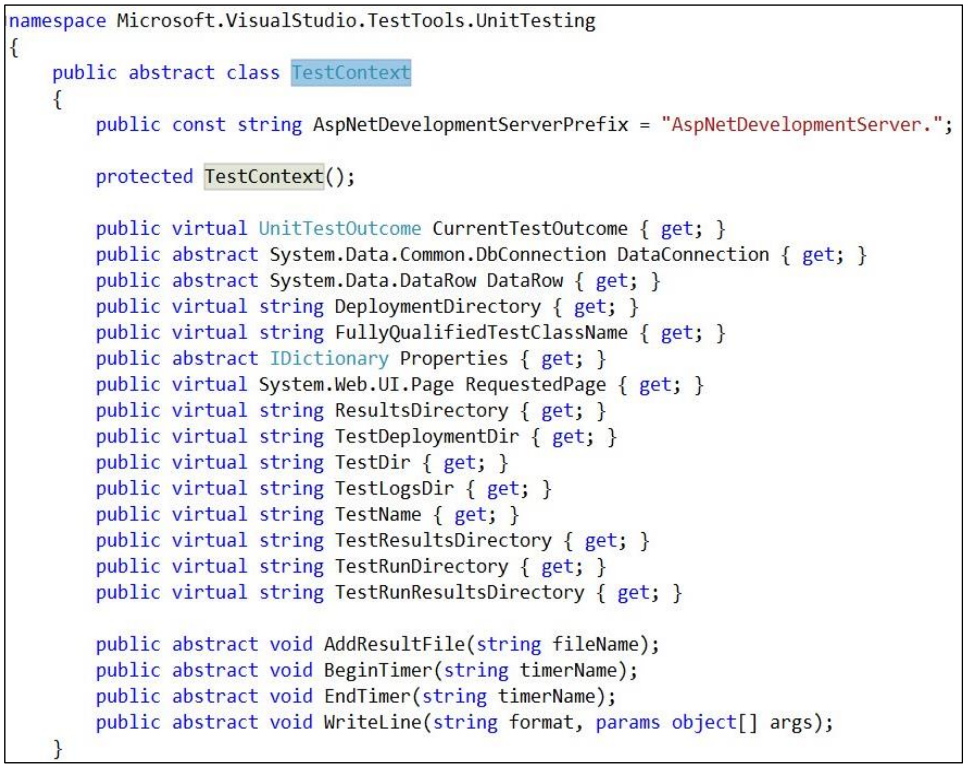


```
[TestClass]
public class CalculatorTests
{
    TestContext
        int denominator = 4;
```

Quick Actions and Refactorings... Ctrl+Alt+.
Rename... Ctrl+R, Ctrl+R
Organize Usings
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Execute in Interactive Ctrl+E, Ctrl+E
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining

Figure 25. TextContext - Go to Definition

One of those property is “CurrentTextOutcome” which tell that either current unit test gets passed, failed, inconclusive, inprogress, error, and timeout or aborted. “DataConnect” and “DataRow” is Data Driven properties which we talk about later sections in great details. For now, we can understand as “DataConnection” property provide information about data connection and “DataRow” property provide access to data row that being tested (Phlaz, 2012) (MSDN, 2015). We also have several other properties which tell about directories. Like “DeploymentDirectory” provide information about deployment directories. “FullyQualifiedTestClassName” property as the name suggested, provide name of the test class that contains the unit test that being executed. “RequestedPage” property provides us information about page object if unit test is asp unit test. “TestName” property provides the name of the unit test that being executed. All property of “TextContext” is provided below in Figure 26.



```
namespace Microsoft.VisualStudio.TestTools.UnitTesting
{
    public abstract class TestContext
    {
        public const string AspNetDevelopmentServerPrefix = "AspNetDevelopmentServer.";

        protected TestContext();

        public virtual UnitTestOutcome CurrentTestOutcome { get; }
        public abstract System.Data.Common.DbConnection DataConnection { get; }
        public abstract System.Data.DataRow DataRow { get; }
        public virtual string DeploymentDirectory { get; }
        public virtual string FullyQualifiedTestClassName { get; }
        public abstract IDictionary Properties { get; }
        public virtual System.Web.UI.Page RequestedPage { get; }
        public virtual string ResultsDirectory { get; }
        public virtual string TestDeploymentDir { get; }
        public virtual string TestDir { get; }
        public virtual string TestLogsDir { get; }
        public virtual string TestName { get; }
        public virtual string TestResultsDirectory { get; }
        public virtual string TestRunDirectory { get; }
        public virtual string TestRunResultsDirectory { get; }

        public abstract void AddResultFile(string fileName);
        public abstract void BeginTimer(string timerName);
        public abstract void EndTimer(string timerName);
        public abstract void WriteLine(string format, params object[] args);
    }
```

Figure 26. TestContext Properties

Now our task is to add instance of TestContext class in our unit test which would ultimately help in accessing row but as we know that TestContext is abstract class and because of this reason we cannot create its object. So next question is how to get instance of TestContext class. To get the instance of TestContext class we need to create public property that returns TestContext as shown in Figure 27.

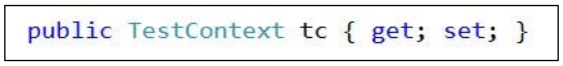

```
public TestContext tc { get; set; }
```

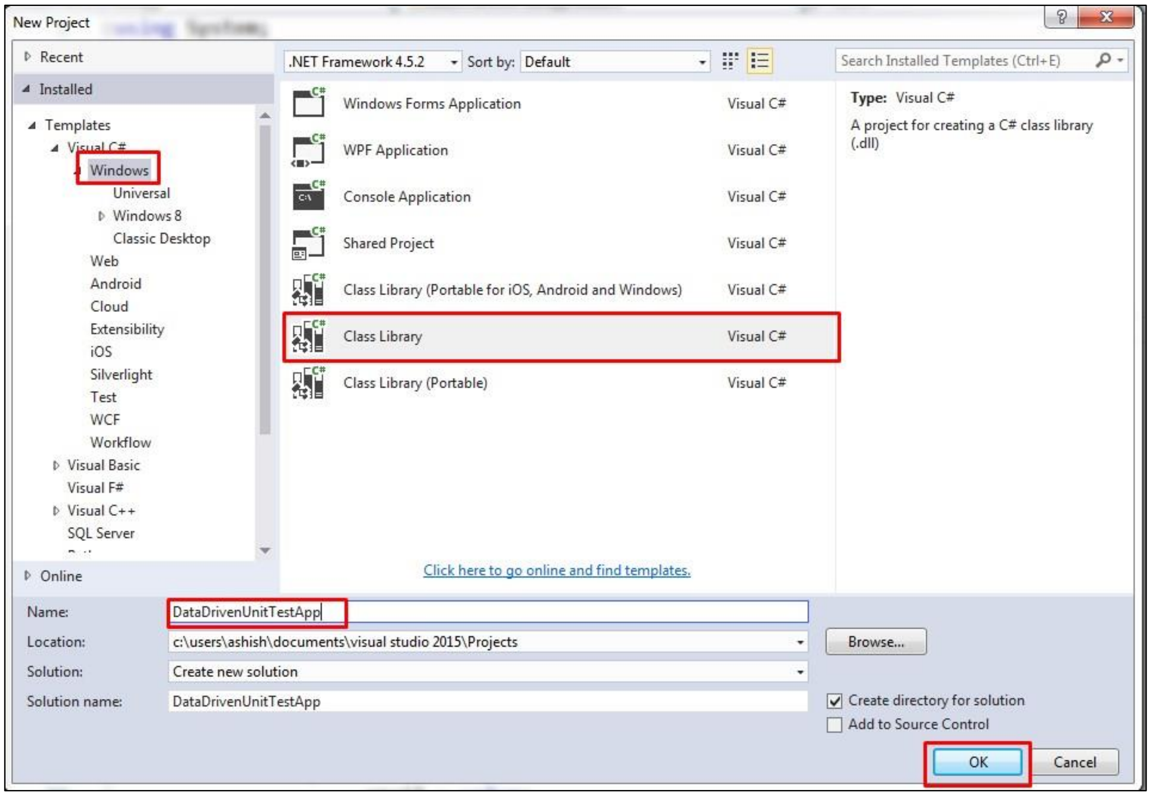
Figure 27. Instance of TestContext

That’s all you have to do to get the instance of TestContext. When execute any unit text within this class we will automatically get instance of TestContext class and we can access that using ‘tc’ property. Now we can use ‘tc’ instance to retrieve information like unit test that being currently executed, the fully classified class name in which that unit test is present and the outcome to unit test whether it has passed, failed or aborted. Any Unit test method which is decorated with [TestCleanup] attribute will run immediately after every unit test within that class. Only one method within Test class can have [TestCleanup] method.

Data-Driven Unit Test

Data-driven unit test allows us to use data from data source with the unit test (MSDN, 2015). The unit test method is executed for every row in the data source. This is extremely useful to test variety of input using single unit test method. Let understand the step involve in setting up data driven unit test.

Here we will create a data driven unit test where we will test student’s email and make sure that all student has valid email format. A valid email has @ and period (.) symbol etc. To do so we will create a project using your visual studio. Open visual studio and go to New -> project. It will open a new window, select Class -> Library Project and named it as DataDrivenUnitTestApp as shown as Figure 28.


New Project

Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
Android
Cloud
Extensibility
iOS
Silverlight
Test
WCF
Workflow
Visual Basic
Visual F#
Visual C++
SQL Server
Online

.NET Framework 4.5.2 Sort by: Default

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Shared Project	Visual C#
Class Library (Portable for iOS, Android and Windows)	Visual C#
Class Library	Visual C#
Class Library (Portable)	Visual C#

Click here to go online and find templates.

Search Installed Templates (Ctrl+E)

Type: Visual C#

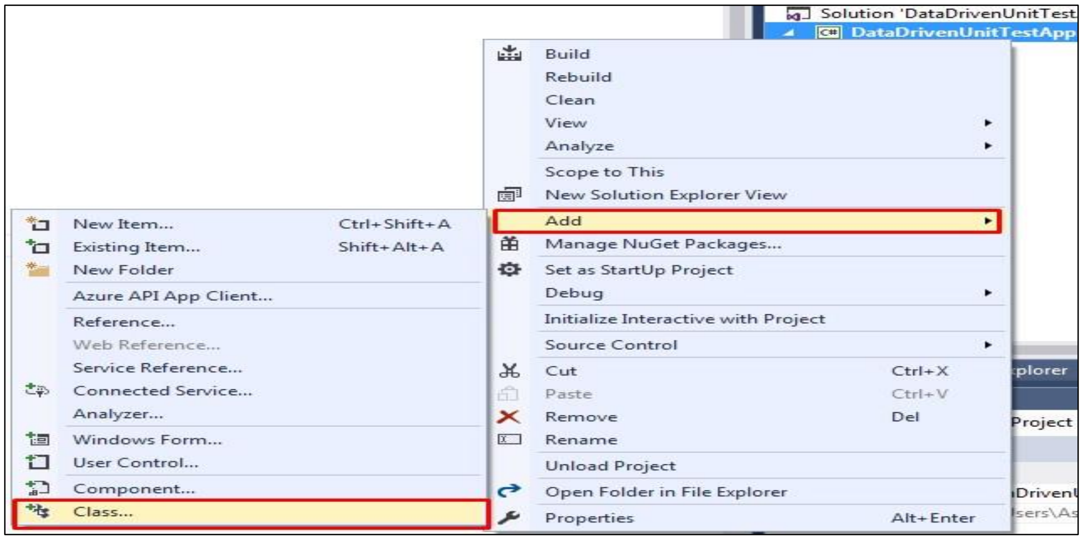
A project for creating a C# class library (.dll)

Name:	DataDrivenUnitTestApp
Location:	c:\users\ashish\documents\visual studio 2015\Projects
Solution:	Create new solution
Solution name:	DataDrivenUnitTestApp

Browse...
[✓] Create directory for solution
[ ] Add to Source Control
OK Cancel

Figure 28. New Class Library Project

Now our next step is to create “cs” file which is actually abbreviation of “CSharp” file. There are many ways to create a “CSharp” file within a “CSharp” project but the steps that we show over here are standard way to create a “CSharp” file this is in effect to skip shortcuts.


Solution 'DataDrivenUnitTest
DataDrivenUnitTestApp

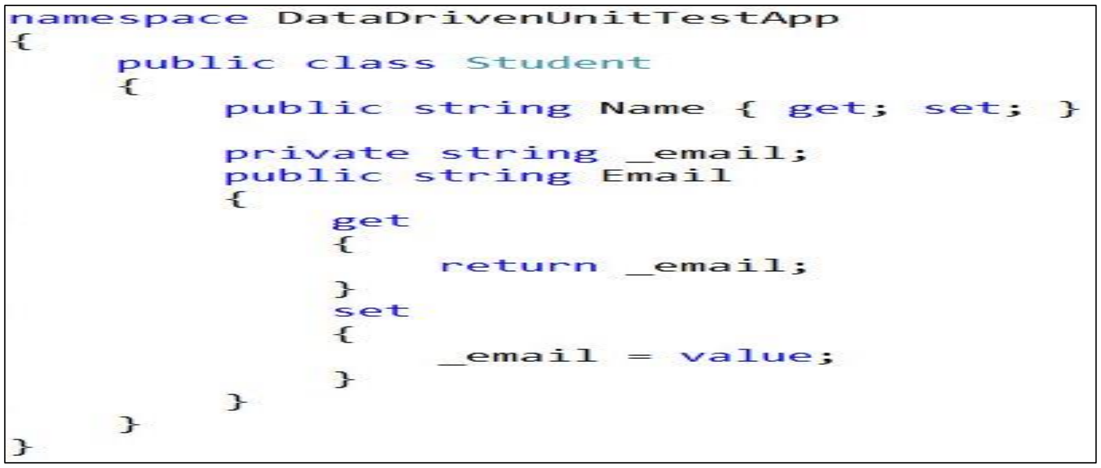
Build	
Rebuild	
Clean	
View	▸
Analyze	▸
Scope to This	
New Solution Explorer View	
Add	▸
Manage NuGet Packages...	
Set as StartUp Project	
Debug	▸
Initialize Interactive with Project	
Source Control	▸
Cut	Ctrl+X
Paste	Ctrl+V
Remove	Del
Rename	
Unload Project	
Open Folder in File Explorer	
Properties	Alt+Enter

New Item...	Ctrl+Shift+A
Existing Item...	Shift+Alt+A
New Folder	
Azure API App Client...	
Reference...	
Web Reference...	
Service Reference...	
Connected Service...	
Analyzer...	
Windows Form...	
User Control...	
Component...	
Class...	

Figure 29. Adding Class Using Context Menu

So to create a “CSharp” file right click on “DataDrivenUnitTestApp” project and from the context menu select “Add” then go to “Class” and select a class file and named it as “student.cs” file. Steps are shown in Figure 29.

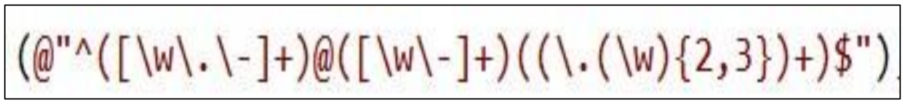
Within this student.cs class, create “Name” and “Email” property. “Name” property will simply have get and set (Figure 30). We will manipulate “Email” property so that it can verify student’s email format.



```
namespace DataDrivenUnitTestApp
{
    public class Student
    {
        public string Name { get; set; }
        private string _email;
        public string Email
        {
            get
            {
                return _email;
            }
            set
            {
                _email = value;
            }
        }
    }
}
```

Figure 30. Student Class with Get and Set Properties

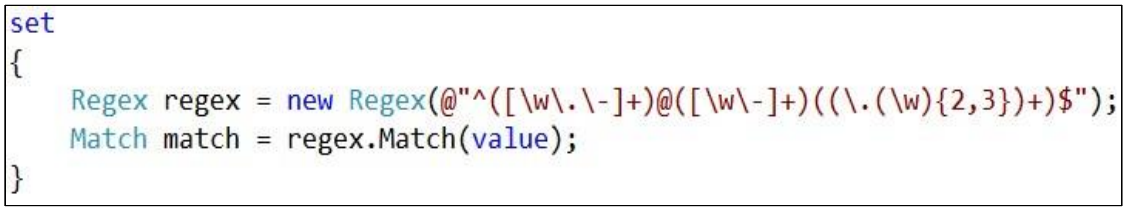
Here “_email” is private field. “Email” is public property where get accessor is simply going to return the value that we have in our private “_email” field. Now we will implement some logic within set accessor. If someone create an instance of “student” class and try to set value for “Email” property, then we want to make sure that format of that email is correct. A valid email should have domain within it, should have @ symbol etc (Regular Expression, 2016). The best way to check if format is correct is by using regular expression like shown in Figure 31.



```
(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$")
```

Figure 31. Regular Expression to Validate Email format

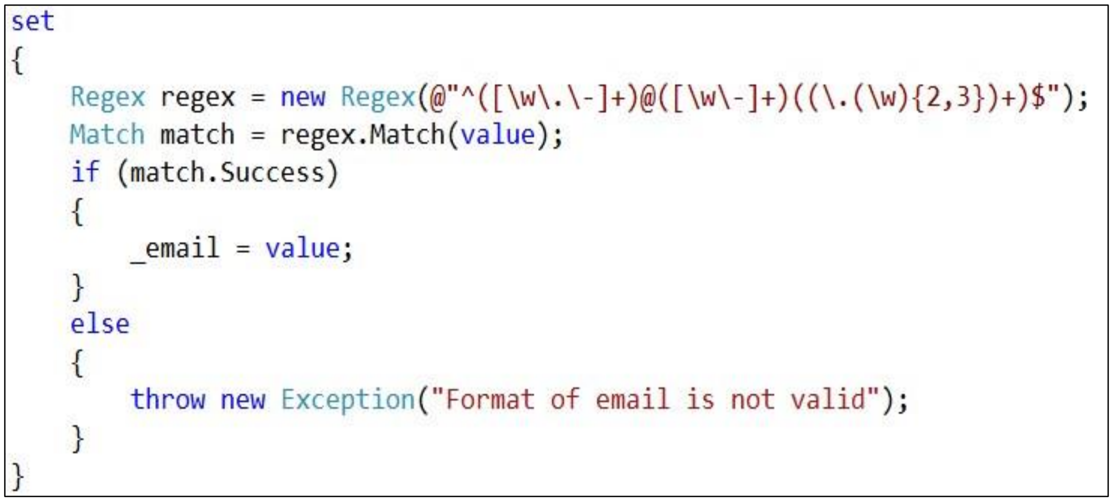
It validates format of the email. So let’s use regular expression class instance. This class is present in “System.Text.RegularExpression” and we set the value as given in Figure 32.



```
set
{
    Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
    Match match = regex.Match(value);
}
```

Figure 32. Set Property with Regular Expression

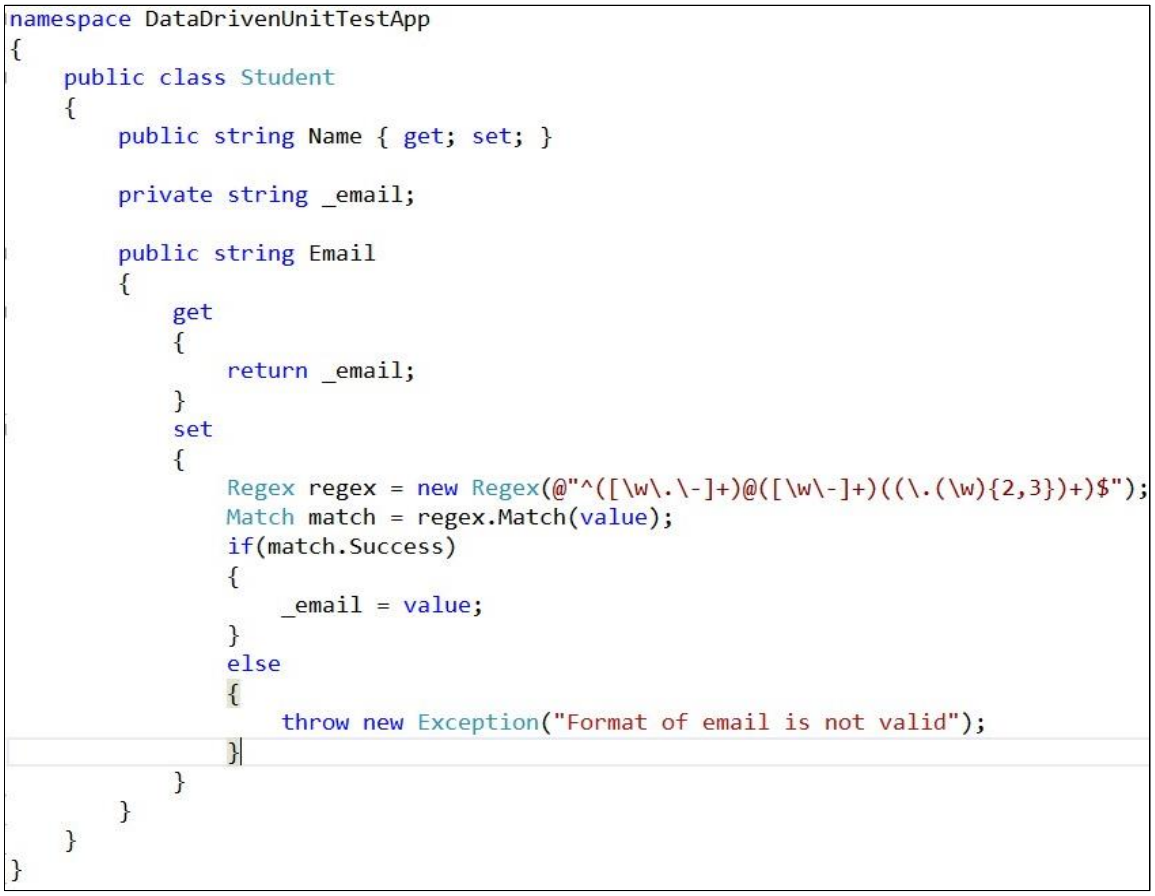
Apart from regular Expression we also created “match” variable of type “Match” which is going to check email format provided by users which would be available in “value”. If match is successful we set the “value” to “_email” else if format of email is not valid then want to throw an exception “Invalid Email Format”. To implement this we could use “match.success” property (which return bool) under if-else condition.



```
set
{
    Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
    Match match = regex.Match(value);
    if (match.Success)
    {
        _email = value;
    }
    else
    {
        throw new Exception("Format of email is not valid");
    }
}
```

Figure 33. Match Property within If-Else Condition

So now if we see our complete code within “student.cs” file would look same as Figure 33. It has “Name” variable with default get and set property. “Email” string variable has default get property but its set property has a regular expression which validates format of provided email address. Don’t be worry if you don’t know about regular expression as they are quite easy to learn and you can find several regular expressions for validating emails if you search it on google.



```
namespace DataDrivenUnitTestApp
{
    public class Student
    {
        public string Name { get; set; }

        private string _email;

        public string Email
        {
            get
            {
                return _email;
            }
            set
            {
                Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
                Match match = regex.Match(value);
                if(match.Success)
                {
                    _email = value;
                }
                else
                {
                    throw new Exception("Format of email is not valid");
                }
            }
        }
    }
}
```

Figure 34. Complete Code of Student Class

So here in Figure 34, we have two very simple properties - name and email and this is what we want to test. Now the next task is to make sure to have data available, so we are going to use database table.

In SQLServer Management Studio we have a database “TestDB”. Within “TestDB” we have a table named as “Student” and within this “Student” table we have 10 rows which contains sample name and email. So this is going to be our test data. Here is SQL script (Figure 35) to create the database, to create the table and populate the database with test data (Technet, 2012).

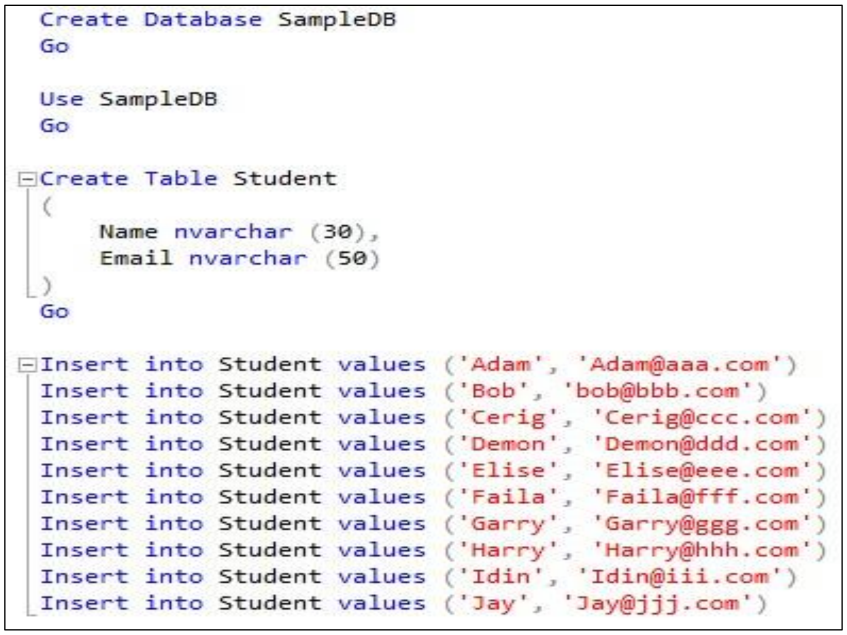

```
Create Database SampleDB
Go

Use SampleDB
Go

Create Table Student
(
    Name nvarchar (30),
    Email nvarchar (50)
)
Go

Insert into Student values ('Adam', 'Adam@aaa.com')
Insert into Student values ('Bob', 'bob@bbb.com')
Insert into Student values ('Cerig', 'Cerig@ccc.com')
Insert into Student values ('Demon', 'Demon@ddd.com')
Insert into Student values ('Elise', 'Elise@eee.com')
Insert into Student values ('Faila', 'Faila@fff.com')
Insert into Student values ('Garry', 'Garry@ggg.com')
Insert into Student values ('Harry', 'Harry@hhh.com')
Insert into Student values ('Idin', 'Idin@iii.com')
Insert into Student values ('Jay', 'Jay@jjj.com')
```

Figure 35. SQL Query for Student Table

This will create "SampleDB" and student table, as shown in Figure 36.

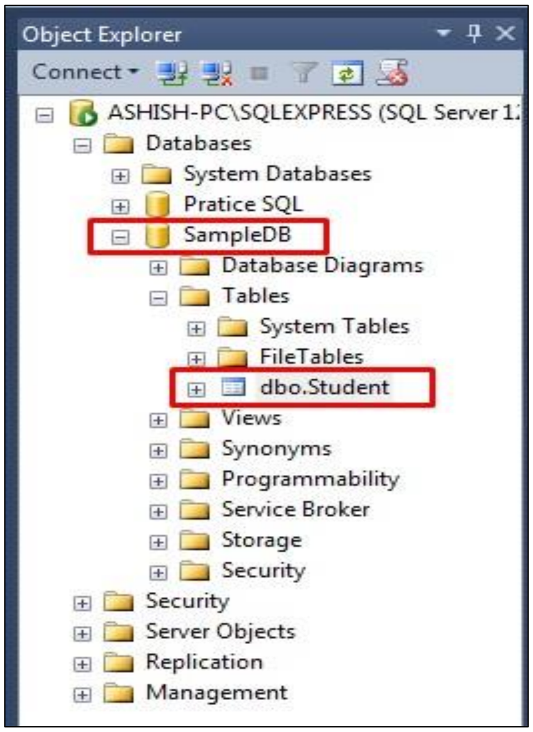
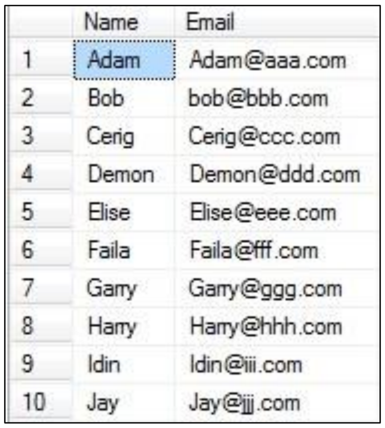


Figure 36. MSSQL Object Explorer for SampleDB

Given script in Figure 35 will create student table with name and email property as shown in below given Figure 37.

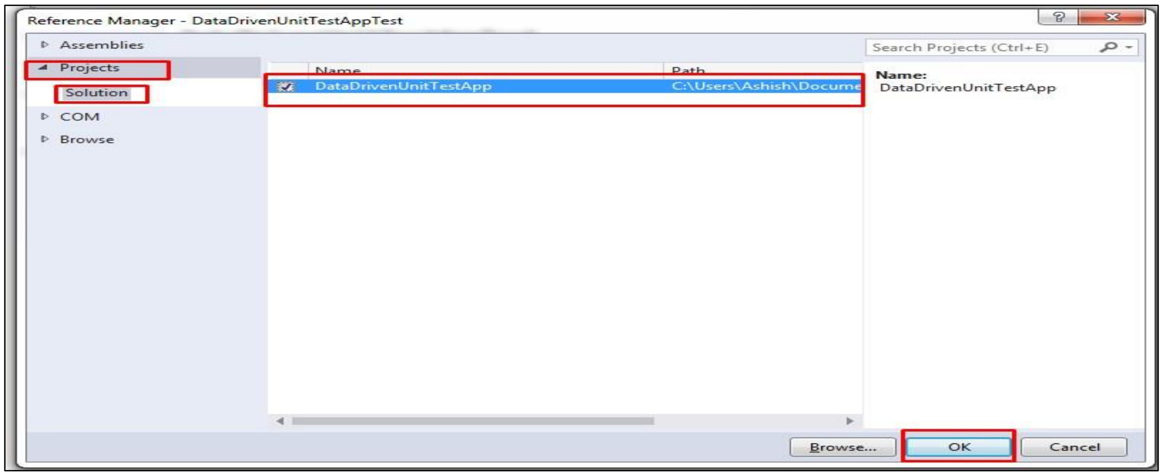


	Name	Email
1	Adam	Adam@aaa.com
2	Bob	bob@bbb.com
3	Cerig	Cerig@ccc.com
4	Demon	Demon@ddd.com
5	Elise	Elise@eee.com
6	Faila	Faila@fff.com
7	Garry	Garry@ggg.com
8	Harry	Harry@hhh.com
9	Idin	Idin@iii.com
10	Jay	Jay@jjj.com

Figure 37. Student Table

The next step is to add a test project to our solution. Right click on the solution and from the context menu select "Add" and go to "New Project". This will open a new window, select "Test" go to "Unit test Project" and give it a meaningful name. In this example the name of our project as "DataDrivenUnitTestApp" so let give this project name as "DataDrivenUnitTestApp.Test".

First thing we have to make sure is to add necessary references. This step is quite easy while using visual studio but sometimes as there are too many options are available within reference section, make user confuse. So here we will try to eliminate this. Here we suppose to add to reference. First we have to add the reference of project of whom we are testing which is, in this case is DataDrivenUnitTestApp. Second reference that we require is data assembly. Now to add these two reference, right click on "Reference" and select "Add Reference". This opens "Reference Manager" solution. Select "Solution" then go to "Project", here you can lists of project. Find and select your test project, which is in this case is DataDrivenUnitTestApp. Steps are given in Figure 38.

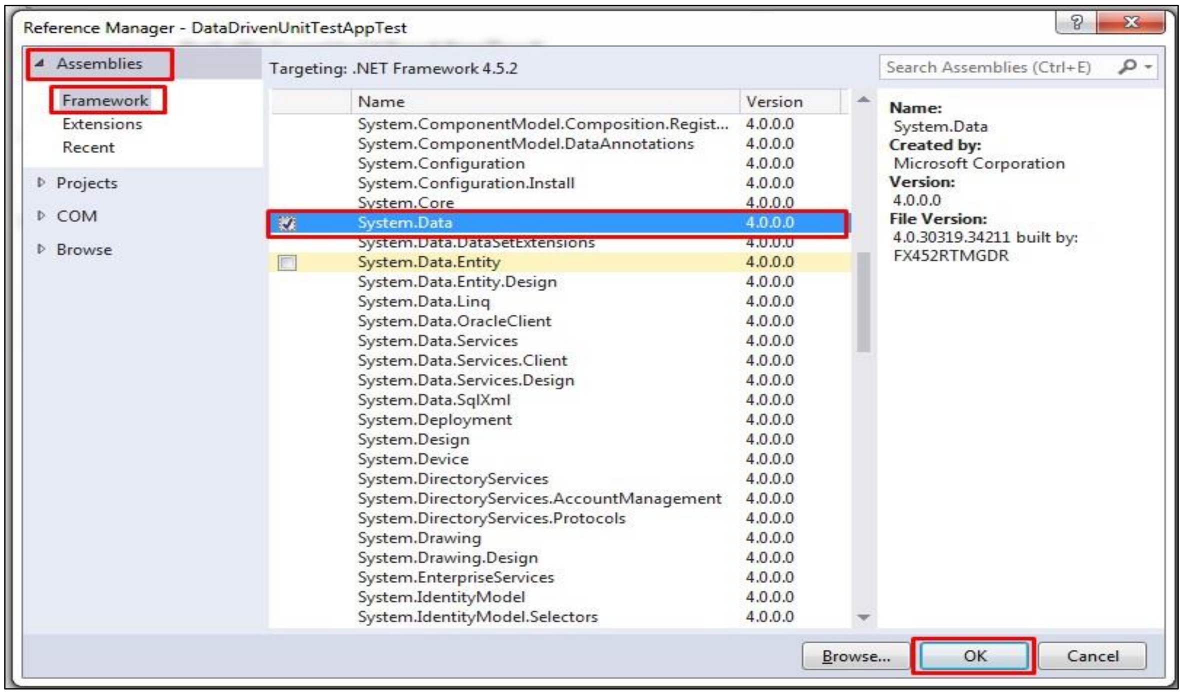


Name	Path
DataDrivenUnitTestApp	C:\Users\Ashish\Docume

Name:
DataDrivenUnitTestApp

Figure 38. Adding Reference of Test Project

Now again go to “Assembly” then “Framework” and select “System.Data” and press OK. This will add these two references in DataDrivenApplication.Test project as shown in Figure 39.


Reference Manager - DataDrivenUnitTestAppTest
Assemblies
Framework
Extensions
Recent
Projects
COM
Browse
Search Assemblies (Ctrl+E)
Browse...
OK
Cancel

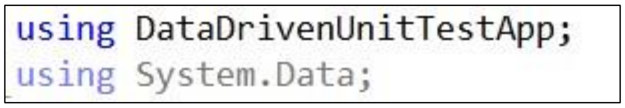
Targeting: .NET Framework 4.5.2

Name	Version
System.ComponentModel.Composition.Regist...	4.0.0.0
System.ComponentModel.DataAnnotations	4.0.0.0
System.Configuration	4.0.0.0
System.Configuration.Install	4.0.0.0
System.Core	4.0.0.0
System.Data	4.0.0.0
System.Data.DataSetExtensions	4.0.0.0
System.Data.Entity	4.0.0.0
System.Data.Entity.Design	4.0.0.0
System.Data.Linq	4.0.0.0
System.Data.OracleClient	4.0.0.0
System.Data.Services	4.0.0.0
System.Data.Services.Client	4.0.0.0
System.Data.Services.Design	4.0.0.0
System.Data.SqlXml	4.0.0.0
System.Deployment	4.0.0.0
System.Design	4.0.0.0
System.Device	4.0.0.0
System.DirectoryServices	4.0.0.0
System.DirectoryServices.AccountManagement	4.0.0.0
System.DirectoryServices.Protocols	4.0.0.0
System.Drawing	4.0.0.0
System.Drawing.Design	4.0.0.0
System.EnterpriseServices	4.0.0.0
System.IdentityModel	4.0.0.0
System.IdentityModel.Selectors	4.0.0.0

Name:
System.Data
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.34211 built by: FX452RTMGDR

Figure 39. Adding Reference of System.Test

To use these two references in studentTest class file we must write using statement as shown in Figure40.



```
using DataDrivenUnitTestApp;
using System.Data;
```

Figure 40. Using Statement

As we have now test project and all required references we can now create data driven unit test. Before doing so let’s give a meaningful name to our class file (instead of it’s by default name UnitTest1.cs). To do so right click on current class file (UnitTest1.cs) and from the context menu select “Rename” (as shown in Figure 41) and provide any name. For this project we give “StudentTest.cs”.

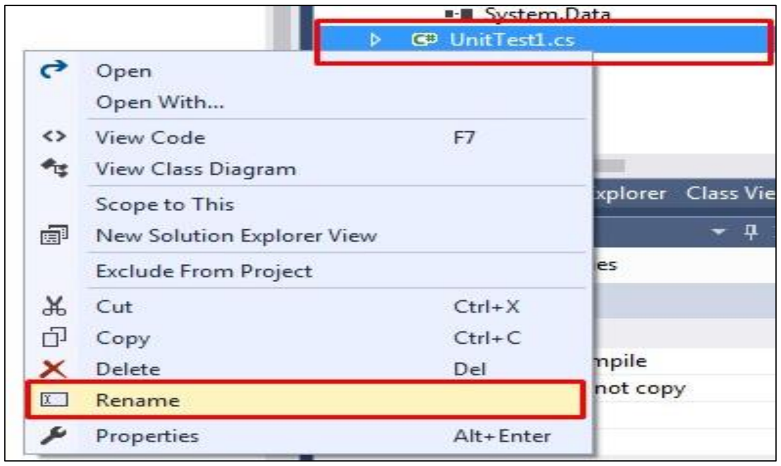
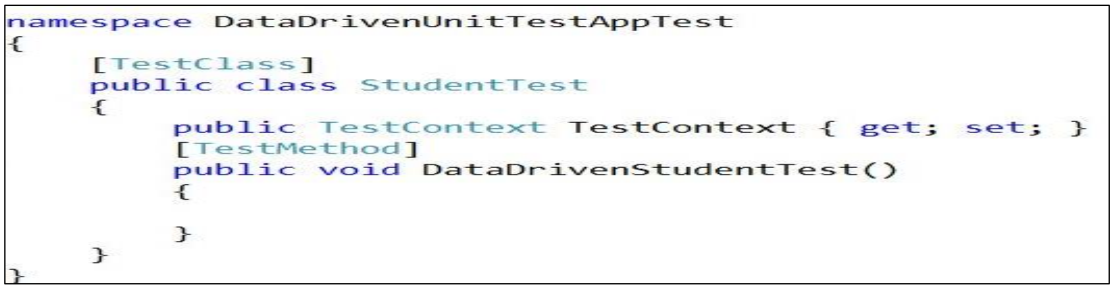


Figure 41. Renaming Class File

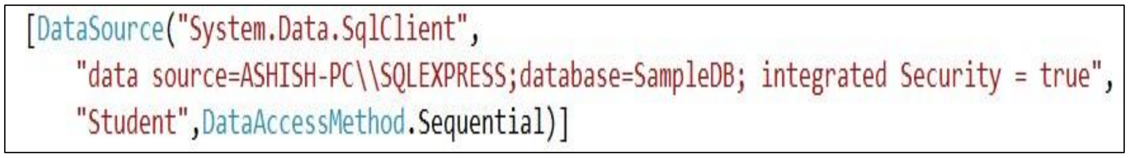
So now we have a TestClass and within this class we have TestMethod. Now let’s include a property for test context class and give this property “TestContext” name. Now our StudentTest class would look same as in Figure 42.



```
namespace DataDrivenUnitTestAppTest
{
    [TestClass]
    public class StudentTest
    {
        public TestContext TestContext { get; set; }
        [TestMethod]
        public void DataDrivenStudentTest()
        {

        }
    }
}
```

Figure 42. Student Test Class

We know that within our “DataDrivenStudentTest” unit test our objective is to use data from Employee table of TestDB database and then call “DataDrivenStudentTest” unit test method for every row (MSDN, 2016) (Stackoverflow, 2012). To do so we must tell our test unit about Employee table and the easiest way to do so is by using DataSource.



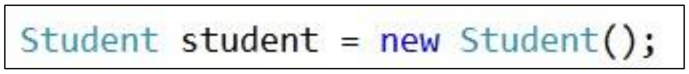
```
[DataSource("System.Data.SqlClient",
    "data source=ASHISH-PC\\SQLEXPRESS;database=SampleDB; integrated Security = true",
    "Student",DataAccessMethod.Sequential)]
```

Figure 43. Data Source

Here in Figure 43, Data Source has four parameters.

1. System.Data.SQlClient - It’s Provider name and it have string as return type.
2. Data source = ASHISH-PC\\ASHISHPC; database=TestDB;integrated security=true” - It’s a ado.net connection string. Second part of this string tell about database which is in this case TestDB. Third part is integrated security which is true.
3. Employee - Its table name.
4. DataAccessMethod.Sequential - Its data access method. It can be sequential or random. Here we selected sequential so that this unit test executed for every row sequentially as the data appear in the data source.

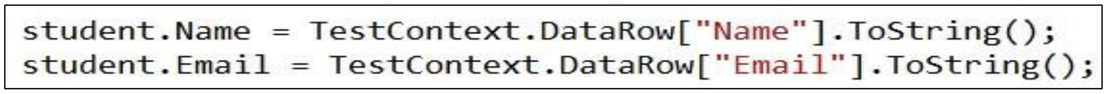
So now our unit test DataDrivenEmployeeTest know from where to pull data. For every row within dataset this unit test will execute. Now let’s create instance of our Student class and name it as student (shown in Figure 44).



```
Student student = new Student();
```

Figure 44. Student Object

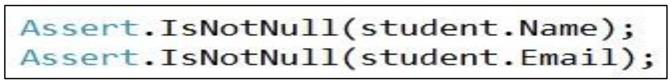
Now in order to populate the name and email property of Student class we will use "DataRow" property of "TextContext". "DataRow" property of "TextContext" retrieve column value of given database table, as shown in Figure 45.



```
student.Name = TestContext.DataRow["Name"].ToString();
student.Email = TestContext.DataRow["Email"].ToString();
```

Figure 45. Using Data Row Property to Populate Name and Email

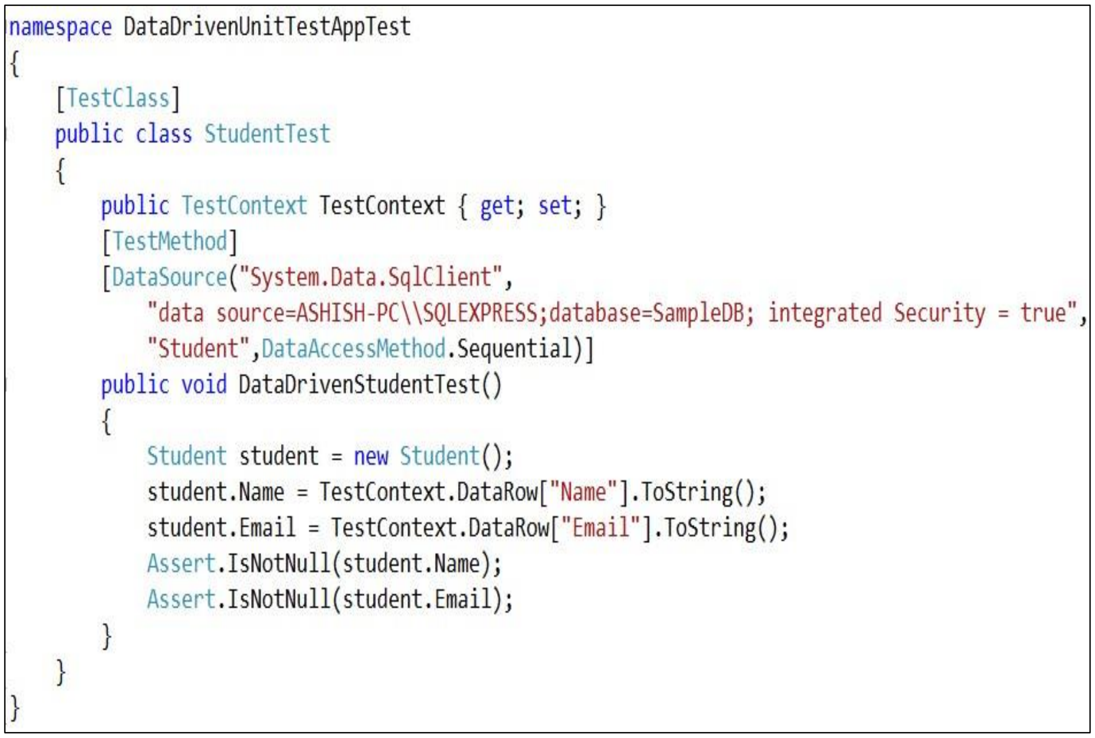
Here, return type of "DataRow" is object and to convert it to string we are using "ToString()" property. Now all we have do here is to check name and email are not null and to do that we will use "Assert.IsNotNull" property as shown in Figure 46.



```
Assert.IsNotNull(student.Name);
Assert.IsNotNull(student.Email);
```

Figure 46. Assert.IsNotNull Property

Overall, complete code of StudentTest class will look like same as in Figure 47.



```
namespace DataDrivenUnitTestAppTest
{
    [TestClass]
    public class StudentTest
    {
        public TestContext TestContext { get; set; }
        [TestMethod]
        [DataSource("System.Data.SqlClient",
            "data source=ASHISH-PC\\SQLEXPRESS;database=SampleDB; integrated Security = true",
            "Student",DataAccessMethod.Sequential)]
        public void DataDrivenStudentTest()
        {
            Student student = new Student();
            student.Name = TestContext.DataRow["Name"].ToString();
            student.Email = TestContext.DataRow["Email"].ToString();
            Assert.IsNotNull(student.Name);
            Assert.IsNotNull(student.Email);
        }
    }
}
```

Figure 47. Complete Code of Student Test

As soon as you build the solution “Test Explorer” should show our unit test - DataDrivenStudenTest. Right click on it and select “Run Selected Test”. Steps are given in Figure 48.

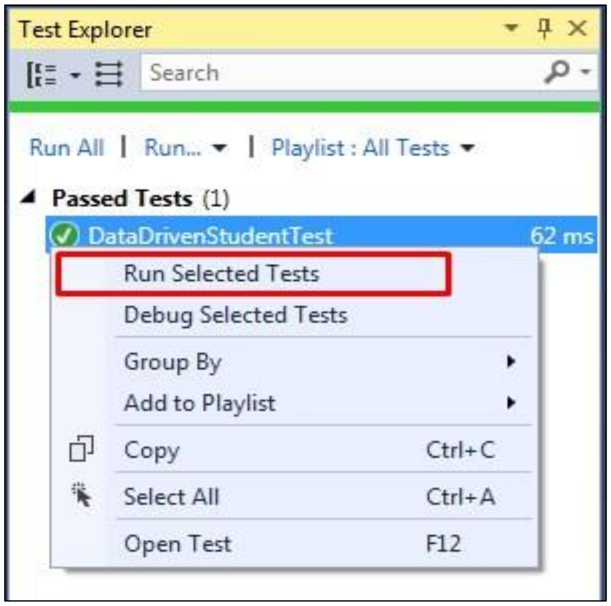


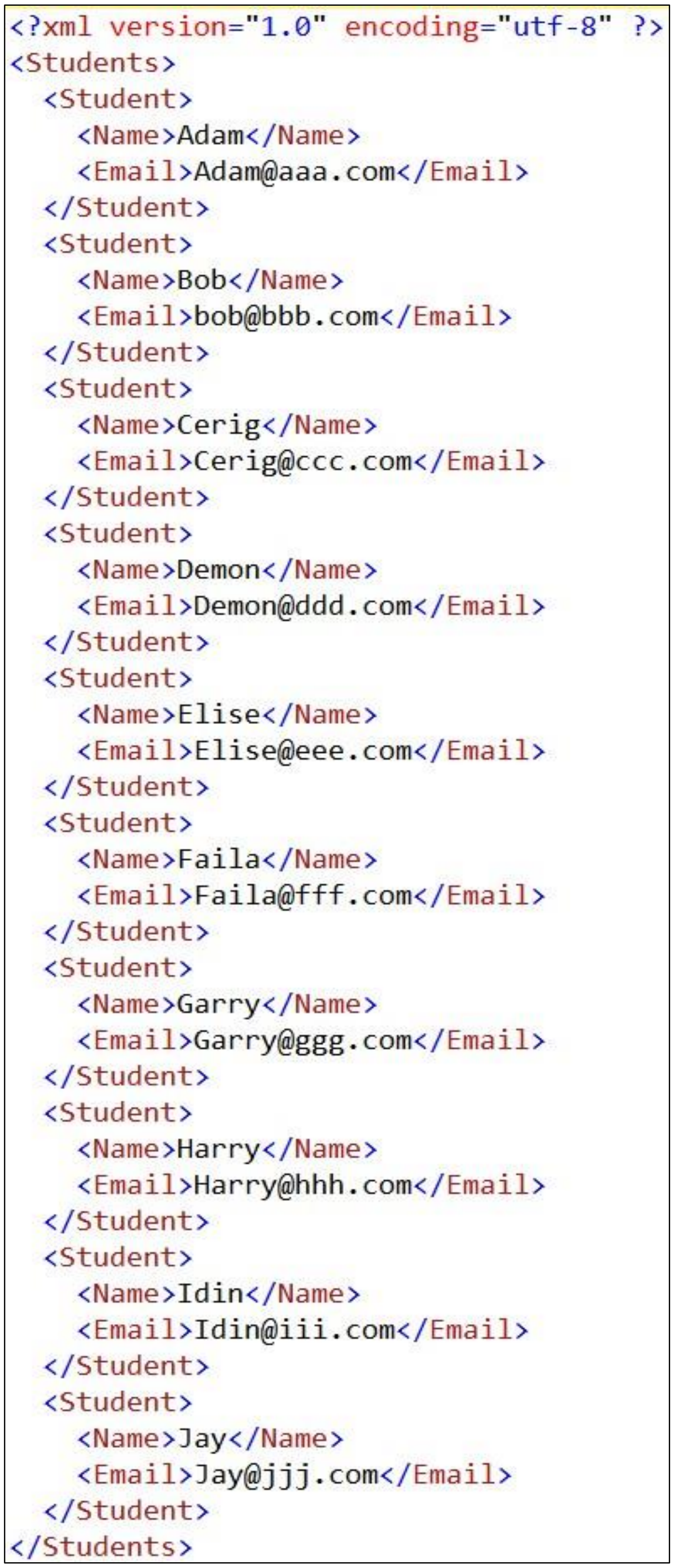
Figure 48. Test Explorer with DataDrivenStudentTest

It will run the test and pass all values of Employee table (from SQL Server DB) to this unit test to check if format is right or not.

Data Driven Unit Test – XML

In our previous section we discuss how to drive the unit test from the data which is present in database table. In this section we will discuss how to drive same unit test from the data which is present is XML file instead of from the database table.

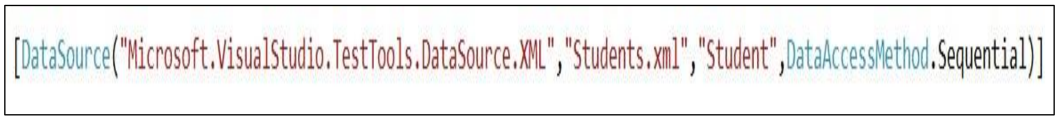
So first step is to add the XML file to our unit test. So for this right click on our Test project DataDrivenUnitTestAppTest and from the context menu navigate to “Add” and select “New Item”. It will open “Add New Item” window, select “Data” then “XML” File and provide a meaningful name (here in this example we named it as Students.xml) (Stackoverflow, 2012) (MSDN, 2015).



```xml
<?xml version="1.0" encoding="utf-8" ?>
<Students>
  <Student>
    <Name>Adam</Name>
    <Email>Adam@aaa.com</Email>
  </Student>
  <Student>
    <Name>Bob</Name>
    <Email>bob@bbb.com</Email>
  </Student>
  <Student>
    <Name>Cerig</Name>
    <Email>Cerig@ccc.com</Email>
  </Student>
  <Student>
    <Name>Demon</Name>
    <Email>Demon@ddd.com</Email>
  </Student>
  <Student>
    <Name>Elise</Name>
    <Email>Elise@eee.com</Email>
  </Student>
  <Student>
    <Name>Faila</Name>
    <Email>Faila@fff.com</Email>
  </Student>
  <Student>
    <Name>Garry</Name>
    <Email>Garry@ggg.com</Email>
  </Student>
  <Student>
    <Name>Harry</Name>
    <Email>Harry@hhh.com</Email>
  </Student>
  <Student>
    <Name>Idin</Name>
    <Email>Idin@iii.com</Email>
  </Student>
  <Student>
    <Name>Jay</Name>
    <Email>Jay@jjj.com</Email>
  </Student>
</Students>
```

Figure 49. Student Xml File

Now open "Student.xml" test file and let's put some data on our file. In Figure 48 root element is going to be 'Students'. 'Students' element will have several 'Student' elements and each 'Student' element will have 'Name' and 'Email'. Given below (in Figure 49) is Students.xml file code. Now we have to tell our unit test to use the data which is present in Students.xml file. For this we have to manipulate "DataSource" attribute. At the moment this DataSource is pointing to "Student" database table and we have to modify it to point to xml file. To do so we suppose to change its parameters as shown in Figure 50.



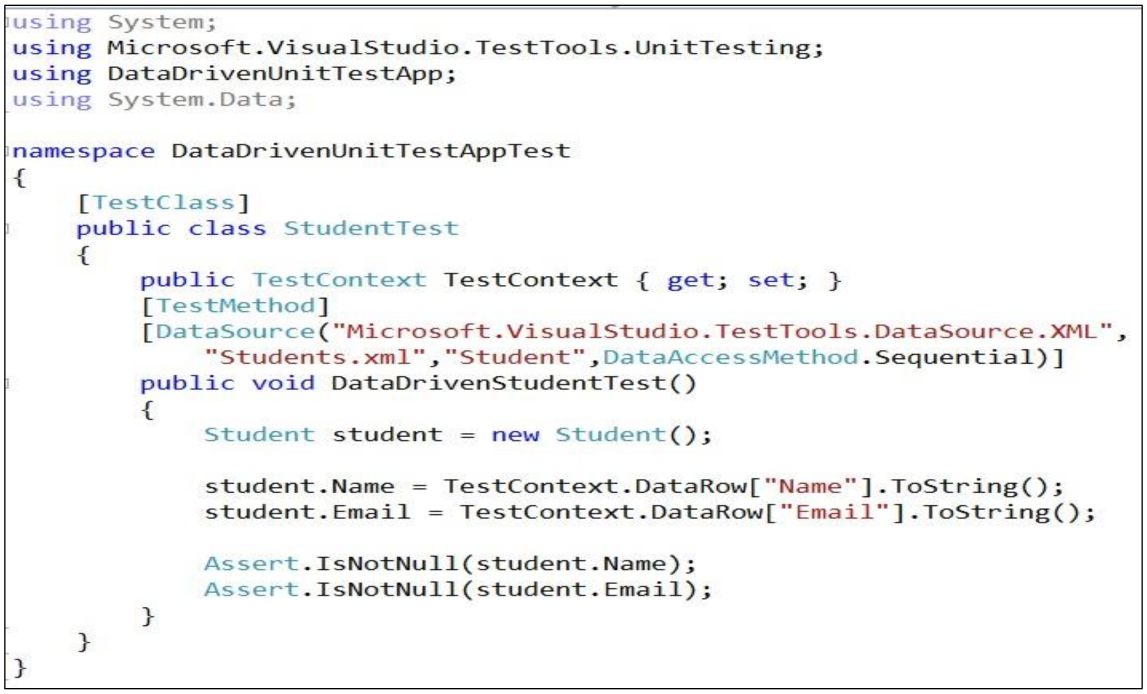
```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML","Students.xml","Student",DataAccessMethod.Sequential)]
```

Figure 50. DataSource for Xml

As shown in Figure 50, Data Source for xml has four parameters.

1. Microsoft.Visualstudio.TestTools.DataSource.XML – its provider name for xml.
2. Employee.xml – its file name.
3. Employee – its table name, which in this case is Employee element.
4. DataAccessMethodSequential – its data access method. It can be sequential or random. Here we selected sequential so that this unit test executes for every row sequentially as the data appear in the data source (MSDN, 2015).

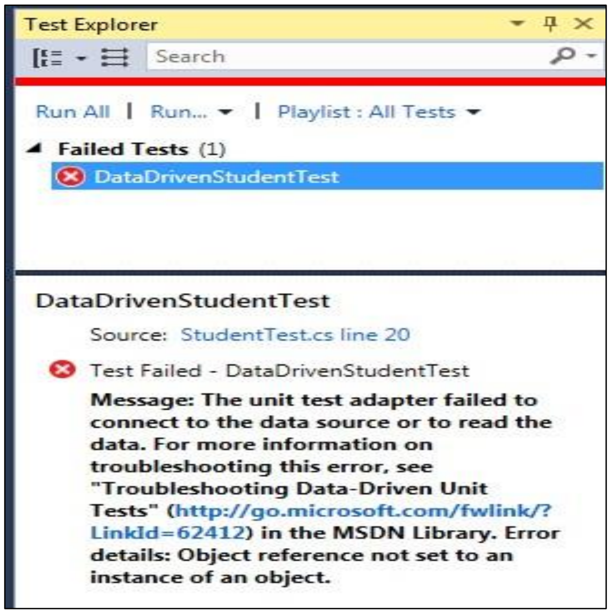
Now our Data Source is pointing to desired xml file. So our overall Unit test will look like same as in Figure 51. This method now apart from having "TestClass" and "TestMethod" attributes, it also has "DataSource" attribute. This attribute contains all 4 element described above.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using DataDrivenUnitTestApp;
using System.Data;

namespace DataDrivenUnitTestAppTest
{
    [TestClass]
    public class StudentTest
    {
        public TestContext TestContext { get; set; }
        [TestMethod]
        [DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML",
            "Students.xml","Student",DataAccessMethod.Sequential)]
        public void DataDrivenStudentTest()
        {
            Student student = new Student();

            student.Name = TestContext.DataRow["Name"].ToString();
            student.Email = TestContext.DataRow["Email"].ToString();

            Assert.IsNotNull(student.Name);
            Assert.IsNotNull(student.Email);
        }
    }
}
```

Figure 51. Complete Code of StudentTest for XML

Now when we build this project and run unit test it will get failed and throw Null Exception stating "object reference not set to an instance of an object" which means that it not able to find Employees.xml file (shown in Figure 52).



DataDrivenStudentTest

Source: StudentTest.cs line 20

Test Failed - DataDrivenStudentTest

Message: The unit test adapter failed to connect to the data source or to read the data. For more information on troubleshooting this error, see "Troubleshooting Data-Driven Unit Tests" (http://go.microsoft.com/fwlink/?LinkId=62412) in the MSDN Library. Error details: Object reference not set to an instance of an object.

Figure 52. Failed Unit Test

Reason of this error is that “Students.xml” file need to be copied in output folder. In order to fix this error right click on “DataDrivenUnitTestAppTest” unit test project and from the context menu select “Open Folder in File Explorer”. Steps are shown in Figure 53.

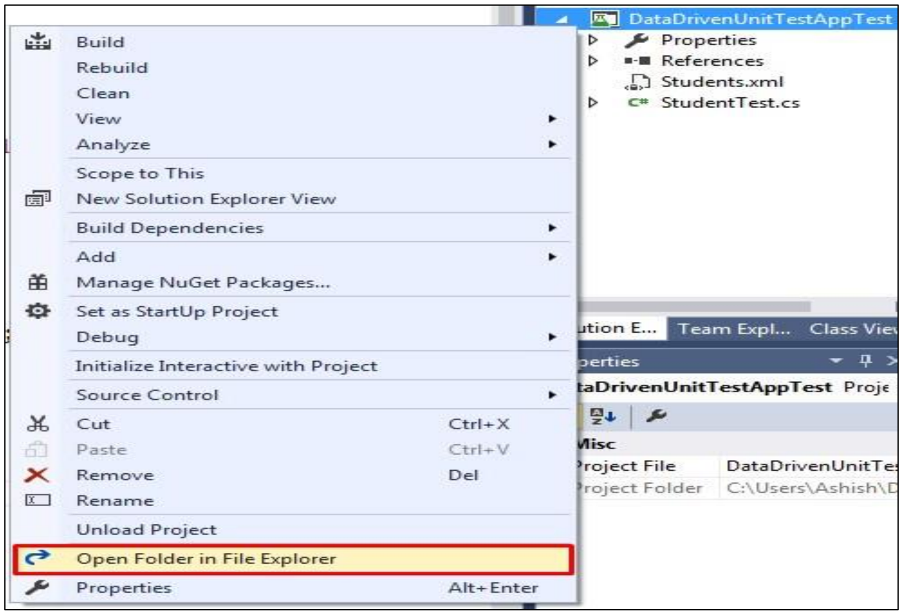


Figure 53. Open Folder in File Explorer

This will open project folder, navigate to “bin” then “obj” folder and this is the place where we need to have “Students.xml” file.

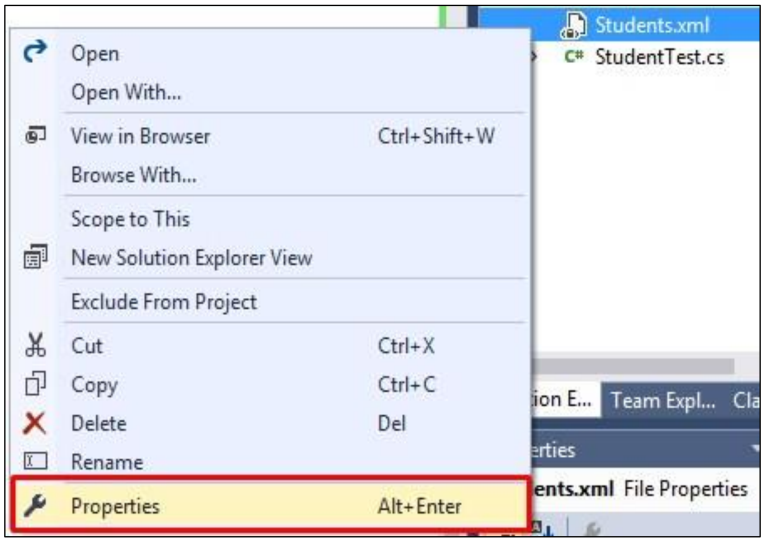
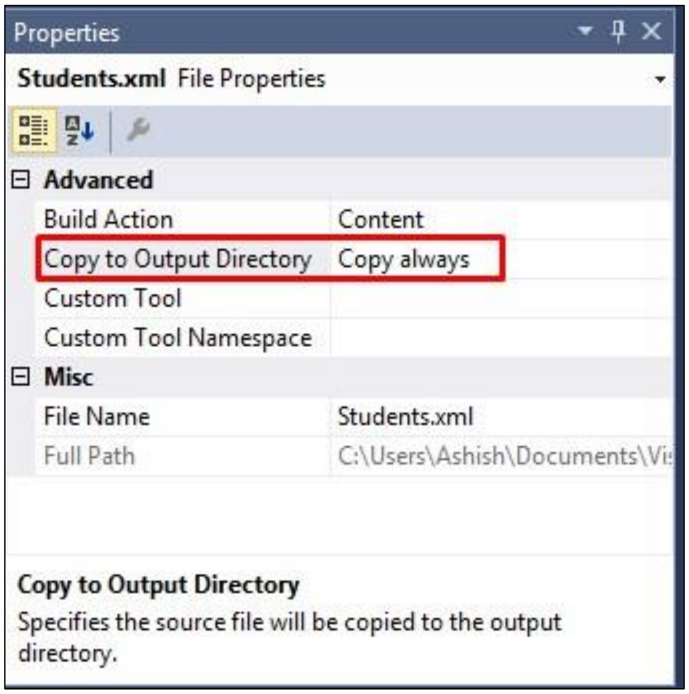


Figure 54. Selecting Properties of Students File

The easiest way to do it from Visual Studio's solution explorer. Right click on Employee.xml and from the context menu select "Properties" (shown in Figure 54).


Properties

Students.xml File Properties

Advanced	
Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
Misc	
File Name	Students.xml
Full Path	C:\Users\Ashish\Documents\Vi

Copy to Output Directory
Specifies the source file will be copied to the output directory.

Figure 55. Properties of Student File

It will open "Properties" (same as Figure 55) window and if you carefully look it has "copy to output directory" option which right now is set as "Do not copy", Change it from "Do not copy" to "copy always". Now as soon as you build this project it will copy it to output folder.

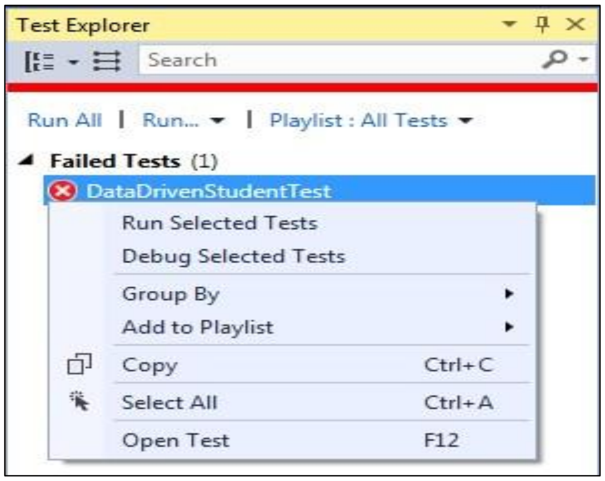
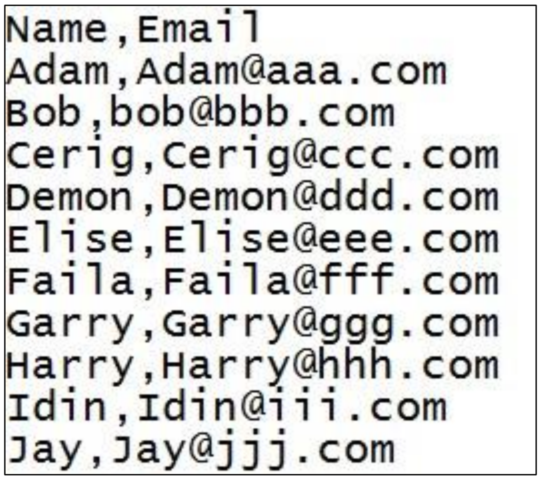


Figure 56. Run Selected Tests

If you want to check it go to above given location and it should have Employee.xml file. Now this unit test has all material for successful run. To run this unit test, fallow the steps provided in Figure 56. So when we run this unit test and will compare all the data from the xml file with given regular expression.

Data Driven Unit Test – CSV

In our previous section we discuss how to drive unit test with the data which is present in xml file. In this section we will discuss how to drive same unit test with the data which is present in csv (Comma Separated Values) file (KudVenkat, 2016). The first step here is to create a csv file. CSV file is nothing but comma separated file and to create it open notepad and write data (name and email) and separate it with comma (Prasad Honrao, n.d.). Save the file with .csv extension, like Students.csv. Content of Students.csv file is provided in Figure 57.

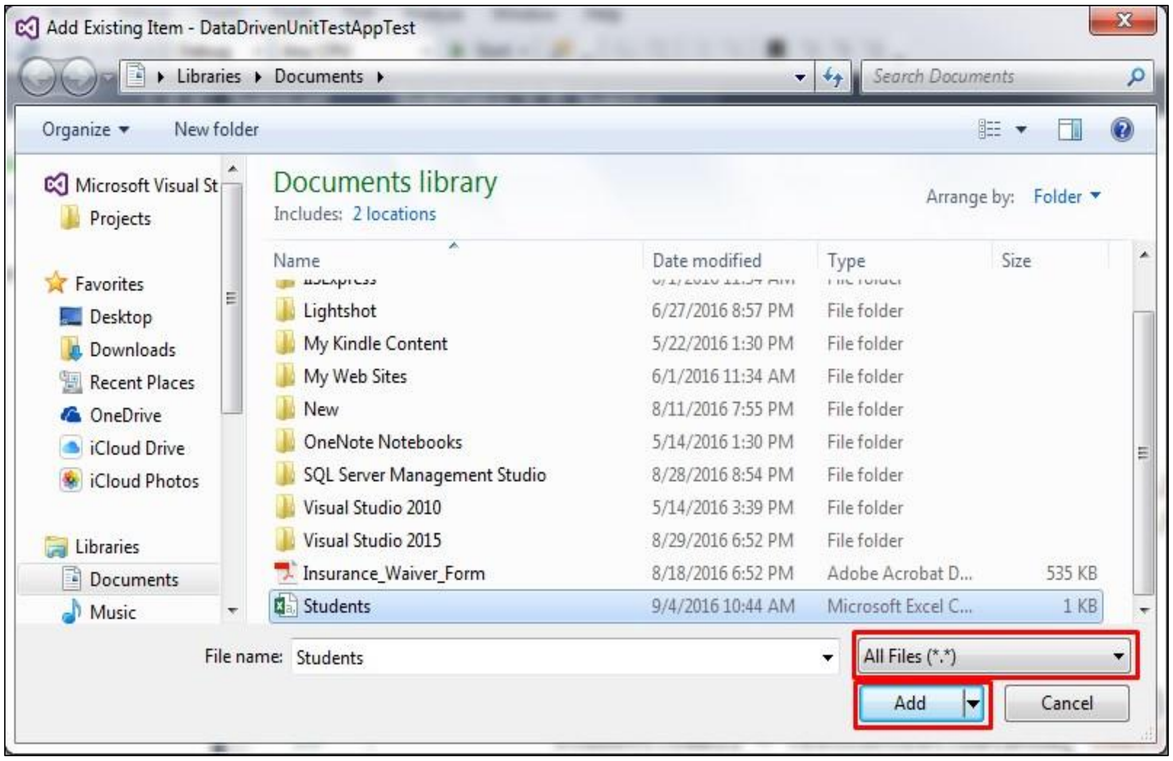


```
Name,Email
Adam,Adam@aaa.com
Bob,bob@bbb.com
Cerig,Cerig@ccc.com
Demon,Demon@ddd.com
Elise,Elise@eee.com
Faila,Faila@fff.com
Garry,Garry@ggg.com
Harry,Harry@hhh.com
Idin,Idin@iii.com
Jay,Jay@jjj.com
```

Figure 57. Csv File of Student

Note: - Name and email at the top of students.csv file is nothing but column header.

Now let’s add this “Students.csv” file to our “DataDrivenUnitTestAppTest” project by right clicking on it and selecting “Add” and “Existing Item”. Navigate to Students.csv file select it and click on “Add”, as shown in Figure 58. (If Students.csv file does not show up on its location then make sure that select “All Files (.*.)”).



Name	Date modified	Type	Size
Lightshot	6/27/2016 8:57 PM	File folder	
My Kindle Content	5/22/2016 1:30 PM	File folder	
My Web Sites	6/1/2016 11:34 AM	File folder	
New	8/11/2016 7:55 PM	File folder	
OneNote Notebooks	5/14/2016 1:30 PM	File folder	
SQL Server Management Studio	8/28/2016 8:54 PM	File folder	
Visual Studio 2010	5/14/2016 3:39 PM	File folder	
Visual Studio 2015	8/29/2016 6:52 PM	File folder	
Insurance_Waiver_Form	8/18/2016 6:52 PM	Adobe Acrobat D...	535 KB
Students	9/4/2016 10:44 AM	Microsoft Excel C...	1 KB

File name: Students
All Files (*.*)

Figure 58. Adding Student File of Csv Format

Now we want this Students.csv file copied to our output directory automatically just like xml file in the last section. For this right click on “Students.csv” file and from the context menu select “properties” (shown in Figure 59).

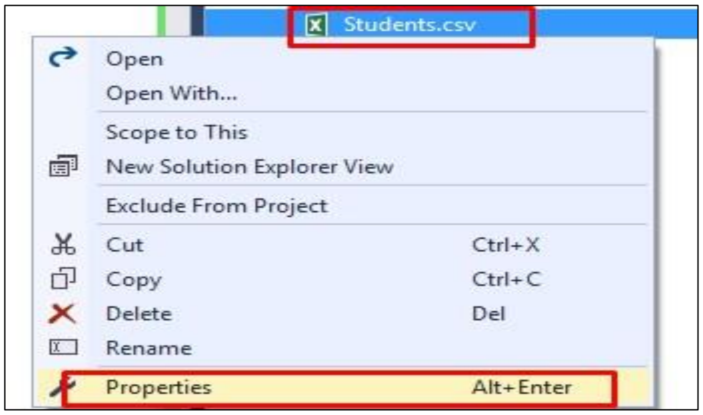
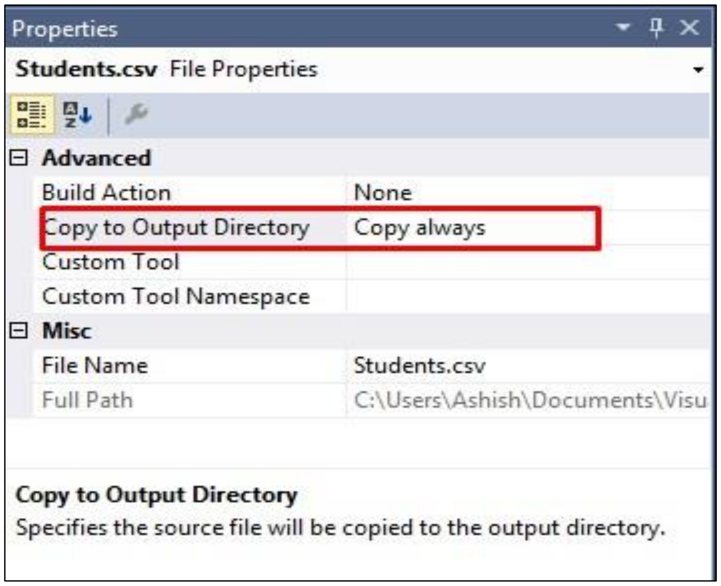


Figure 59. Selecting Properties of Student File

It will open Properties window which has “copy to output directory” option. Right now is set as “Do not copy”, set it from “Do not copy” to “copy always”. So now as soon as we build this solution our “Students.csv” file will get copied to our output folder. Steps are shown in Figure 60.


Properties

Students.csv File Properties

Advanced	
Build Action	None
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
Misc	
File Name	Students.csv
Full Path	C:\Users\Ashish\Documents\Visu

Copy to Output Directory
Specifies the source file will be copied to the output directory.

Figure 60. Properties of Student File

Now same as last section we have to tell our unit test to use Students.csv file as data source and we will accomplish it by configuring “DataSource” attribute. At the moment this “DataSource” attribute is pointing to xml file on which we worked on our last section. So let configure it as given in Figure 61.

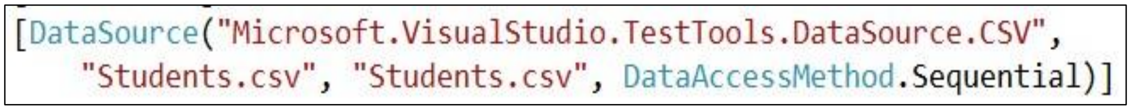

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
    "Students.csv", "Students.csv", DataAccessMethod.Sequential)]
```

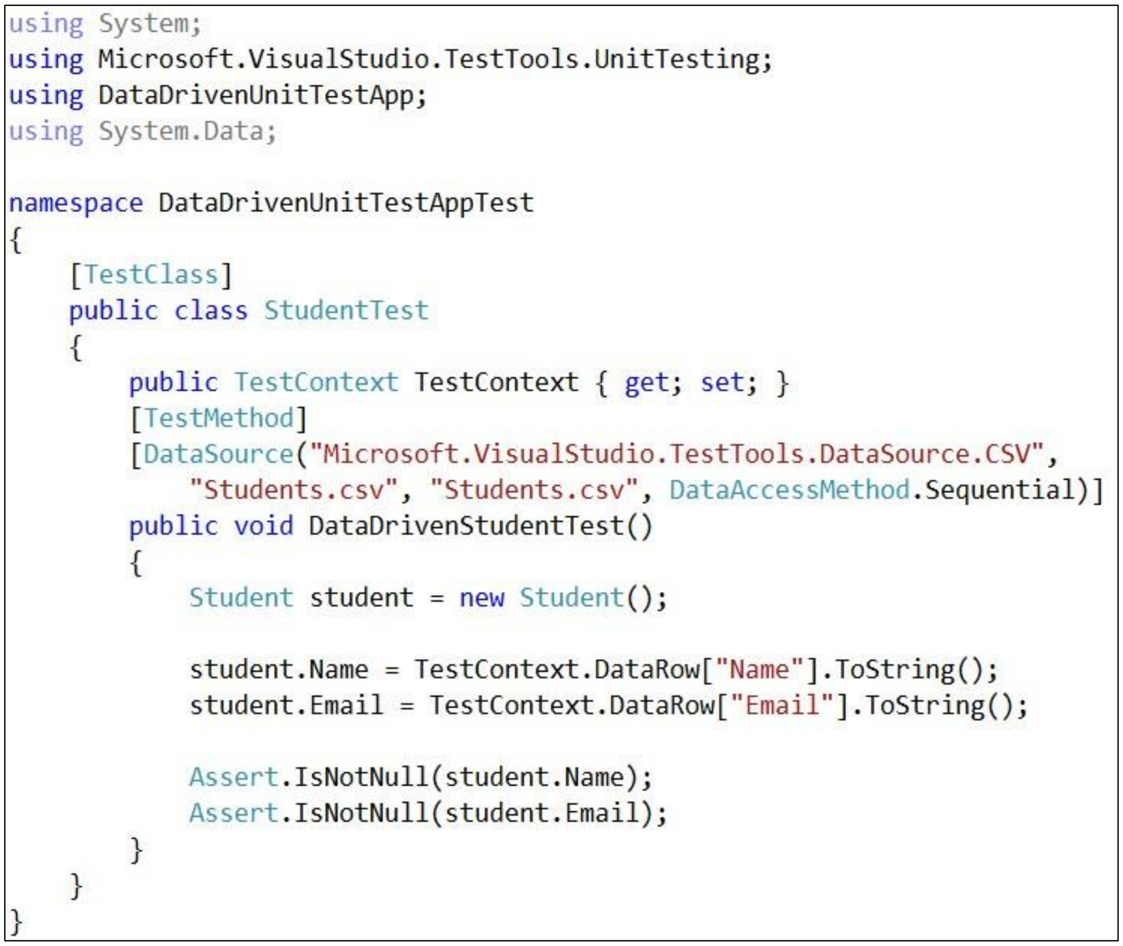
Figure 61. Data Source for Csv

Here Data Source has four parameters.

1. Microsoft.VisualStudio.TestTools.DataSource.CSV – its provider for csv.
2. Employees.csv – its name of csv file.
3. Employees.csv – its table name, which is going to same as Employees.csv file in this case.

4. DataAccessMethod.Sequential – its data access method. It can be sequential or random. Here we selected sequential so that this unit test execute for every row sequentialy as the data appear in the data source (Todd Meinershagen, 2011).

Now our Data Source is pointing to desired csv file. So our overall Unit test will look same as in Figure 62.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using DataDrivenUnitTestApp;
using System.Data;

namespace DataDrivenUnitTestAppTest
{
    [TestClass]
    public class StudentTest
    {
        public TestContext TestContext { get; set; }
        [TestMethod]
        [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
            "Students.csv", "Students.csv", DataAccessMethod.Sequential)]
        public void DataDrivenStudentTest()
        {
            Student student = new Student();

            student.Name = TestContext.DataRow["Name"].ToString();
            student.Email = TestContext.DataRow["Email"].ToString();

            Assert.IsNotNull(student.Name);
            Assert.IsNotNull(student.Email);
        }
    }
}
```

Figure 62. Complete Code of Student Test for Csv

Now when we build this project and run unit test (Figure 63) it should pass.

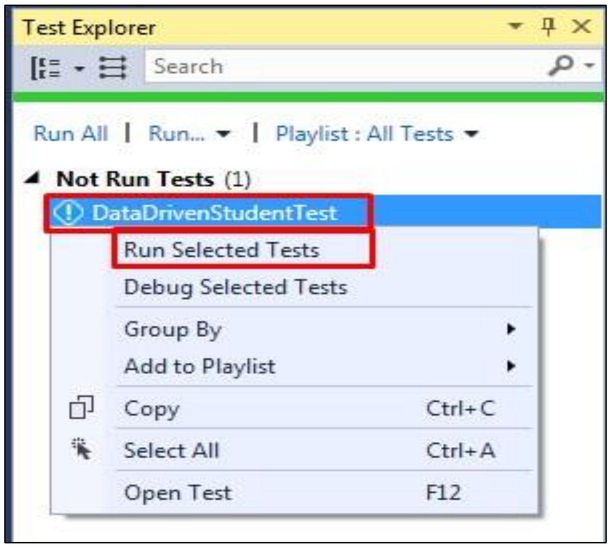


Figure 63. Running Unit Test

It ran successfully and sequentially check all email address of Employee.csv file with our regular expression.

LEARNING OBJECT 3 – ASSERT CLASSES OF UNIT TEST

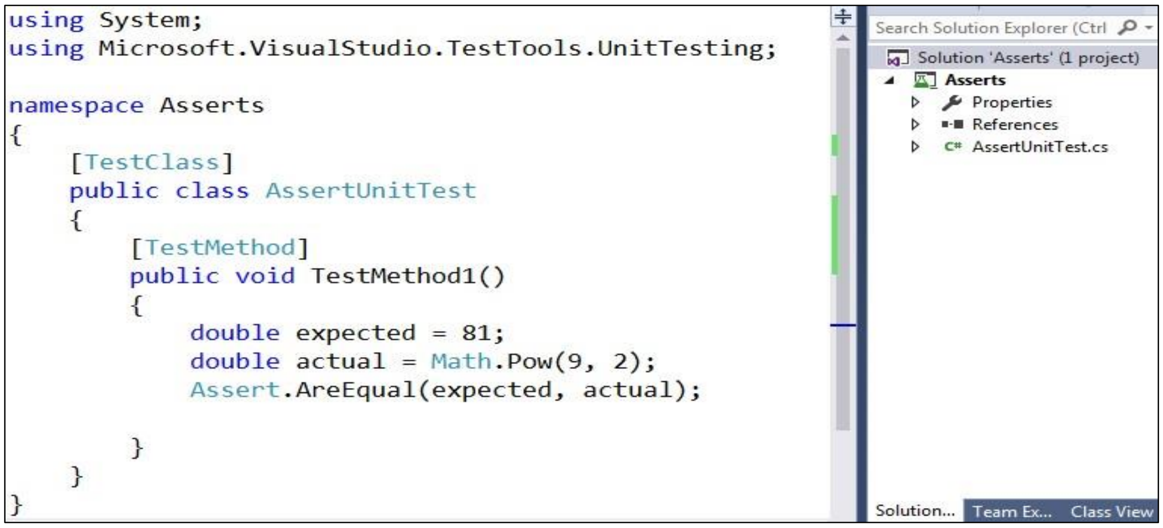
We include Assert statement within our unit tests to verify the correctness of code that we are testing. To perform these assertion, MS Test framework provide three different classes – Assert Class, CollectionAssert Class and StringAssert Class. Let's discuss these three different assert classes and different methods that they provide in the fallowing sections.

Assert Class

Assert class provide several static methods that can be used to verify assertions. Like, Assert.AreEqual, Assert.AreNotEqual, Assert.AreSame, Assert.AreNotSame, Assert.Fail, Assert.Inconclusive, Assert.True, Assert.False, Assert.IsNull, Assert.IsNotNull, Assert.IsInstanceOfType and Assert.IsNotInstanceOfType (MSDN, n.d.). Let's look these test methods one by one with example.

Assert.AreEqual

To Assert.AreEqual we pass expected and actual value. If they match assertion succeed. When the assertion succeed unit test passes. If they do not match the assertion fails.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Asserts
{
    [TestClass]
    public class AssertUnitTest
    {
        [TestMethod]
        public void TestMethod1()
        {
            double expected = 81;
            double actual = Math.Pow(9, 2);
            Assert.AreEqual(expected, actual);

        }
    }
}
```

Figure 64. Assert.AreEqual Class

When the assertion is fails “assert.fail” exception is thrown. When this exception is thrown the unit test fails. Let’s understand this with “AssertUnitTest” program under “Asserts” Project (Figure 64). This is very simple program where we are testing Math.Pow() method. This method helps us to compute power of given value.

Right here (Figure 64) we are computing 9 raised to the power 2 which is 81 so we assign this value to “expected” variable and “actual” variable will have the output of Math.Pow(9,2) method. Finally, we are comparing expected value with actual value. In this case they are going to match and thus this unit test will pass. Steps are giving me in Figure 65.

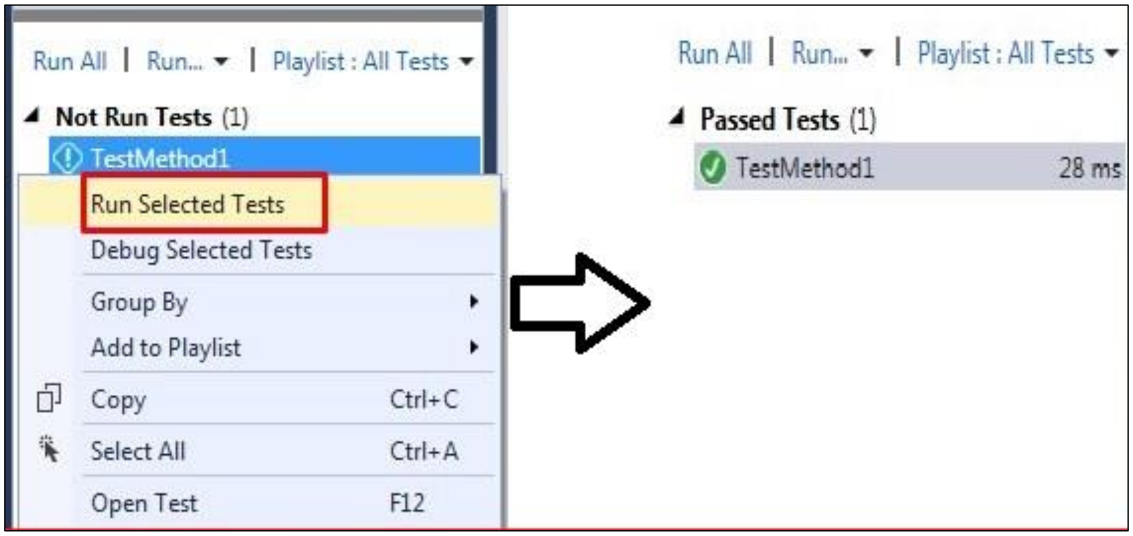
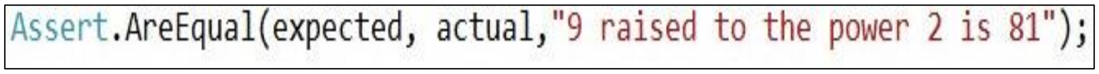


Figure 65. Result of Assert.AreEqual

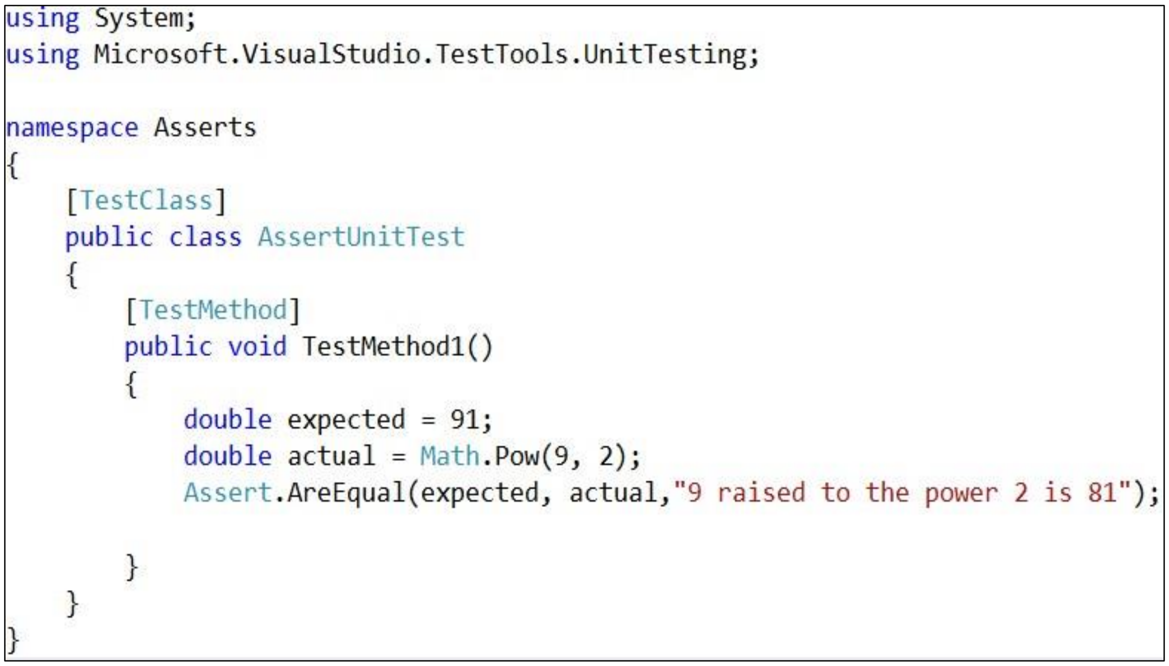
When the assertion fail we can also include a message that we want to display. To do this we use third and fourth parameter of Assert.AreNotEqual. Suppose whenever this unit test fails we want to display the message “9 raised to the power 2 is 81” then we can do as given in Figure 66.



```
Assert.AreEqual(expected, actual,"9 raised to the power 2 is 81");
```

Figure 66. Assert.AreEqual with Third parameter

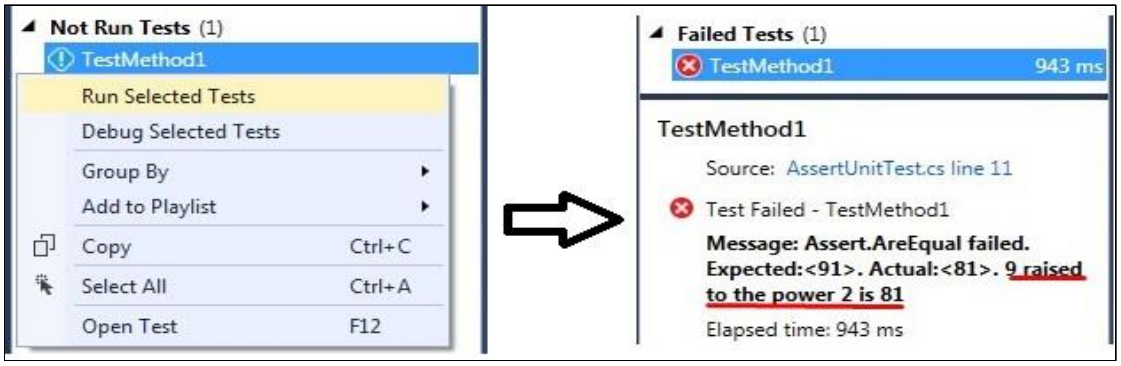
Now if we change the "expected" value to any number other than 81 (say 91), then expected and actual values will not be equal and this unit test will fail. Unit Test is given in Figure 67.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Asserts
{
    [TestClass]
    public class AssertUnitTest
    {
        [TestMethod]
        public void TestMethod1()
        {
            double expected = 91;
            double actual = Math.Pow(9, 2);
            Assert.AreEqual(expected, actual,"9 raised to the power 2 is 81");

        }
    }
}
```

Figure 67. Assert.AreEqual (To Be Fail)

Now if we run this unit test it will get fail and will show message that "9 raised to the power 2 is 81" as Figure 68.



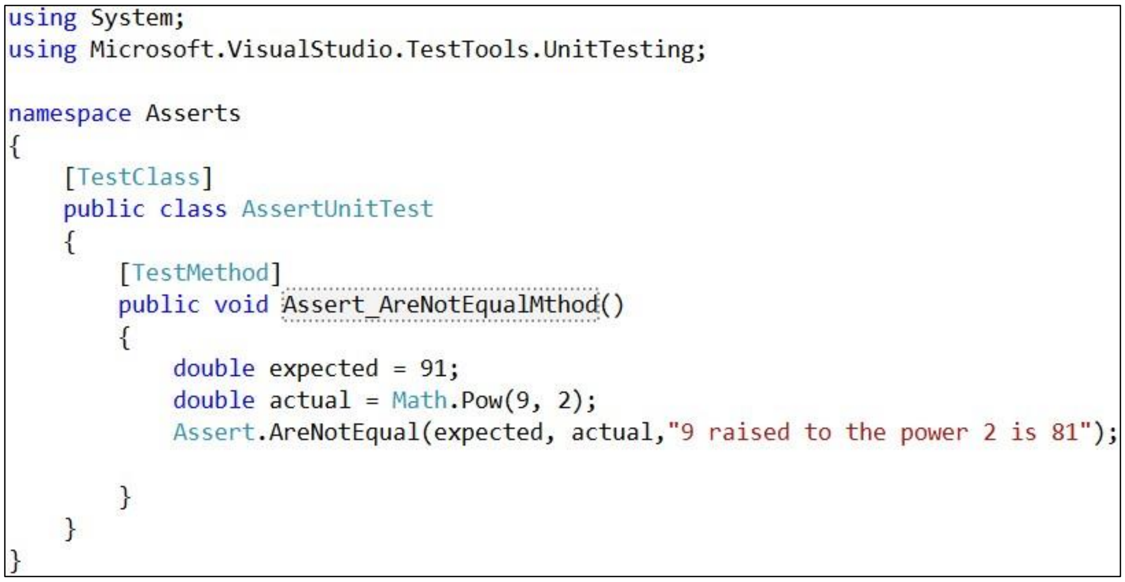
TestMethod1
Source: AssertUnitTest.cs line 11
Test Failed - TestMethod1
Message: Assert.AreEqual failed. Expected:<91>. Actual:<81>. 9 raised to the power 2 is 81
Elapsed time: 943 ms

Figure 68. Failed Unit Test with Message

This parameter to provide message as string is available in assert methods.

Assert.AreNotEqual

To Assert.AreNotEqual is opposite of Assert.AreEqual method. we pass expected and actual value. If they do not match, assertion succeed. When the assertion succeed unit test passes. If they match the assertion fails. When the assertion is fails “assert.fail” exception is thrown. When this exception is thrown the unit test fails. In the above example (Figure 66) if implement Assert.AreNotEqual then it will get pass as expected and actual values are not same, as given in Figure 69.



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Asserts
{
    [TestClass]
    public class AssertUnitTest
    {
        [TestMethod]
        public void Assert_AreNotEqualMthod()
        {
            double expected = 91;
            double actual = Math.Pow(9, 2);
            Assert.AreNotEqual(expected, actual,"9 raised to the power 2 is 81");

        }
    }
}
```

Figure 69. Assert.AreNotEqual

As this unit test passes it will not show “9 raised to the power 2 is 81” message, as shown in Figure 70.

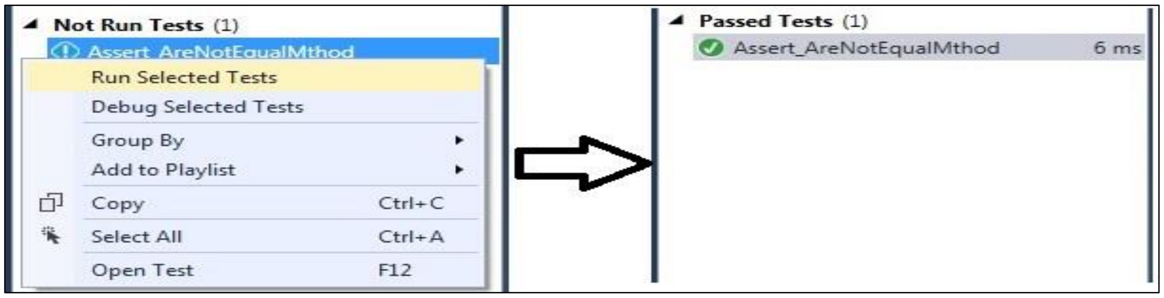
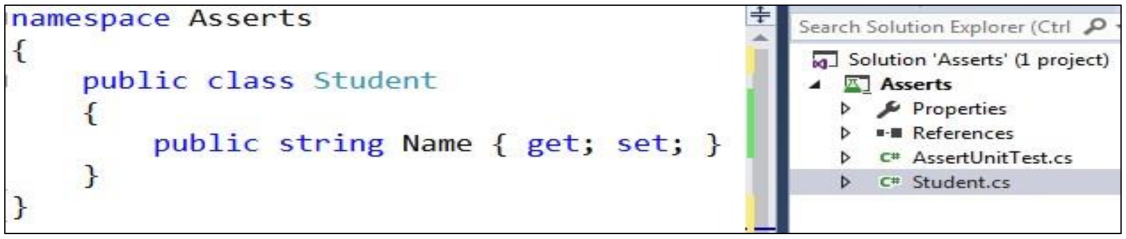


Figure 70. Result of Assert.AreNotEqual

Assert.AreSame

This method is going to assert if two object reference variable point to same object. To better understand it lets create a “Student” class with single property “Name”, as shown in Figure 71.

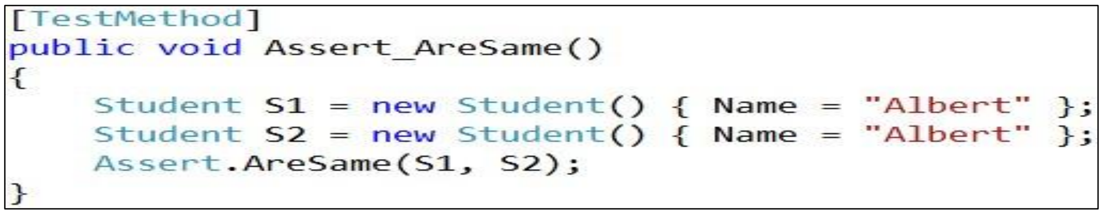


```
namespace Asserts
{
    public class Student
    {
        public string Name { get; set; }
    }
}
```

Search Solution Explorer (Ctrl
Solution 'Asserts' (1 project)
Asserts
Properties
References
AssertUnitTest.cs
Student.cs

Figure 71. Student Class with Name Property

Now let’s create two “Student” objects S1 and S2 and provide same “Name” property value “Albert” to both S1 and S2 object, (Figure 72).



```
[TestMethod]
public void Assert_AreSame()
{
    Student S1 = new Student() { Name = "Albert" };
    Student S2 = new Student() { Name = "Albert" };
    Assert.AreSame(S1, S2);
}
```

Figure 72. Assert.AreSame Method

Now though both property values are same these reference variable S1 and S2 actually pointing to two different object in memory thus references are not same. Hence assertion is going to fail here (shown in Figure 73).

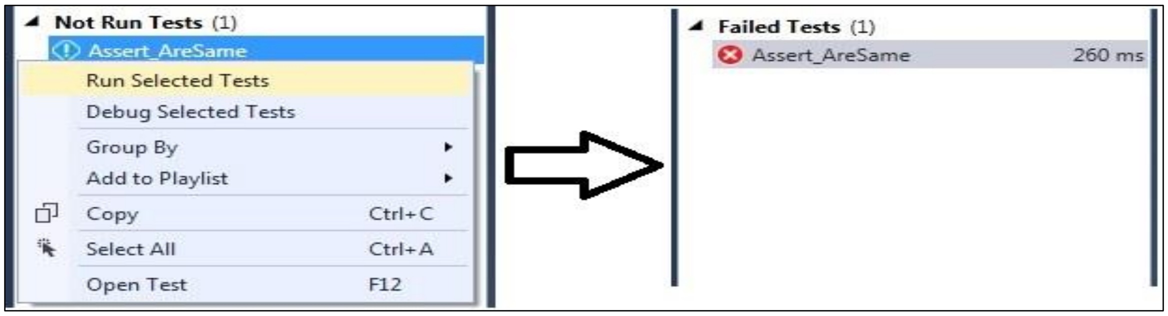
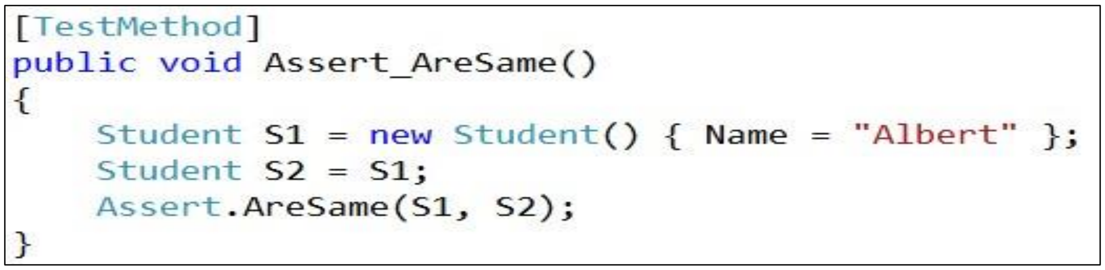


Figure 73. Assert.AreSame (Failed As Ref. Variable Are Different)

Now if S1 and S2 point to same direction then assertion should succeed. If we have something do as given in Figure 74.



```
[TestMethod]
public void Assert_AreSame()
{
    Student S1 = new Student() { Name = "Albert" };
    Student S2 = S1;
    Assert.AreSame(S1, S2);
}
```

Figure 74. Reference Variables S1 and S2 Point To Same Object

Where S1 and S2 reference variable pointing to same object. Now when we run this unit test it should succeed as shown in Figure 75.

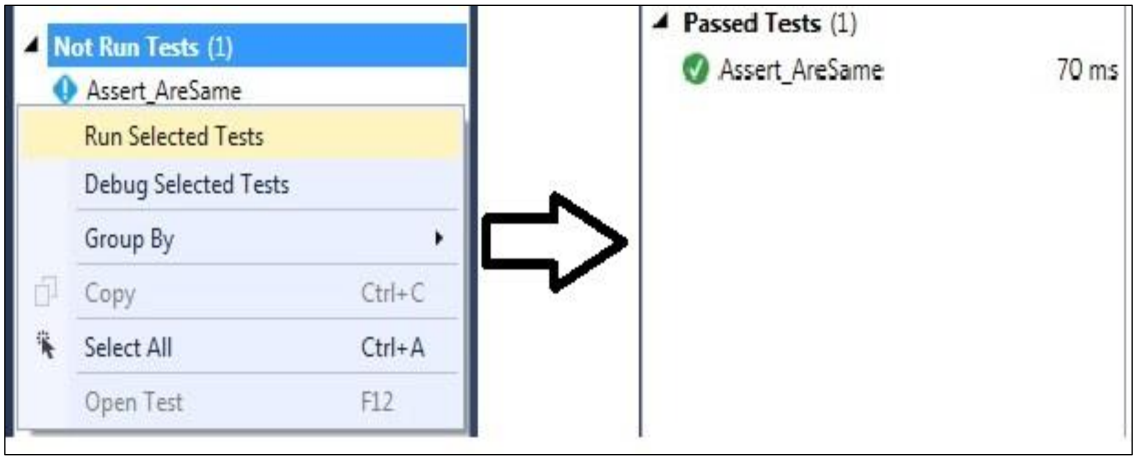
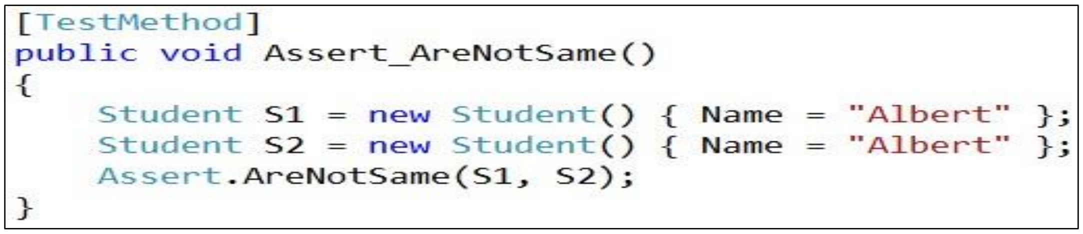


Figure 75. Assert.AreSame

Assert.AreNotSame

The opposite of Assert.AreSame method is Assert.AreNotSame method. This method makes sure that two object reference variable does not point to same object. If reference variable point to different object, then assertion succeed and thus unit test gets pass. In the above example (Figure 73) if we use Assert.AreNotSame method then assertion will get succeed as both reference variable S1 and S2 pointing to different objects as shown in Figure 76.



```
[TestMethod]
public void Assert_AreNotSame()
{
    Student S1 = new Student() { Name = "Albert" };
    Student S2 = new Student() { Name = "Albert" };
    Assert.AreNotSame(S1, S2);
}
```

Figure 76. Assert.AreNotSame Method

When we run this unit test it should get pass as shown in Figure 78.

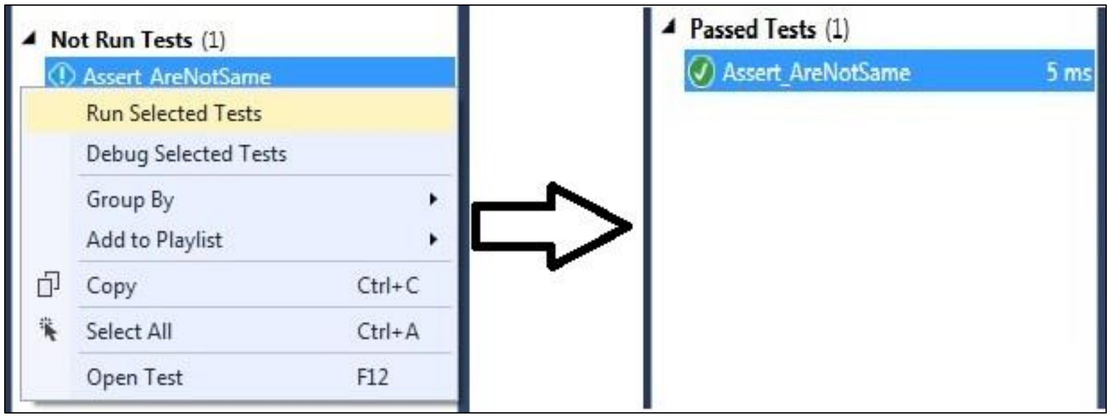
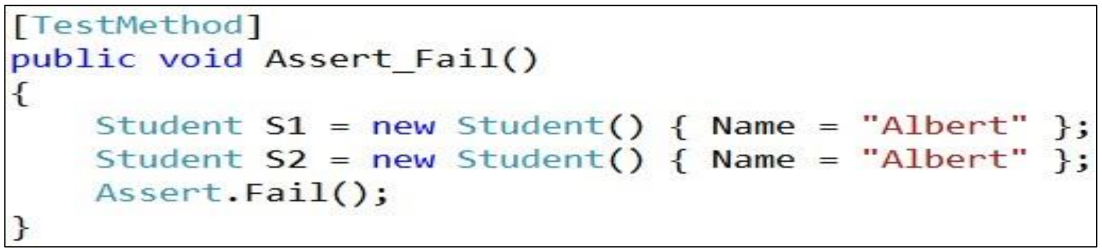


Figure 77. Assert.AreNotSame Pass

Assert.Fail

Assert.Fail method fail the assertion without checking any condition whatsoever. Implemented in Figure 78.



```
[TestMethod]
public void Assert_Fail()
{
    Student S1 = new Student() { Name = "Albert" };
    Student S2 = new Student() { Name = "Albert" };
    Assert.Fail();
}
```

Figure 78. Asert_Fail Method

Assert.Fail method does not take any parameter and get failed without asserting (Figure 79).

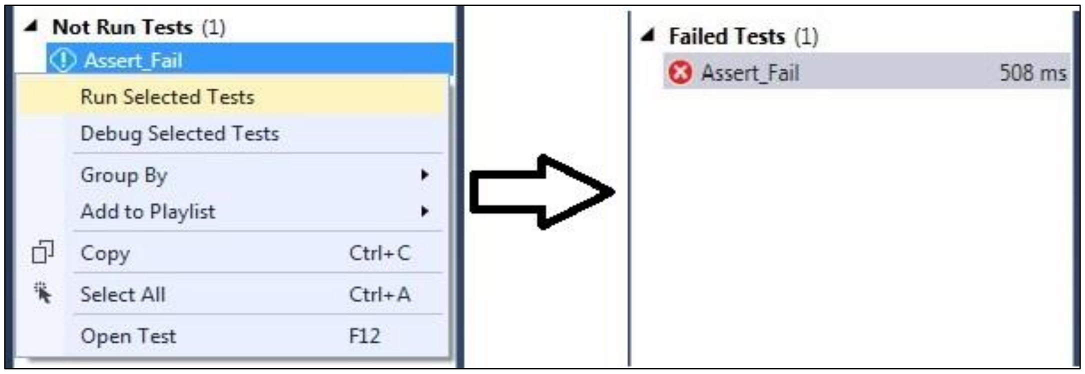
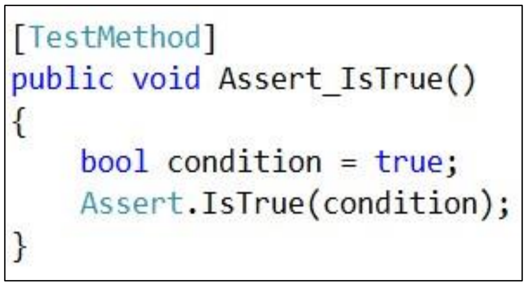


Figure 79. Assert.Fail Gets Fail without Assertion

At this point one question comes to our mind that why and where we ever use this method. One of the most common reason to use Assert.Fail is to get reminder for incomplete unit test. Let’s say we started a new unit test and we don’t have time to complete the implementation, as the result we may be don’t have any assertion within that unit test. When we don’t have any assertion and try to run our unit test then obviously that unit test is going to pass. In that case we may use Assert.Fail method so when we run our unit test that specific unit test in which have Assert.Fail method is going to fail and when this fails it’s a reminder for us that unit test is incomplete.

Assert.IsTrue

This method is going to verify whether specified condition is true. The assertion fails if the condition is false.



```
[TestMethod]
public void Assert_IsTrue()
{
    bool condition = true;
    Assert.IsTrue(condition);
}
```

Figure 80. Assert.IsTrue

Here in Figure 80, specified condition is true so this unit test will get pass (shown in Figure 81).

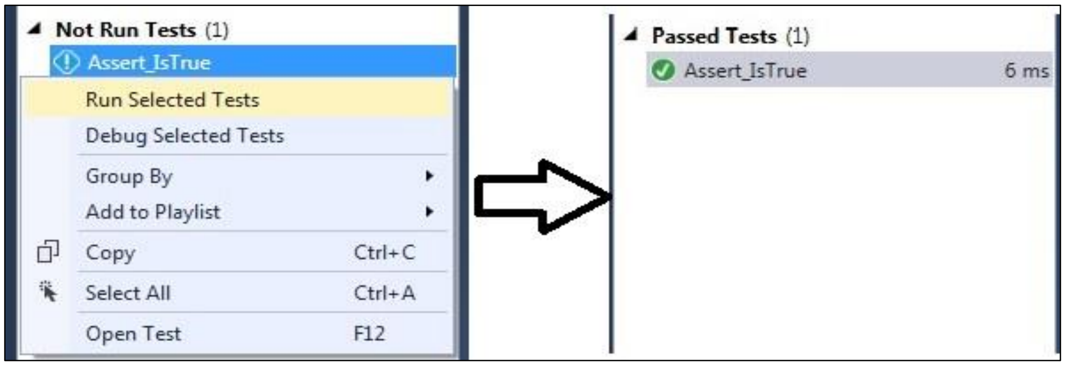
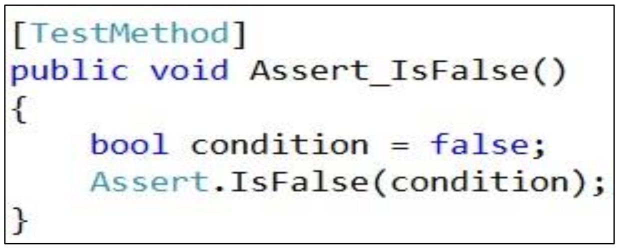


Figure 81. Assert.isTrue Get Pass

Assert.IsFalse

The opposite of Assert.IsTrue method is Assert.IsFalse. This method (Figure 82) is going to verify whether specified condition is false.



```
[TestMethod]
public void Assert_IsFalse()
{
    bool condition = false;
    Assert.IsFalse(condition);
}
```

Figure 82. Assert.IsFalse

Here specified condition is false so this unit test will get pass (shown in Figure 83).

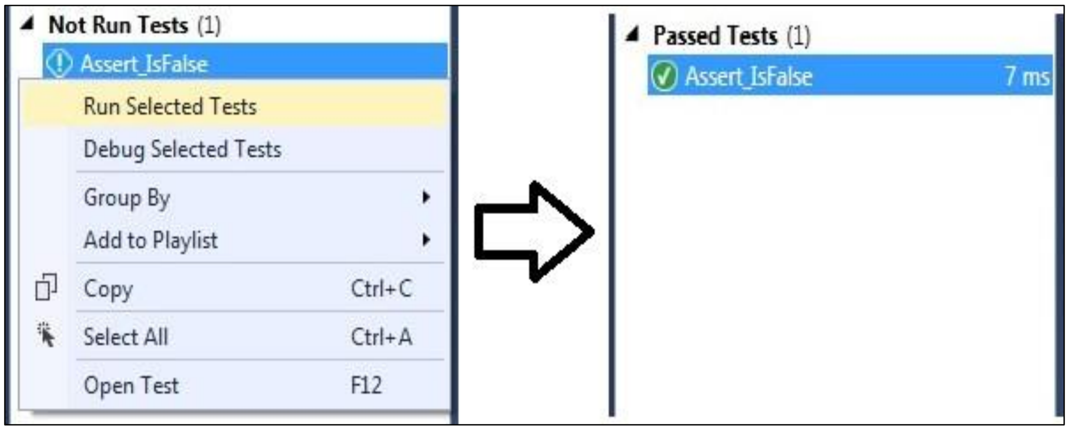
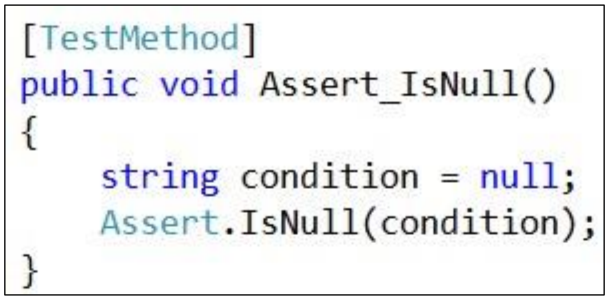


Figure 83. Assert.isFalse Get Pass

Assert.IsNull

This method is going to verify whether passed object is null. If it is not null, then assertion is going to fail and if it is null then assertion is going to pass.



```
[TestMethod]
public void Assert_IsNull()
{
    string condition = null;
    Assert.IsNull(condition);
}
```

Figure 84. Implementation of Assert.isNull Method

Here in Figure 84 “condition” string is null so this unit test will pass (as shown in Figure 85).

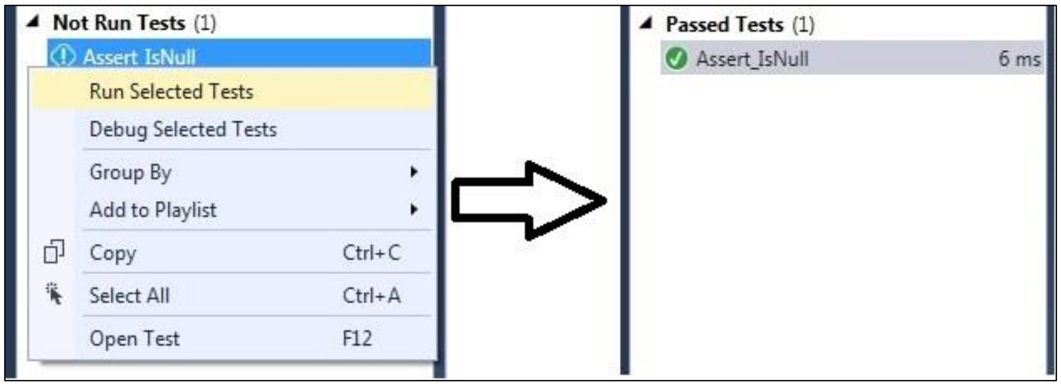
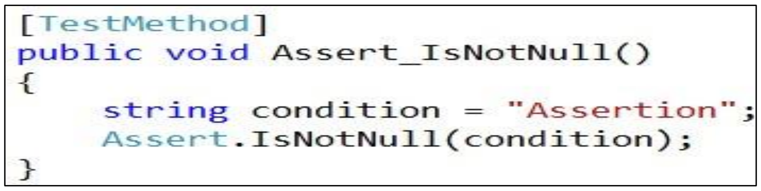


Figure 85. Assert.IsNullPass

Assert.IsNotNull

The opposite of Assert.IsNull method is Assert.IsNotNull method. This method is going to verify whether passed object is not null.



```
[TestMethod]
public void Assert_IsNotNull()
{
    string condition = "Assertion";
    Assert.IsNotNull(condition);
}
```

Figure 86. Implemenatation of Assert.IsNotNull Method

If it is null, then assertion is going to fail and if it is not null then assertion is going to pass as shown in Figure 86. Here "condition" string is not null so this unit test will pass (Figure 87).

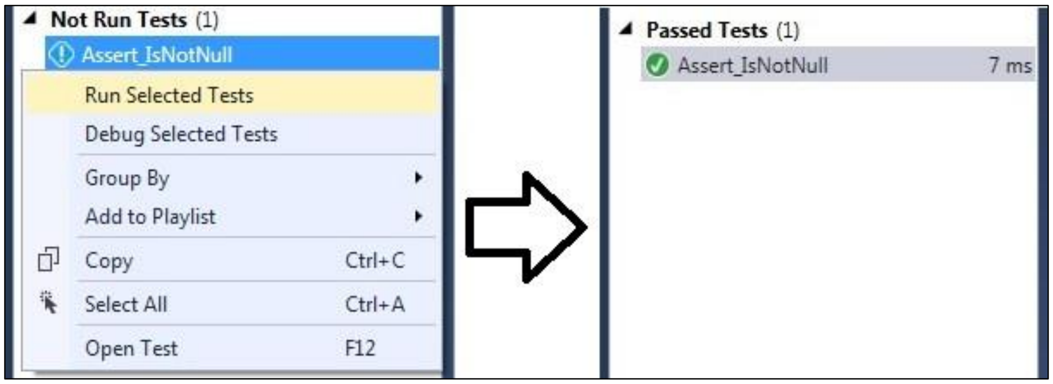
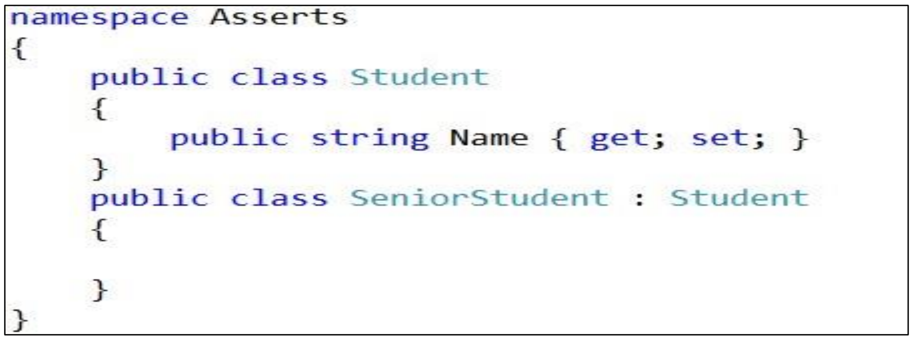


Figure 87. Assert.IsNotNull Pass

Assert.IsInstanceOfType

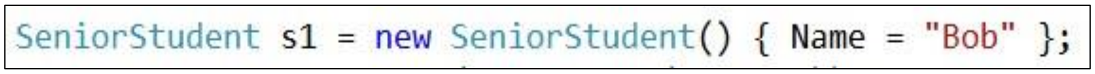
To understand this test method let's first look at "SeniorStudent" class given at figure88. Here "SeniorStudent" class is inheriting from "Student" class.



```
namespace Asserts
{
    public class Student
    {
        public string Name { get; set; }
    }
    public class SeniorStudent : Student
    {

    }
}
```

Figure 88. SeniorStudent Class

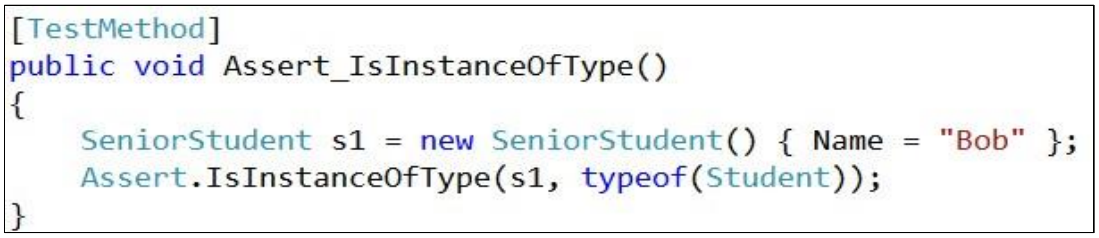
Now we will create instance of "SeniorStudent" class (s1) and as "SeniorStudent" class is child class of "Student" class, it can access "Name" Property of "Student" class. So let's initialize "Name" property as "Bob" (shown in Figure 89).



```
SeniorStudent s1 = new SeniorStudent() { Name = "Bob" };
```

Figure 89. Instance of SeniorStudent Class

Now we will implement assert.IsInstanceofType test method and assert instance of SeniorStudent class "s1" to typeof "Student" class.



```
[TestMethod]
public void Assert_IsInstanceOfType()
{
    SeniorStudent s1 = new SeniorStudent() { Name = "Bob" };
    Assert.IsInstanceOfType(s1, typeof(Student));
}
```

Figure 90. Implementation of Assert.IsInstanceOfType Test Method

Now as we know "SeniorStudent" class is driving from "Student" class so we can say "s1" is instance of type "Student" as they are related by inheritance (implemented in Figure 90). So when we run this unit test it should succeed as shown as in Figure 91.

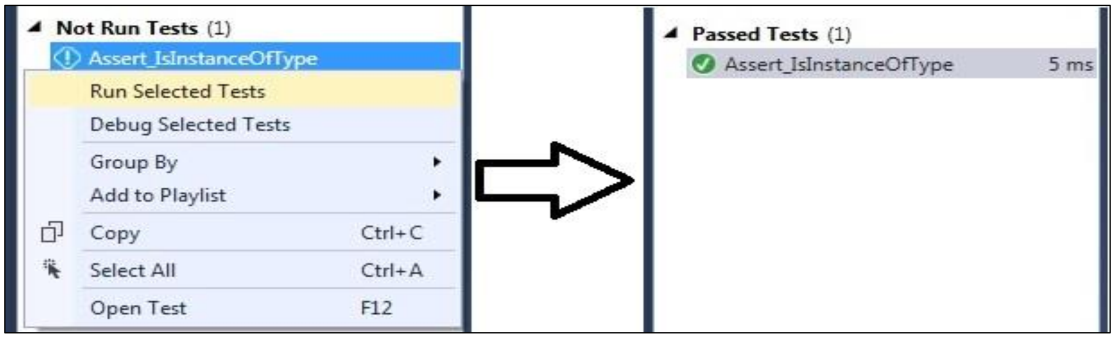
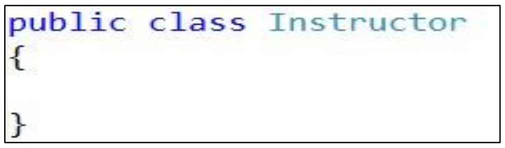


Figure 91. Assert.IsInstanceOfType Test Method Passes

Assert.IsNotInstanceOfType

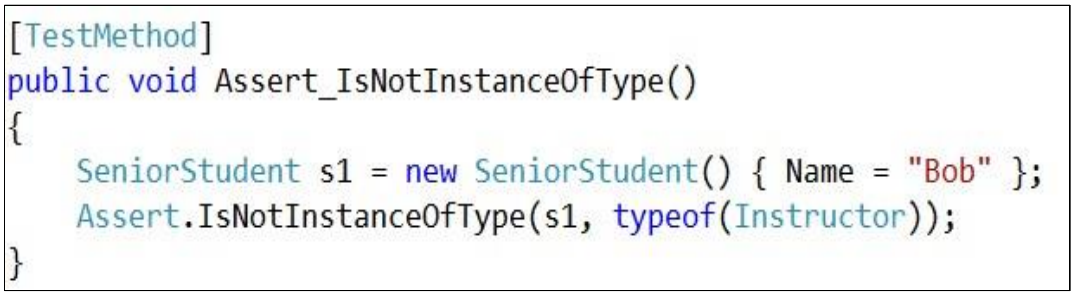
The opposite of Assert.IsInstanceOfType test method is Assert.IsNotInstanceOfType test method. This method verifies that the specified object is not an instance of the specified type. The assertion fails if the type is found in the inheritance hierarchy of the object. If we look at below given example (Figure 92) "Instructor" class is not inheriting from Student class.



```
public class Instructor
{

}
```

Figure 92. Instructor Class

Now when we create instance of “Student” class and try to assert it with “Instructor” class using Assert.IsNotInstanceOfType (Figure 93) test method it should get pass.



```
[TestMethod]
public void Assert_IsNotInstanceOfType()
{
    SeniorStudent s1 = new SeniorStudent() { Name = "Bob" };
    Assert.IsNotInstanceOfType(s1, typeof(Instructor));
}
```

Figure 93. Implementation of Assert.IsNotInstanceOfType Test Method

As Instructor class is not of type SeniorStudent’s instance (s1), so this unit test succeeds (shown inFigure 94).

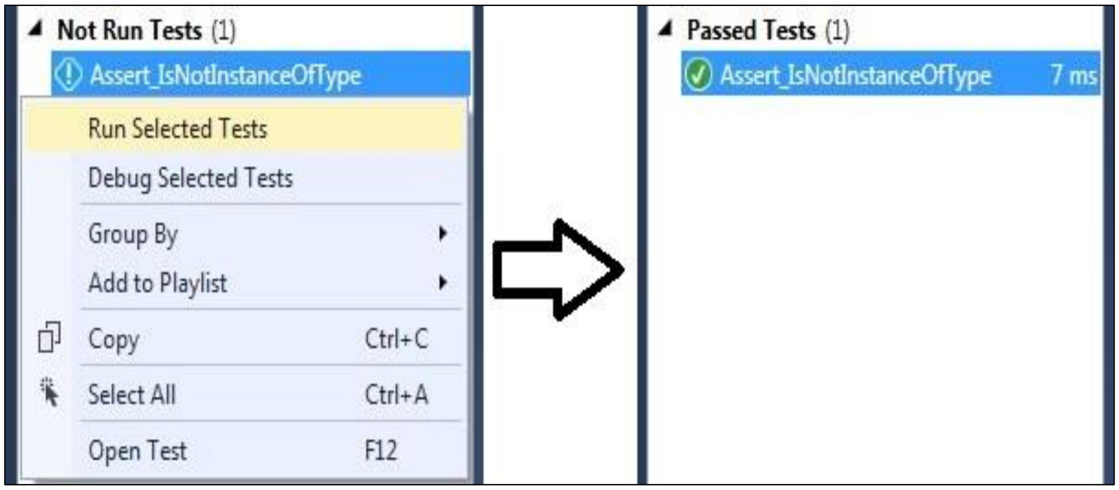


Figure 94. Assert.IsNotInstanceOfType Passes

Assert_IsNotInstanceOfType could be very helpful where we have to create too may classes and we get confuse that which one belongs to which one. This assert method would prove very helpful on those cases.

CollectionAssert Class

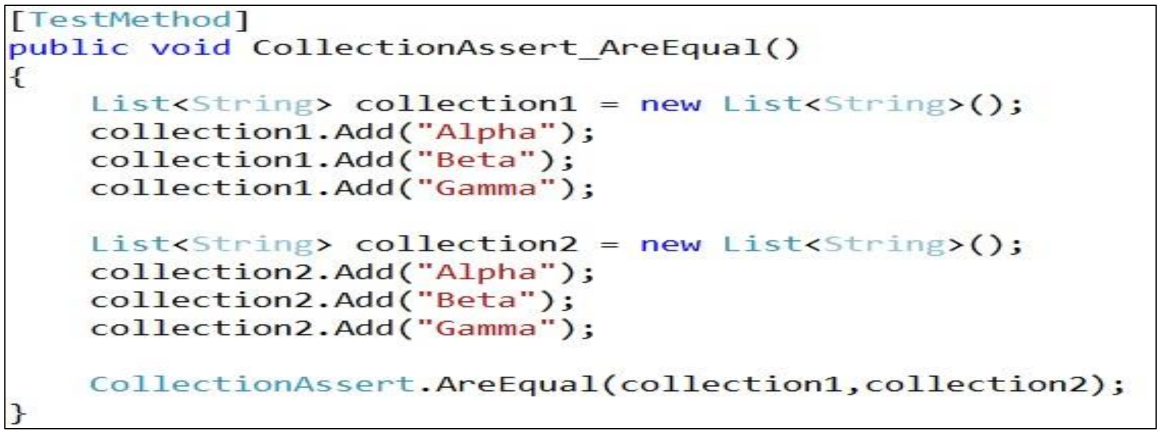
CollectionAssert class provides number of methods to compare collection (MSDN, 2013), like

1. CollectionAssert.AreEqual
2. CollectionAssert.AreNotEqual
3. CollectionAssert.Equivalent
4. CollectionAssert.AreNotEquivalent
5. CollectionAssert.Contains
6. CollectionAssert.DoesNotContains
7. CollectionAssert.IsSubsetOf
8. CollectionAssert.IsNotSubsetOf
9. CollectionAssert.AllItemsAreUnique
10. CollectionAssert.AllItemsAreNotNull
11. CollectionAssert.AllItemsAreInstanceOfType

Let's look these test methods one by one with example.

CollectionAssert.AreEqual

CollectionAssert.AreEqual test method verifies if two specified collections have same elements and elements within collections are present in same order. Let's look at example provided in Figure 95. Within "CollectionAssert_AreEqual" unit test unit test we have two collections – collection1 and collection2. Within collection1 we got three strings – "Alpha", "Beta" and "Gamma". Within collection2 we got same set of string element, "Alpha", "Beta" and "Gamma". And at the last statement we are passing these two collections – collection1 and collection2.



```
[TestMethod]
public void CollectionAssert_AreEqual()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Alpha");
    collection2.Add("Beta");
    collection2.Add("Gamma");

    CollectionAssert.AreEqual(collection1,collection2);
}
```

Figure 95. Implementation of Collection.AreEqual Test Method

So when we run this unit test we expect it to pass as both collections contain same elements in same order (as shown in Figure 96).

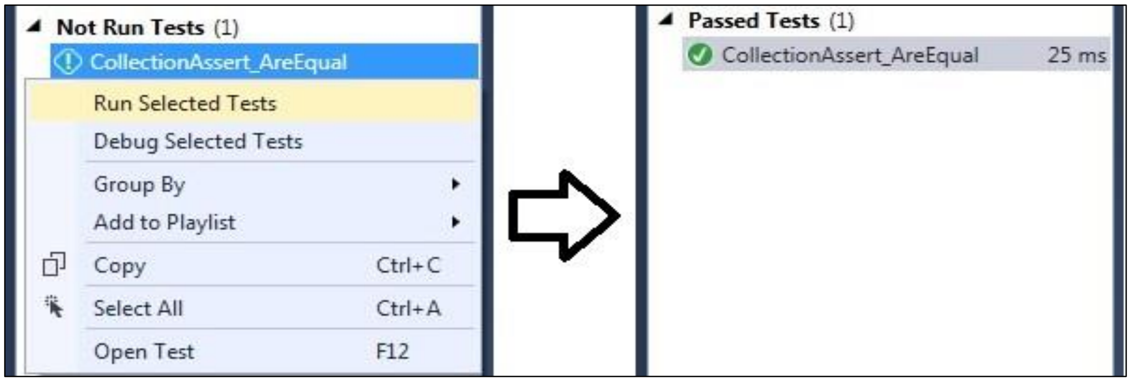
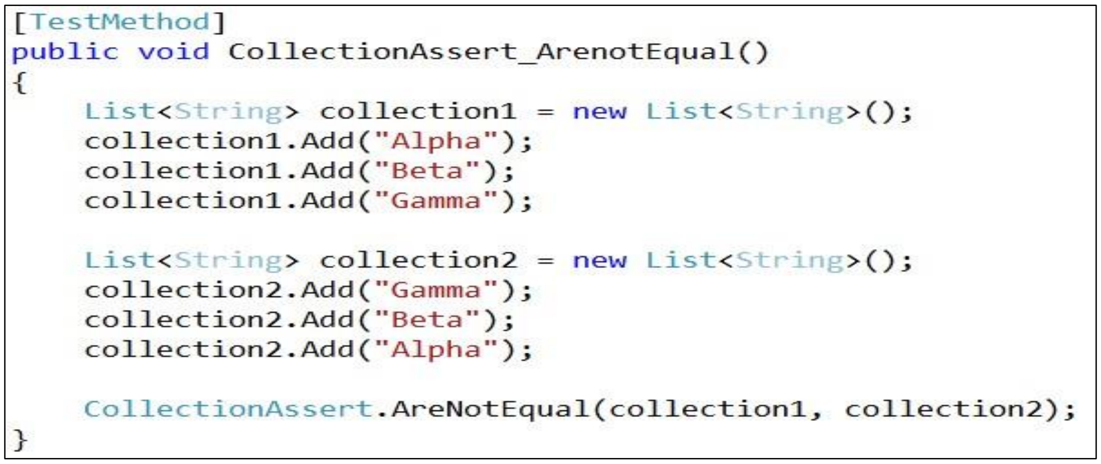


Figure 96. CollectionAssert.AreEqual Test Method Gets Pass

Unit test passes as expected.

CollectionAssert.AreNotEqual

The opposite of CollectionAssert.AreSame test method is CollectionAssert.AreNotSame. To illustrate it lets change the order of elements in collection2 such that “Gamma” comes first followed by “Beta” and “Alpha” as shown in Figure 97.



```
[TestMethod]
public void CollectionAssert_ArenotEqual()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Gamma");
    collection2.Add("Beta");
    collection2.Add("Alpha");

    CollectionAssert.AreNotEqual(collection1, collection2);
}
```

Figure 97. Implementation of CollectionAssert.AreNotSame Test Method

Here collection1 and collection2 have same elements but are in different order hence they are not equal and hence when we run this unit test we expect it to pass as shown in Figure 98.

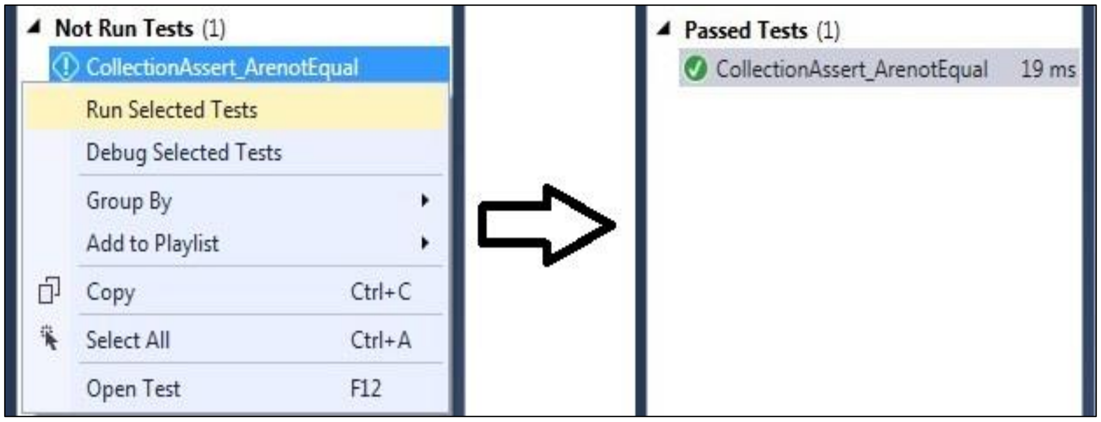
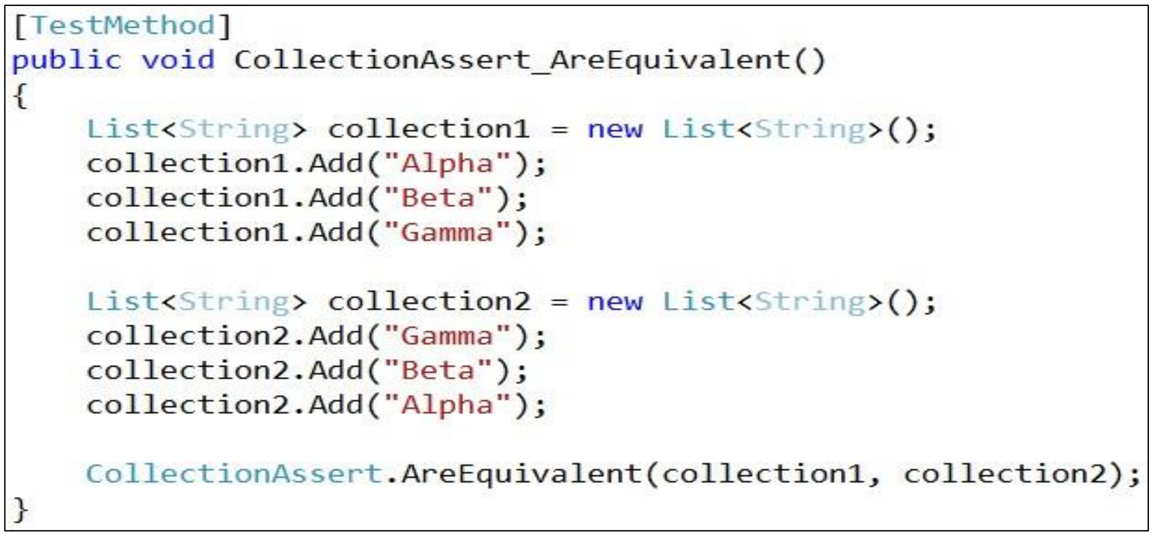


Figure 98. CollectionAssert.AreNotSame Gets Pass

CollectionAssert.AreEquivalent

CollectionAssert.AreEquivalent verifies if both collections got same elements. It doesn't care about the order in which those elements are present in collection (unlike collectionAssert.AreEqual test method). Look at example in Figure 99, both collection1 and collection2 have same elements but in different order.



```
[TestMethod]
public void CollectionAssert_AreEquivalent()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Gamma");
    collection2.Add("Beta");
    collection2.Add("Alpha");

    CollectionAssert.AreEquivalent(collection1, collection2);
}
```

Figure 99. Implementation of CollectionAssert.AreEquivalent Test Method

If we assert collection1 and collection2 using collecionAssert.AreEquivalent test method, it should get pass as both collections – collection1 and collection2 have same elements.

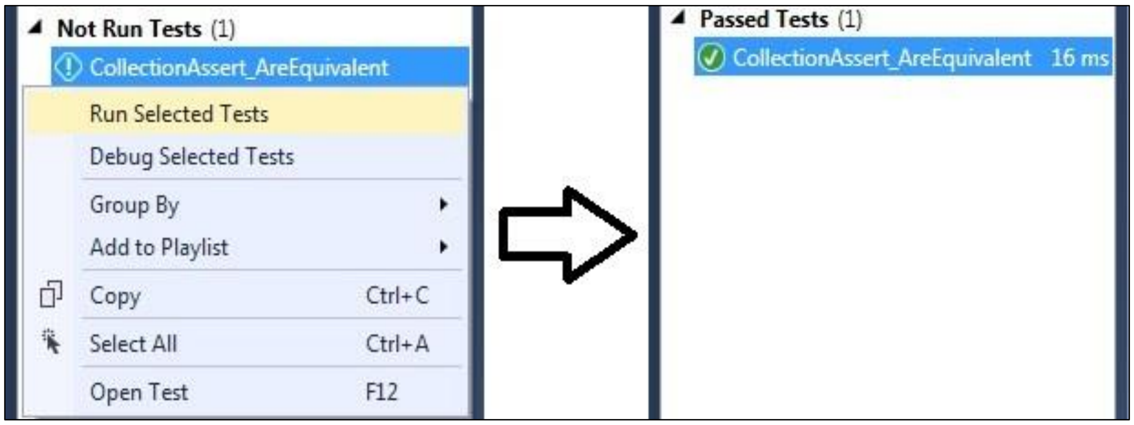
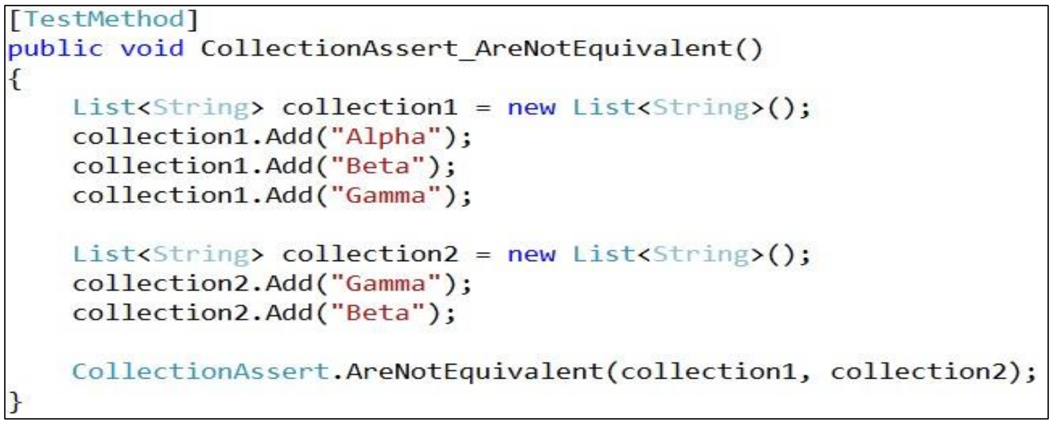


Figure 100. CollectionAssert.AreEquivalent Gets Pass

Unit test passes as expected (shown in Figure 100).

CollectionAssert.AreNotEquivalent

This test method is opposite of CollectionAssert.AreEquivalent test method. Here assertion gets pass if specified collections does not have same elements. If specified collections have same elements, then assertion get fail (Venema, 2015). In the example provide in Figure 101 collection1 have three elements “Alpha”,” Beta” and “Gamma” whereas collection2 only has “Gamma” and “Beta”.



```
[TestMethod]
public void CollectionAssert_AreNotEquivalent()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Gamma");
    collection2.Add("Beta");

    CollectionAssert.AreNotEquivalent(collection1, collection2);
}
```

Figure 101. Implementation of CollectionAssert.AreNotEquivalent Test Method

So when we run this unit test we expect it to pass (as shown in Figure 102) as collection1 and collection2 does not have same elements thus not equivalent.

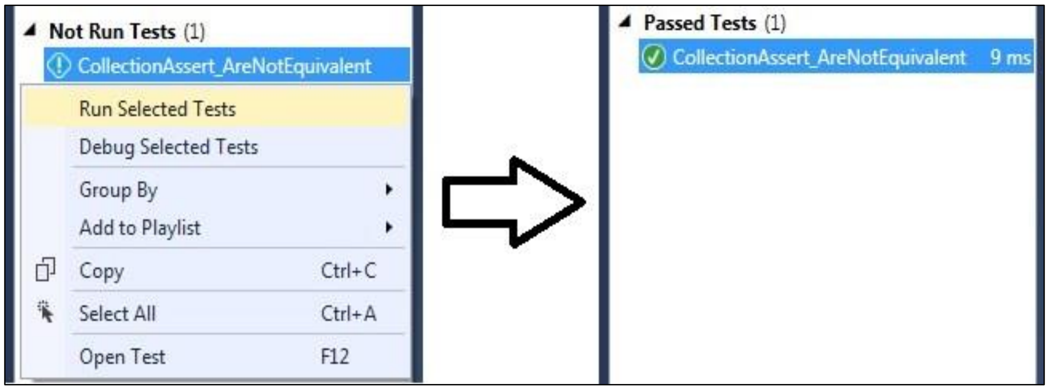


Figure 102. CollectionAssert.AreNotEquivalent Gets Pass

Unit test passes as expected.

CollectionAssert.Contains

This test method verifies if a given collection contains a given element/items. If we look at the CollectionAssert_Contains unit test provide in Figure 103, it has collection of strings called as collection1 with three items “Alpha”, “Beta” and “Gamma”.

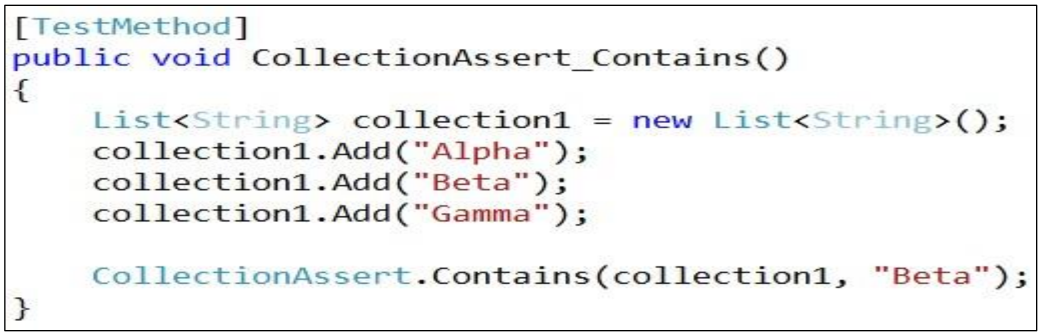

```
[TestMethod]
public void CollectionAssert_Contains()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    CollectionAssert.Contains(collection1, "Beta");
}
```

Figure 103. Implementation of CollectionAssert.Contains Test Method

Here CollectionAssert.contains test method is verifying if collection1 contains a string whose value is equal to “Beta”. Since it is true in this case, this unit test should succeed as shown in Figure 104.

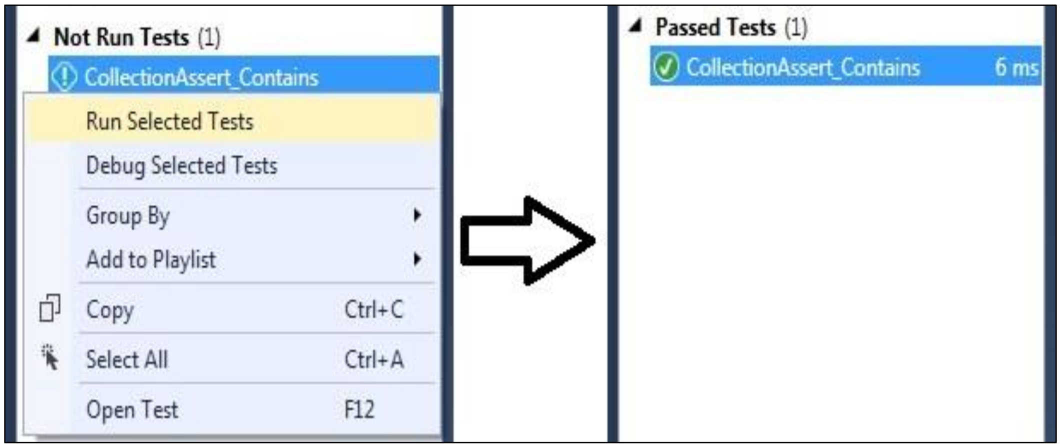


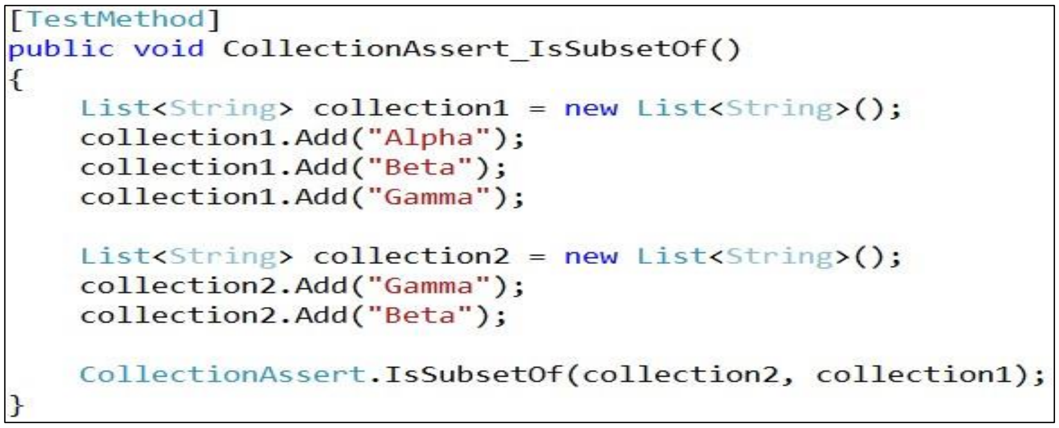
Figure 104. CollectionAssert.Contain Unit Test Gets Pass

As expected this unit test get pass. CollectionAssert.Contains test method does not care about the position of particular item it does only check whether particular item is available in provided collection or not.

CollectionAssert.IsSubsetOf

This test method verifies if a given collection is a subset of another given collection. Here in example provided in Figure 105, we got two collections – collection1 and collection2.

Collection1 has three string items “Alpha”, “Beta” and “Gamma” and collection2 has two string items “Alpha” and “Beta”.



```
[TestMethod]
public void CollectionAssert_IsSubsetOf()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Gamma");
    collection2.Add("Beta");

    CollectionAssert.IsSubsetOf(collection2, collection1);
}
```

Figure 105. Implementation of CollectionAssert.IsSubsetOf Test Method

If we carefully look at collection1 and collection2, collection2 is subset of collection1. Test method CollectionAssert.IsSubsetOf verifying same thing that is, if collection2 is subset of collection1 which is true here so when we run this unit test it should get pass as shown in Figure 106.

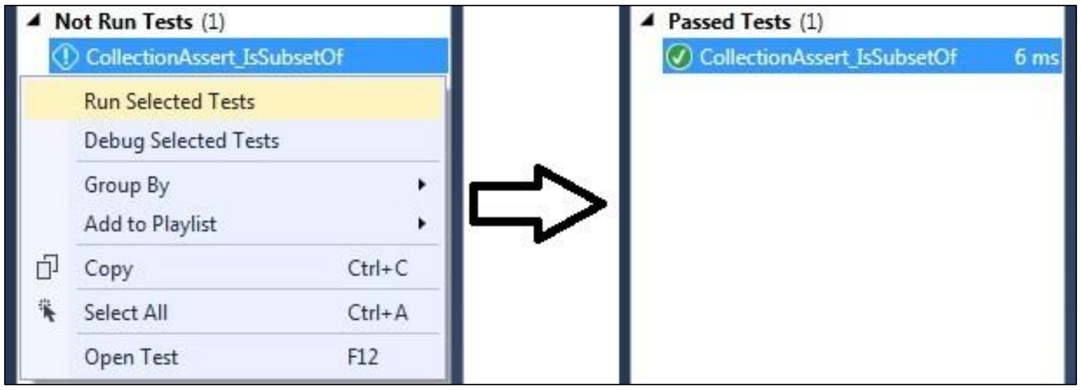
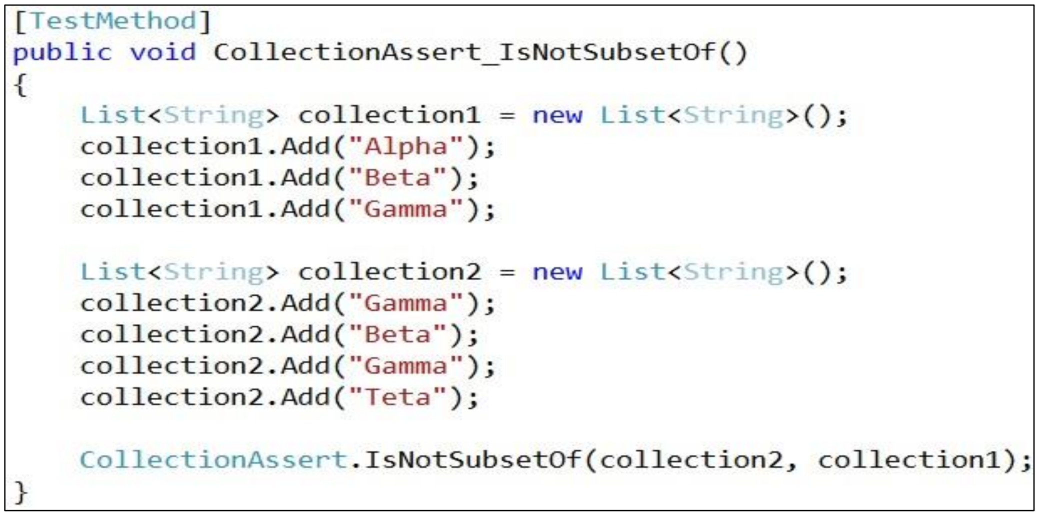


Figure 106. CollectionAssert_IsSubsetOf Unit Test Gets Pass

As expected this unit test gets pass.

CollectionAssert.IsNotSubsetOf

This test method is opposite of CollectionAssert.IsSubsetOf test method. This method verifies if a given collection is not a subset of another given collection. Here in example provided in Figure 107, we got two collections – collection1 and collection2. Collection1 has three string items “Alpha”, “Beta” and “Gamma” and collection2 has four string items “Alpha”, “Beta”, “Gamma” and “Teta”.



```
[TestMethod]
public void CollectionAssert_IsNotSubsetOf()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    List<String> collection2 = new List<String>();
    collection2.Add("Gamma");
    collection2.Add("Beta");
    collection2.Add("Gamma");
    collection2.Add("Teta");

    CollectionAssert.IsNotSubsetOf(collection2, collection1);
}
```

Figure 107. Implementation 0f CollectionAssert.IsNotSubsetOf Test Method

Here it is obvious that collection2 is not a subset of collection1. So this test method CollectionAssert.IsNotSubsetOf should get pass. Steps to run CollectionAssert.IsNotSubsetOf unit test are shown in Figure 108.

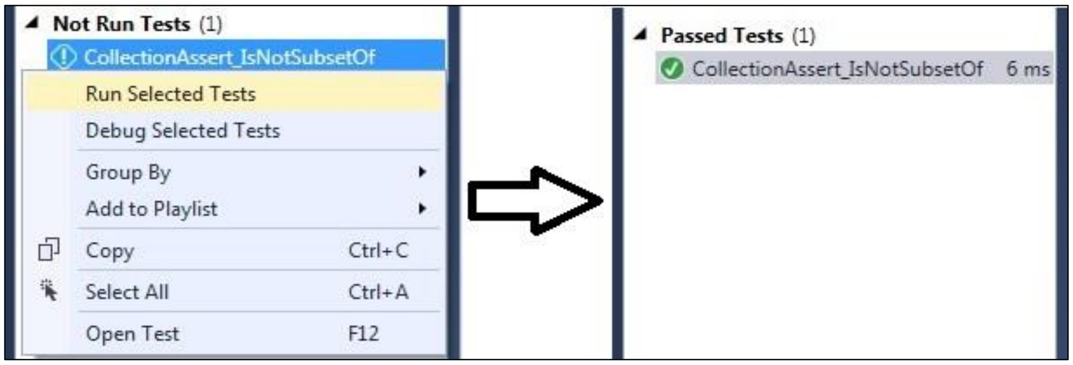
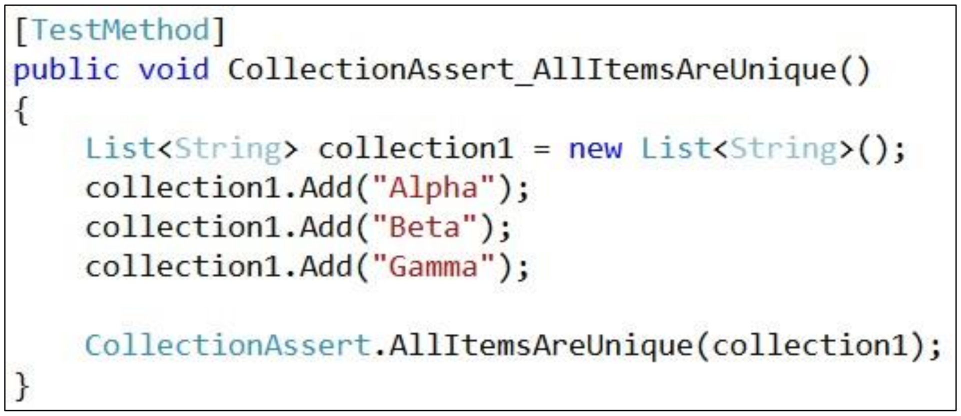


Figure 108. CollectionAssert_isNotSubsetOf Unit Test Gets Pass

As expected this unit test get pass which assert that collection 1 is not a subset of collection 2. This Assert method would compare any two collection at a given time.

CollectionAssert.AllItemsAreUnique

As name implies this method is going to verify if all the items in given collection are unique. If we look at collection1 given with CollectionAssert_AllItemsAreUnique unit test (Figure 109) all items has unique values – "Alpha", "Beta" and "Gamma".



```
[TestMethod]
public void CollectionAssert_AllItemsAreUnique()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    CollectionAssert.AllItemsAreUnique(collection1);
}
```

Figure 109. Implementation of CollectionAssert.AllItemsAreUnique Test Method

So when we run this unit test it should succeed as shown in Figure 110.

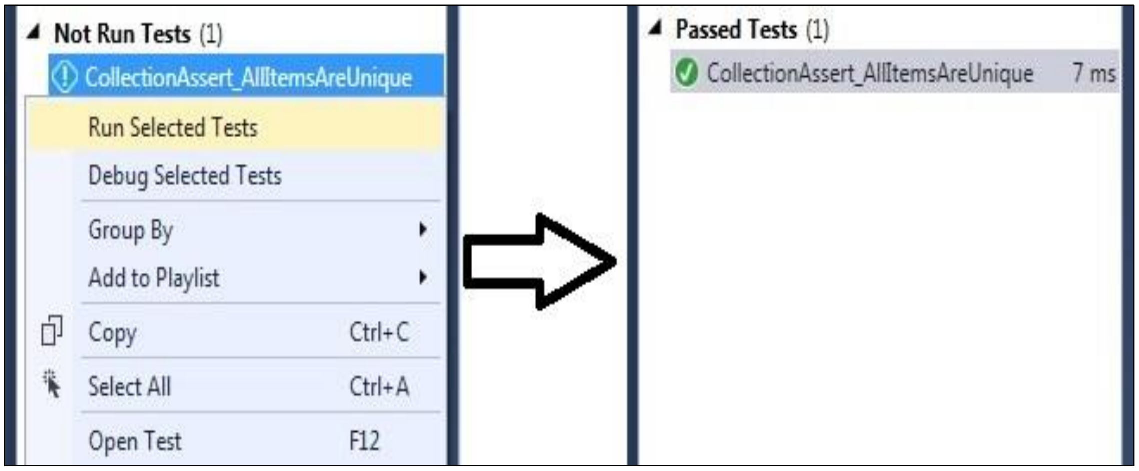
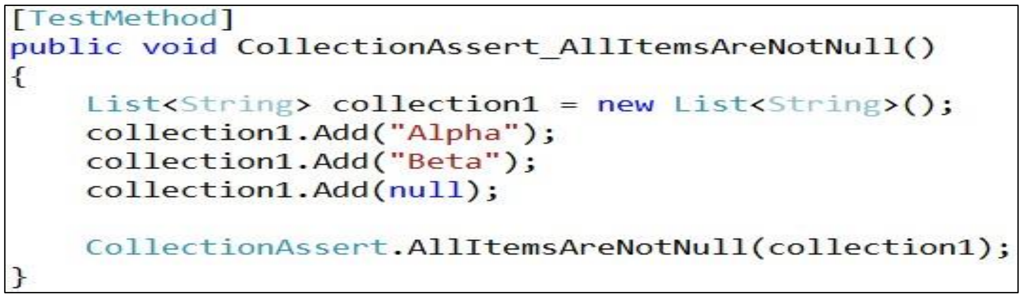


Figure 110. CollectionAssert_AllItemsAreUnique Unit Test Gets Pass

As expected this unit test gets pass.

AllItemsAreNotNull

This method is going to verify if all the items within the given collection are not null. If you look at the example given in Figure 111, collection1 has one null item.



```
[TestMethod]
public void CollectionAssert_AllItemsAreNotNull()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add(null);

    CollectionAssert.AllItemsAreNotNull(collection1);
}
```

Figure 111. Implementation of CollectionAssert.AllItemsAreNotNull (Has Null Value)

So when we run this unit test it should fail as collection1 contains null item, as shown in Figure 112.

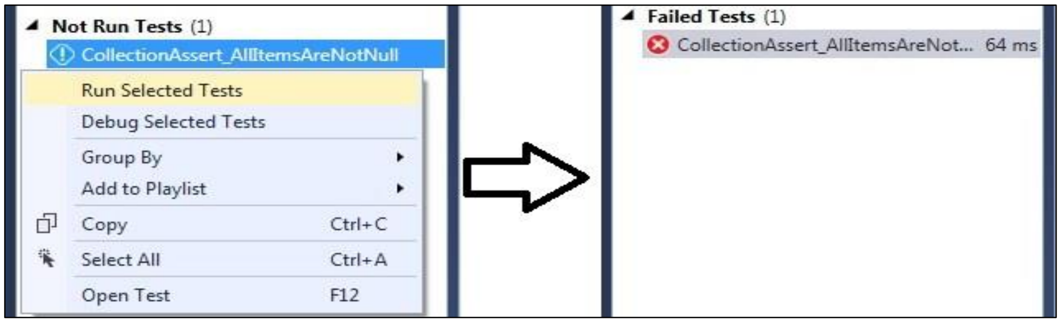
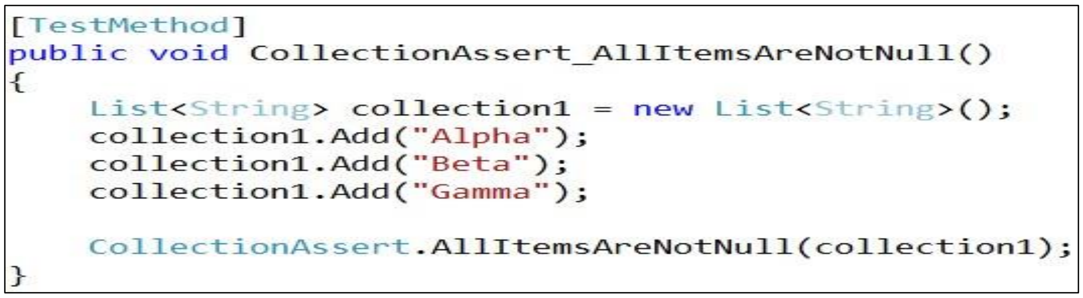


Figure 112. CollectionAssert_AllItemsAreNotNull Unit Test Gets Fail

Here unit test gets fail as expected. If we want to succeed this unit test, we have to get rid of null item, as shown in Figure 113 (KudVenkat, 2016).



```
[TestMethod]
public void CollectionAssert_AllItemsAreNotNull()
{
    List<String> collection1 = new List<String>();
    collection1.Add("Alpha");
    collection1.Add("Beta");
    collection1.Add("Gamma");

    CollectionAssert.AllItemsAreNotNull(collection1);
}
```

Figure 113. Implementation of CollectionAssert.AllItemsAreNotNull

So as now collection1 does not contain any null values CollectionAssert that contains unit test method AllItemsAreNotNull, should succeed as shown in Figure 114.

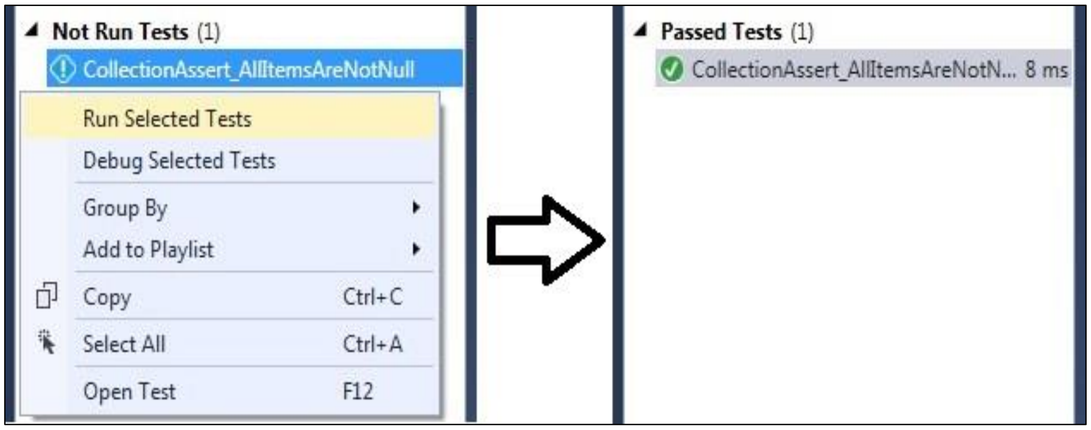


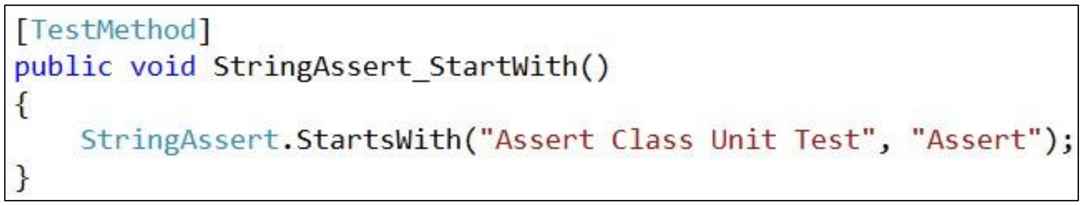
Figure 114. CollectionAssert_AllItemsAreNotNull Unit Test Gets Pass

StringAssert Class

StringAssert class provide several methods that verifies true and false proposition associated with strings in unit test (KudVenkat, 2016). The methods provided by this class are StringAssert.StartsWith, StringAssert.EndsWith, and StringAssert.Contains. Let's look these methods one by one with examples.

StringAssert.StartsWith

This method verifies if a given string starts with another given string (MSDN, 2013). Let's look at the StringAssert_StartWith method provided in Figure 115.



```
[TestMethod]
public void StringAssert_StartWith()
{
    StringAssert.StartsWith("Assert Class Unit Test", "Assert");
}
```

Figure 115. Implementation of StringAssert.StartWith Test Method

This method using StringAssert.StartWith assertion to verify if string "Assert Class Unit Test" starts with "Assert". Since it is true when we run this unit test it should succeed. Steps are provided in Figure 116. Please fallow these steps.

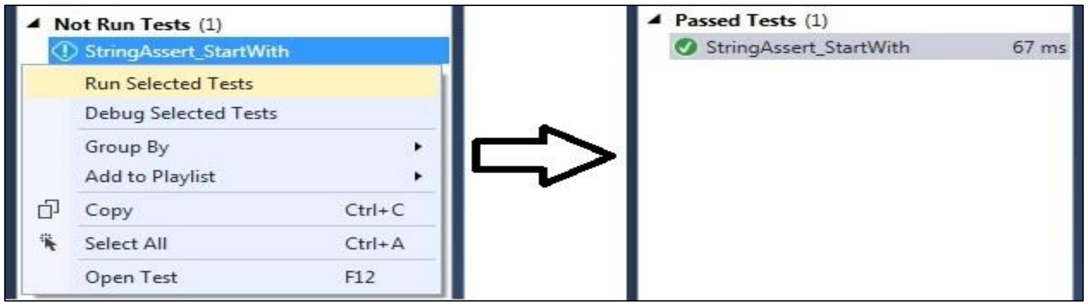
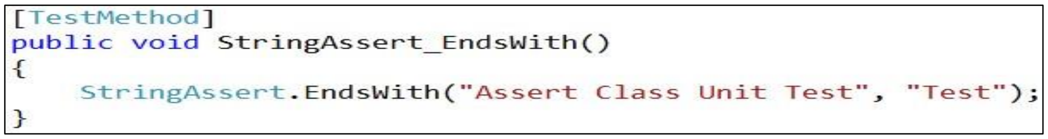


Figure 116. StringAssert_StartsWith Unit Test Gets Pass

StringAssert.EndsWith

This method verifies if a given string ends with another given string. Let's look at the StringAssert_EndsWith method provided in Figure 117.



```
[TestMethod]
public void StringAssert_EndsWith()
{
    StringAssert.EndsWith("Assert Class Unit Test", "Test");
}
```

Figure 117. Implementation of StringAssert.EndsWith Test Method

This method using StringAssert.EndsWith test method to verify if string "Assert Class Unit Test" ends with "Test". Since it is true when we run this unit test it should succeed as shown in Figure 118.

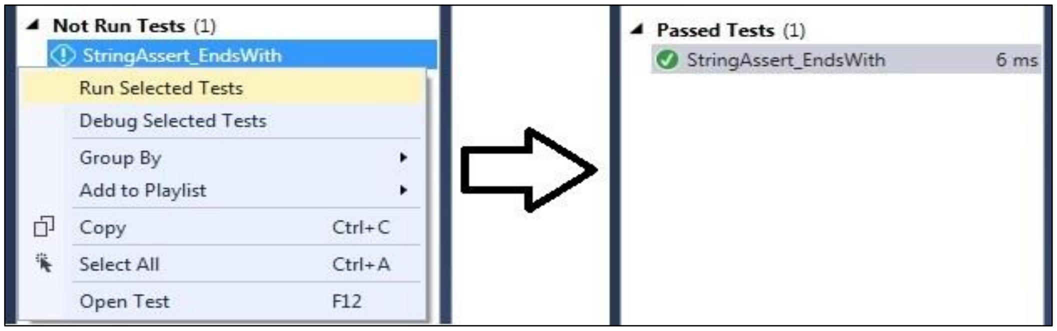
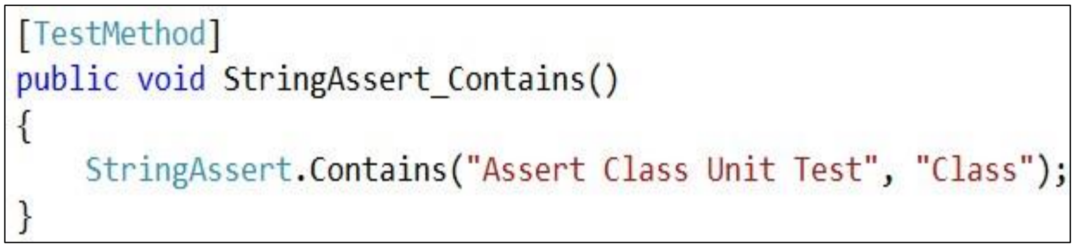


Figure 118. StringAssert_EndsWith Unit Test Gets Pass

StringAssert.Contains

This method verifies if given string contains another given string. Let's look at the StringAssert_Contains method provided in Figure 119.



```
[TestMethod]
public void StringAssert_Contains()
{
    StringAssert.Contains("Assert Class Unit Test", "Class");
}
```

Figure 119. Implementation 0f StringAssert.Contains Test Method

This method using StringAssert.Conatins test method to verify if string “Assert Class Unit Test” Contains “Class”. Since it is true when we run this unit test it should succeed. Steps to run StringAssert.Contains unit test are provided in Figure 120. Fallow these steps to run this unit test and it should get pass.

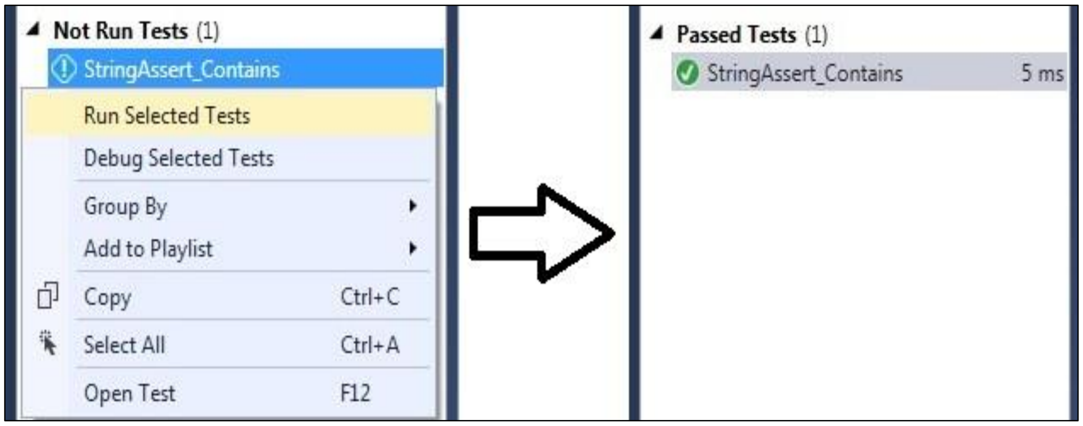


Figure 120. StringAssert_Contains Unit Test Gets Pass

Unit test passes as expected. This unit test could be very useful when we have substring from the string and we want to make sure that substring is in correct format. StringAssert_Contains is one of three StringAssert method. The other two StringAssert method as already discussed in this paper.

DISCUSSION AND CONCLUSION

The goal of this paper was to assist Dr. Gursimran Walia's research in Learning Object design that can be used in WReSTT to help students improve their understanding of testing concepts. Three learning objects were presented in this paper – learning object on unit test, learning object on data driven unit test, and learning object on assert classes of unit test. These learning objects comes under the Learning Content section of WReSTT, which apart from learning objects, also contains quizzes and learning strategies. As expected this unit test gets pass.

This paper has four objectives and we address each objective in each learning object. Our first learning object – unit test, provided fundamental concepts of unit tests, shown naming convention for unit test method, classes and project that generally fallow in all big industries, and provided a simple way to create a unit test, thus fulfilled our first objective. Our second leaning object – data driven unit test, provides documentation (with necessary screenshots) on creating data driven unit test with different sources (database, xml, and csv). This learning object also discuss about "regular expression", which means this learning object does not only fulfill our second objective but also provides extra useful information to students. Our third objective was to familiarize students with different assert classes of unit test, is fulfilled by third learning object – assert classes of unit test. Fourth and last objective is fulfilled by quizzes (pre/posttest). These quizzes would help in assessing student's knowledge while taking learning objects.

Quizzes are provided for all three learning objects which would help in analyzing student's knowledge growth by using learning objects. Each learning objects also has practice quiz, which would be given to students before actual quiz to make them accustom to quiz pattern. Actual quiz would act as pre and post-test.

REFERENCES

KudVenkat. (2016). Retrieved from www.udemy.com: https://www.udemy.com/mstest-unit-testing-tutorial-for-beginners/learn/v4/t/lecture/4319632?start=505

MSDN. (2013). *CollectionAssert Class*. Retrieved from https://msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.collectionassert.aspx

MSDN. (2013). *StringAssert Class*. Retrieved from https://msdn.microsoft.com/: https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.stringassert.aspx

MSDN. (2015). *How To: Create a Data-Driven Unit Test*. Retrieved from msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/ms182527.aspx

MSDN. (2015). *Walkthrough: Using a Configuration File to Define a Data Source*. Retrieved from https://msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/ms243192.aspx

MSDN. (2016). *DataSourceAttribute Class*. Retrieved from https://msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.datasourceattribute.aspx

MSDN. (n.d.). *Assert Class*. Retrieved from https://msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx

Network, D. (2005). *https://msdn.microsoft.com*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ms404699(v=vs.80).aspx

Netwrok, M. D. (2016, July 7). *https://msdn.microsoft.com*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/hh694602.aspx

Osherove, R. (2012). *Art of Unit Testing*. Retrieved from http://artofunittesting.com/: http://artofunittesting.com/

Peter J. Clarke, D. D. (October 2014). Integrating Testing into Software Engineering Courses Supported by a Collaborative Learning Environment. *ACM Trans. Comput. Educ*, 33.

Phlaz. (2012, November 14). *TestContext.DataRow null in Data Driven Test*. Retrieved from Social.msdn.microsoft.com: https://social.msdn.microsoft.com/Forums/vstudio/en-US/8df9d95c-3408-4bbe-b14a-0e179b950ec5/testcontextdatarow-null-in-data-driven-test?forum=vsunittest

Prasad Honrao. (n.d.). *Back to Basics: Data Driven Unit Testing*. Retrieved from http://prasadhonrao.com: http://prasadhonrao.com/back-to-basics-data-driven-unit-testing/

Regular Expression. (2016, 7 18). *How to Find or Validate an Email Address*. Retrieved from http://www.regular-expressions.info: http://www.regular-expressions.info/email.html

Stackoverflow. (2012, 13). *C# Unit Testing - XML Datasource containing multiple tests*. Retrieved from http://stackoverflow.com: http://stackoverflow.com/questions/14288902/c-sharp-unit-testing-xml-datasource-containing-multiple-tests

Technet. (2012, 10 15). *INSERT Examples (Transact-SQL)*. Retrieved from https://technet.microsoft.com: https://technet.microsoft.com/en-us/library/dd776381(v=sql.105).aspx

Timothy C. Lethbridge, J. D.-H. (2007). Improving software practice through education. *IEEE, Los Alamitos*, 12-28.

Todd Meinershagen. (2011, 2 11). *Creating Data-Driven Tests in MS Test*. Retrieved from https://toddmeinershagen.blogspot.com: https://toddmeinershagen.blogspot.com/2011/02/creating-data-driven-tests-in-ms-test.html

Venema, M. (2015, 1 24). *Why CollectionAssert.AreEqual fails even when both lists contain the same items*. Retrieved from http://softwareonastring.com/: http://softwareonastring.com/357/why-collectionassert-areequal-fails-even-when-both-lists-contain-the-same-items

Wikipedia. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Software_development

APPENDIX – QUIZZES OF LEARNING OBJECTS

Practice Quiz for Learning Object 1 - Unit Test

1) Why we use unit test?

    a) Unit tests are used for testing application as whole.

    b) Unit tests are used for testing IDE.

    c) Unit tests are used for testing unit of work.

    d) Unit tests are used for memory testing.

    Answer: c

2) How we define test class in MS tests,

    a) [Test Class]

    b) [Test Method]

    c) [Test Project]

    d) [Test Solution]

    Answer: a

    Explanation: For defining test class we decorate it with [Test Class] attribute and for defining test method we decorate it with [Test Method] attribute. There is no such method as [Test Project] and [Test Solution].

3) [Test Class] attribute contains

    a) All classes

    b) All methods

    c) All test classes

    d) All test methods

Answer: b

Explanation: A [Test Class] contain all its test methods. A [Test Class] cannot have any other class within it and all the methods within it should be test methods only.

4) Where we define [Test Method] attribute
   a) Before defining each test method
   b) Before defining each test class
   c) After defining each test method
   d) After defining each test class

   Answer: a

   Explanation: We define [Test Method] before each test method and [Test Class] attribute before each test class.

5) We use [Test Class] and [Test Method] attributes because,
   a) It gives better understanding to programmers about test class and test method
   b) These attribute confirm to complier about test class and test method
   c) Both a and b
   d) Neither a nor b

   Answer: c

   Explanation: [Test Class] and [Test Method] attribute confirm to compiler about test class and test method and at the same time it helps programmers to distinguish between class and test class.

6) What is AAA (Arrange Act Assert) pattern in unit test
   a) It's a naming convention pattern for unit tests.
   b) It's a pattern for developing unit tests.

c) It’s a pattern for developing any unit tests.

d) It’s a pattern

Answer: b

7) What is the use of “Act” in AAA pattern?

a) It used to initialize object and set the value of the data that is passed to the method being tested

b) It invokes method outside of current project

c) It invokes the method being tested

d) It used to verify that the method being tested behaves as excepted or not.

Answer: c

Explanation: “Act” is second section of AAA pattern which invoke the actual method which programmer going to test

8) Why programmer emphasis to give meaningful names to their unit test?

a) It gives good idea to programmers about the particular unit test without looking into it.

b) It is necessary for compiler to have meaningful names to unit tests.

c) It’s a way to kill time by programmers in thinking and writing name of unit tests.

d) Programmers does not care about name of unit test.

Answer: a

Explanation: Providing meaningful name to unit test help programmers or novice person in understanding the functionality of unit test without looking deeply into code.

9) Roy Osherove’s naming conventions

a) Used for validating unit test

b) Used for naming convention of unit test

c) Used for writing unit test

d) Used for developing application

Answer: b

10) Suppose we have a multiply() method which accept two integer as arguments (FirstNumber, SecondNumber) and returns its multiplication and we want to create unit test for this method where we want to pass to two positive integers as arguments. One combination of FirstNumber, SecondNumber and Expected values could be

a) FirstNumber= 5,SecondNumber= 4,Expected= 20,

b) FirstNumber= -5,SecondNumber= 4,Expected= -20,

c) FirstNumber= 5,SecondNumber= -4,Expected= -20,

d) FirstNumber=,-5,SecondNumber= -4,Expected= 20,

Answer: a

Explanation: According to question both integers should be positive and it will return their multiplication which is only satisfying by option a.

11) What would be the name of above unit test according to Roy Osherove's naming convention

a) RovOsherove_multiply()

b) test_multiply()

c) muliply_PositiveIntegers_ReturnPositiveMultiplication()

d) multiplyTest()

Answer: c

Explanation: According to Roy's Osherove's naming convention, name of any unit test should be divided into three parts. First part should tell about name of method being tested, which is in this case, is multiply. Second part tells about input parameter, which is in this case, is positive integers and third part tells about return value of function, which is in this case, is multiplication which would be positive as both input parameter is positive.

12) We decorate each test method within test class with [Test Method] attribute

   a) True

   b) False

   Answer: a

   Explanation: We decorate each test method with [Test Method] attribute.

13) Is it possible to run unit test a   utomatically after build using Visual Studio IDE?

   a) True

   b) False

   Answer: a

   Explanation: If we navigate to Test>>Test Settings and select "Run Tests After Build" then after every build unit test run automatically

Quiz for Learning Object 1 – Unit Test

1) What is MS Test

   a) It's a framework for unit tests

   b) It's an alternative name for unit tests

   c) It's a test for programmers

   d) It's a software from Microsoft.

   Answer: a

2) We define test methods in MS unit tests as

   a) [Test Class]

   b) [Test Method]

c) [Test Project]

d) [Test Solution]

Answer: b

Explanation: we define test method with [Test Method] attribute.

3) A [Test Methods] attribute contains

a) All methods

b) Single methods

c) All test methods

d) Single test method

Answer: d

Explanation: one [Test Method] attribute contains no more than one test method. A [Test Method] also cannot contain any method other than test method

4) Where we define [Test Class] attribute

a) Before defining each test method

b) Before defining each test class

c) After defining each test method

d) After defining each test class

Answer: b

Explanation: We define [Test Class] attribute before each test class and [Test Method] before each test method.

5) What is the use of "Arrange" in AAA pattern?

a) It used to initialize object and set the value of the data that is passed to the method being tested

b) It invokes method outside of current project

c) It invokes the method being tested

d) It used to verify that the method being tested behaves as excepted or not.

Answer: a

Explanation: “Arrange” is the first section of AAA pattern which is used to initialize object and set the value of data that is passed to method being tested (which is in most called as “Expected” value).

6) What is the use of “Assert” in AAA pattern?

   a) It used to initialize object and set the value of the data that is passed to the method being tested

   b) It invokes method outside of current project

   c) It invokes the method being tested

   d) It used to verify that the method being tested behaves as excepted or not.

   Answer: d

   Explanation: “Assert” section is third and last section of AAA pattern which used to verify the invoke method behaves as expected or not using “Assert” property.

7) According to Roy Osherove’s naming conventions we should define name of unit test as

   a) [ArbitraryName]

   b) [Test_MethodName]

   c) [RoyOsherove’s_MethodName]

   d) [UnitofWork_StateUnderTest_ExpectedBehaviour]

   Answer: d

Explanation: According to Roy Osherove's naming convention name of any unit test should be divided into three parts. *UnitOfWork* is name of the method being tested, *StateUnderTest* represent the input values for the method and *ExpectedBehaviour* is what the method returns for the specified input.

8) Suppose we have a multiply() method which accept two integer as arguments (FirstNumber, SecondNumber) and returns its multiplication and we want to create unit test for this method where we want to pass one positive integers and one negative integer as arguments. one combination of FirstNumber, SecondNumber and Expected value could be

   a) FirstNumber= 5, SecondNumber= 4, Expected= 20,

   b) FirstNumber= -5, SecondNumber= 4, Expected= -24,

   c) FirstNumber= 5, SecondNumber= -4, Expected= -20,

   d) FirstNumber=,-5, SecondNumber= -4, Expected= 20,

   Answer: c

   Explanation: According to question one integer should be positive and one should be negative and it will return their multiplication which is only satisfying by option c.

9) What would be the name of above unit test according to Roy Osherove's naming convention?

   a) RovOsherove_multiply()

   b) test_multiply()

   c) muliply_PositiveIntegerAndNegativeInteger_ReturnNegativeMultiplication()

   d) multiplyTest()

   Answer: c

Explanation: According to Roy's Osherove's naming convention, name of any unit test should be divided into three parts. First part should tell about name of method being tested, which is in this case, is multiply. Second part tells about input parameter, which is in this case, is one positive integer and one negative integer and third part tells about return value of function, which is in this case, is multiplication which would be negative.

10) How many unit tests can we develop for testing one method?

    a) 1

    b) 3

    c) 10

    d) No limit

    Answer: d

    Explanation: There are no limit on the number for developing unit test for particular method,

11) We decorate each test class with [Test Class] attribute

    a) True

    b) False

    Answer: a

    Explanation: We need to decorate each test class with [Test Class] attribute

12) Can we run multiple selected unit test within a test class using Visual Studio IDE?

    a) True

    b) False

    Answer: a

    Explanation: There are many ways within Visual Studio for running multiple unit test.

Practice Quiz for Learning Object 2 – Data Driven Unit Test

1) What is the use of TestContext class in the Data Driven unit test?

   a. It provides access to data row.

   b. It provides class for unit test.

   c. TestContext is attribute required for any unit test.

   d. It requires for connection to database.

   Answer: a

   Explanation – TestContext has many properties and DataRow is one of them.

2) What is data driven unit test?

   a. It allows unit tests to communicate with each other.

   b. It allows us to conduct unit test without any flaws.

   c. It allows us to store data in data source with unit test.

   d. It allows us to use data from data source with unit test.

   Answer: d

   Explanation: data driven unit test uses data source (either from database, xml or csv) for unit test so that single unit test may have multiple values to test.

3) For any data driven unit test data source could be,

   a. Databases

   b. Xml

   c. Csv

   d. Either of them

   Answer: d

Explanation: Data driven unit test accept data either from databases, xml file or csv file.

4) What is the functionality of DataRow property in TestContest class (TestContest.DataRow)

   a. It retrieves column values of given database table.

   b. It stores column values on given database table.

   c. It converts given data into table format (column and row).

   d. It converts given table formatted data into comma separated values (csv)

   Answer: a

   Explanation: DataRow property of TestContest retrieves column values of given database table.

5) Does it necessary for xml/csv file to put in "output folder" so that it can behave as data source

   a. Yes

   b. No

   Answer: a

   Explanation: It is necessary to put xml/csv file to put in "output folder" otherwise it throw an error "object reference not set to an instance of an object".

6) What is the functionality of "Match" method in data driven application?

   a. It matches current value with provided value

   b. It deletes current value if it does not match with provided value.

   c. It deletes provided value if it does not match with current value.

   d. It copies current value on provided value.

   Answer: a

Explanation: Match function matches current value with provided value.

7) We decorate every unit test class with

   a. [TestMethod]

   b. [Method-Test]

   c. [TestClass]

   d. [Class-Test]

   Answer: c

   Explanation: We suppose to decorate unit test class with [TestClass] and method with [TestMethod].

8) "DataRow" is property of

   a. DataConnect

   b. TestContext

   c. Match

   d. Regex

   Answer: b

   Explanation: DataRow is property of TestContext, TextContext.DataRow.

9) System.Data reference is required in data driven application

   a. Yes

   b. No

   Answer: a

   Explanation: System.Data is required in data driven application

Actual Quiz for Learning Object 2 – Data Driven Unit Test

1) What is TestContext?

   a. It’s an interface.

   b. It’s an abstract class

   c. It’s a method.

   d. It’s a enum.

   Answer: b

   Explanation: TestContext is abstract class.

2) What are the benefits of data driven unit test?

   a. It useful to test variety of unit test using single unit test.

   b. It useful to test variety of input using single unit test method.

   c. It generates automatic input for any unit test.

   d. It generates automatic unit test for any project.

   Answer: b

   Explanation: Main advantage of data driven unit test is to provide multiple input for single unit tests.

3) Which class do we use in data driven applications for connection to data source.

   a. TestContext

   b. TestMethod

   c. DataSource

   d. DataAccessMethod

   Answer: c

Explanation: We use DataSource class to connected to any data source (database, xml or csv files)

4) What does Assert.IsNotNull() property do?

   a. It makes sure that provided value is not empty.

   b. It provides random value if given value is null.

   c. It checks whether provided value is empty or not.

   d. It deletes value (if not null) to make sure it null.

   Answer: c

   Explanation: Assert.IsNotNull() check whether provided value is empty or not.

5) Which property of "Match" do we use to check whether match is successful or not.

   string s1 = "This is for quiz.";

   string s2 = "This is also for quiz"

   Mach match = s1.Match(s2);

   a. Match.Success()

   b. Match.IsSuccess()

   c. Match.Success

   d. Match.IsSuccess

   Answer: c

   Explanation: Mach.IsSuccess is not any property under Match. Parenthesis in Match.Success() and Match.IsSuccess() signifies that it method not property. Match.Success is correct answer.

6) Can we debug data driven unit test?

a. Yes

b. No

Answer: a

Explanation: Visual Studio IDE provided us facility to debug unit test.

7) To Run Unit test

a. Go to “Solution Explorer”, right click on desired unit test project and select “Start”.

b. Go to “Test Explorer”, right click on desired unit test and select “Run Selected Test”.

c. Go to “Class View”, select desired unit test and expend it.

d. Go to “Solution Explorer”, right click on unit test and select “Properties”

Answer: b

Explanation: “Test Explorer” is designed for unit tests. We cannot run unit tests using “Solution Explorer”.

8) Regex is used for

a. Regular Expression

b. Lambda Expression

c. Mathematics Expression

d. Linq

Answer: a

Explanation: Regex is abbreviated for Regular Expression

9) Which property do we use to connect to database in data driven unit test

a. [Test Method]

b. [Test Class]

c. DataSource

d. DataRow

Answer: c

Explanation: To connect to any kind of data source we use "DataSource" property.

10) CurrentTestOutcome, DataConnect, DataRow, DataConnection are properties of which class?

a. DataConnect

b. TestContext

c. Match

d. Regex

Answer: TestContext

Explanation: TestContext have the properties of TestContext CurrentTestOutcome, DataConnect, DataRow, DataConnection

Practice Quiz for Learning Object 3 – Assert Classes of Unit Test

1) Test method Assert.AreNotEqual() asserts

a. Expected object is not equal to actual object.

b. Expected object is equal to actual object.

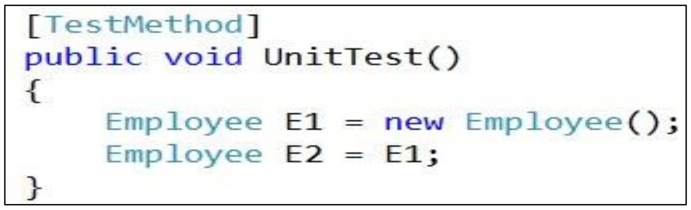
c. Expected ICollection is equal to actual ICollection.

d. Expected string is not equal to actual string.

Answer: a

Explanation - Assert.AreNotEqual() test method asserts that expected object is not equal to actual object. For ICollection and string we use AssertCollection and AssertString assertion respectively.

2) In below given unit test which test method successfully assert that Employee E1 and E2 pointing to same object.



```
[TestMethod]
public void UnitTest()
{
    Employee E1 = new Employee();
    Employee E2 = E1;
}
```

Figure A1. Question 2 of Practice Quiz for Assert Classes

a. StringAssert.Match(E1,E2);

b. Assert.IsTrue(E1,E2);

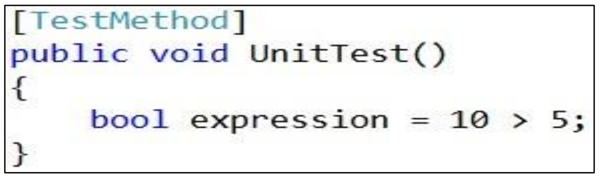
c. Assert.AreSame(E1,E2);

d. CollectionAssert.AreEqual(E1,E2);

Answer: c

Explantion - Here we suppose to compare object so Assert class would be applicable not CollectionAssert and StringAssert classes. Assert.IsTrue() test method assert that provided object is true or false, which is not applicable over here. Asser.AreSame() test method assert if expected object and actual object pointing to same object hence this is right answer.

3) How to assert that fallowing expression is true



```
[TestMethod]
public void UnitTest()
{
    bool expression = 10 > 5;
}
```

Figure A2. Question 3 of Practice Quiz for Assert Classes

a. StringAssert.Matches(expression);

b. Assert.IsTrue(expression);

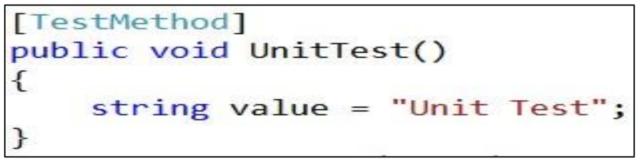
c. Assert.IsSame(expression);

d. CollectionAssert.AllItemsAreUnique(expression);

Answer: b

Explanation -This expression is not string or collection thus StringAssert and CollectionAssert does not work here. Assert.IsSame() compares objects so that's too is not applicable. Only Assert.IsTrue() test method succeed successfully if specified expression/condition is true.

4) Which test method successfully assert that fallowing string (value) is not empty.



```
[TestMethod]
public void UnitTest()
{
    string value = "Unit Test";
}
```

Figure A3. Question 4 of Practice Quiz for Assert Classes

a. Assert.IsNull();

b. Assert.IsNotNull(value);

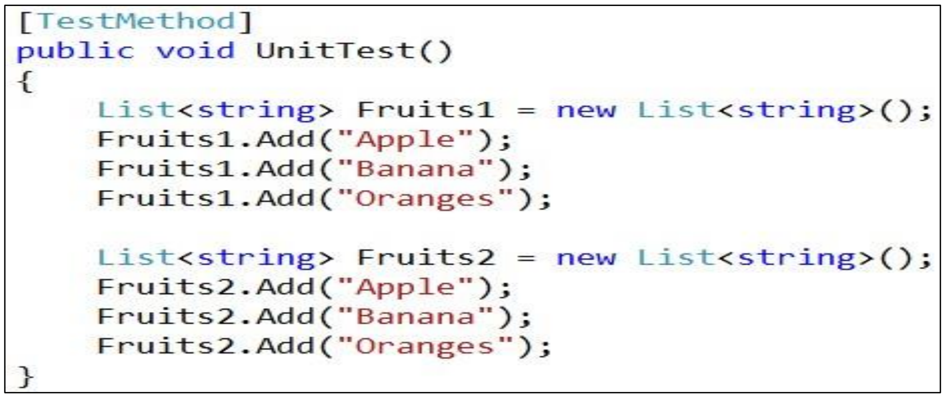
c. Assert.IsNotNull();

d. Assert.IsNull(value);

Answer-b

Explanation - Assert.IsNotNull() test method assert that specified value is not null (empty). So answer is either b or c. Assert.IsNotNull() test method expects value for assertion otherwise it will throw compile time error. Here option b passing value, hence it is right answer.

5) Which test method successfully assert that fallowing Fruits1 and Fruits2 collections contains same items in same order.



```
[TestMethod]
public void UnitTest()
{
    List<string> Fruits1 = new List<string>();
    Fruits1.Add("Apple");
    Fruits1.Add("Banana");
    Fruits1.Add("Oranges");

    List<string> Fruits2 = new List<string>();
    Fruits2.Add("Apple");
    Fruits2.Add("Banana");
    Fruits2.Add("Oranges");
}
```

Figure A4. Question 5 of Practice Quiz for Assert Classes

a. CollectionAssert.Contains(Fruits1, Fruits2);

b. StringAssert.Contains(Fruits1, Fruits2);

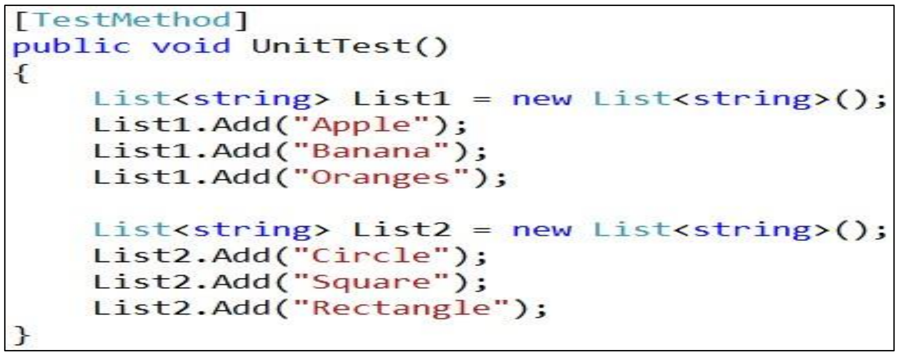
c. CollectionAssert.AreEqual(Fruits1, Fruits2);

d. CollectionAssert.AreEquivalent(Fruits1, Fruits2);

Answer-c

Explanation – CollectionAssert.Contains() test method succeed assertion if a given collection contains a given item which is not applicable here. StringAssert does not work with collection. CollectionAssert.AreEquivalent() test method ensure that both collections contain same elements which is true in this case so this assertion would succeed but this test method does not ensure about the order of items. CollectionAssert.AreEqual() test method ensure that both collections contain same items in same order hence CollectionAssert.AreEqual() is right answer.

6) Which test method successfully assert that List1 and List2 collections does not contains same items



```
[TestMethod]
public void UnitTest()
{
    List<string> List1 = new List<string>();
    List1.Add("Apple");
    List1.Add("Banana");
    List1.Add("Oranges");

    List<string> List2 = new List<string>();
    List2.Add("Circle");
    List2.Add("Square");
    List2.Add("Rectangle");
}
```

Figure A5. Question 6 of Practice Quiz for Assert Classes

a. CollectionAssert.AreNotEquivalent(Lis1, List2);

b. CollectionAssert.AreNotEqual(Lis1, List2);

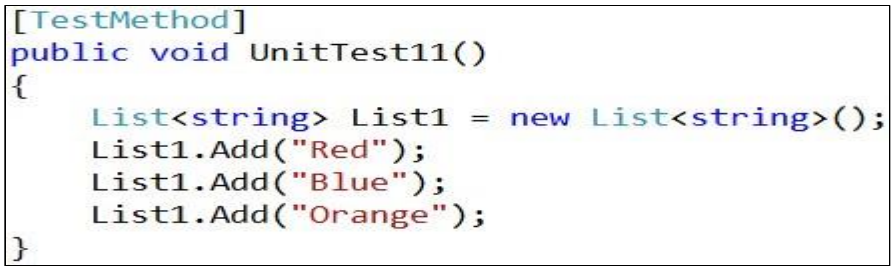
c. CollectionAssert.DoesNotContains(Lis1, List2);

d. Assert.AreNotSame(List1, List2);

Answer-a

Explanation – Assert.AreNotSame() does not work with collections. CollectionAssert.DoesNotContains() test method succeed assertion if a given collection does not contain a given item which is not applicable here. CollectionAssert.AreNotEquivalent() and CollectionAssert.AreNotSame() both succeed assertions but only CollectionAssert.AreNotEquivalent () ensure that List1 and List2 does not contains same items.

7) Which test method successfully assert that List1 does not contains “White” elements



```
[TestMethod]
public void UnitTest11()
{
    List<string> List1 = new List<string>();
    List1.Add("Red");
    List1.Add("Blue");
    List1.Add("Orange");
}
```

Figure A6. Question 7 of Practice Quiz for Assert Classes

a. StringAssert.Contains(List1, “White”);

b. Assert.AreEqual(List1, “White”);

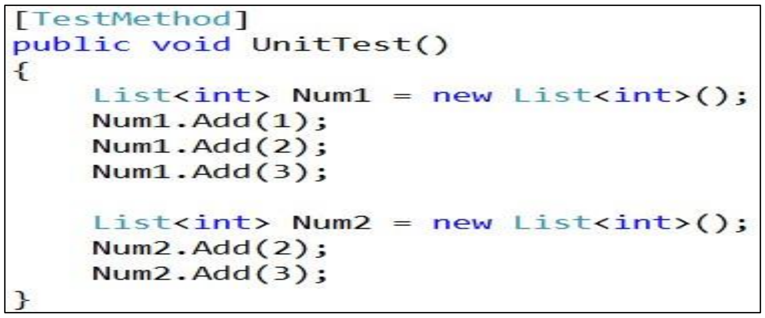
c. CollectionAssert.Contains(List1, “White”);

d. CollectionAssert.DoesNotContains(List1, “White”);

Answer-d

Explanation – Assert and StringAssert classes does not work with collections. CollectionAssert.DoesNotContains() test method successfully assert if a given collection does not contain a given item hence option d is correct.

8) Which test method successfully assert that Num2 collection is subset of Num1 collection.



```
[TestMethod]
public void UnitTest()
{
    List<int> Num1 = new List<int>();
    Num1.Add(1);
    Num1.Add(2);
    Num1.Add(3);

    List<int> Num2 = new List<int>();
    Num2.Add(2);
    Num2.Add(3);
}
```

Figure A7. Question 8 of Practice Quiz for Assert Classes

a. CollectionAssert.IsSubsetOf(collection1, collection2);

b. CollectionAssert.IsSubsetOf(collection2, collection1);

c. CollectionAssert.IsNotSubsetOf(collection1, collection2);

d. CollectionAssert.IsNotSubsetOf(collection2, collection1);

Answer-b

Explanation – CollectionAssert.IssubsetOf() test method ensure that given collection is a subset of another given collection. So it either from option a and b. It take 2 parameter as CollectionAssert.IsSubsetOf(subset,superset) and here Num2 is subset and Num1 is superset hence option b is correct.

9) Which test method successfully assert that specified collection does not have any null item?

a. Assert.IsNull()

b. Assert.IsNotNull()

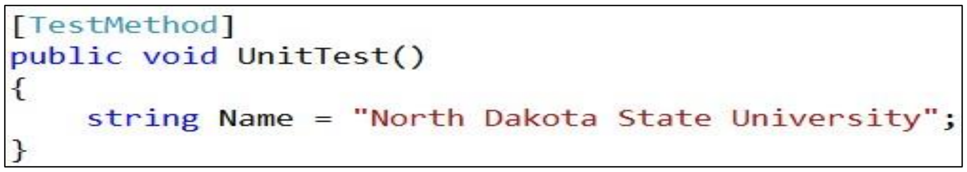
c. StringAssert.EndsWith()

d. CollectionAssert.AllItemsAreNotNull()

Answer-d

Explanation - As we know that Assert and StringAssert class does not work with collections so option a, b and c gets eliminated. CollectionAssert.AllItemsAreNotNull ensure that specified collection does not contain any null items.

10) Which test method successfully assert that "Name" string starts with "North"?



```
[TestMethod]
public void UnitTest()
{
    string Name = "North Dakota State University";
}
```

Figure A8. Question 10 of Practice Quiz for Assert Classes

a. Assert.isTrue("North");

b. StringAssert.StartsWith(Name, "North");

c. StringAssert.Contains(Name, “North”);

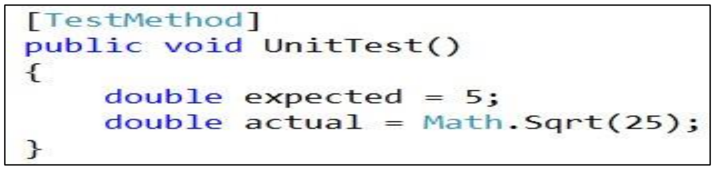
d. StringAssert.EndsWith(Name, “North”);

Answer-b

Explanation – Here StringAssert.StartsWith() and StringAssert.Contains() test method both pass their assertion but only StringAssert.StartsWith() test method ensure that given string starts with another given string.

Quiz for Learning Object 3 – Assert Classes of Unit Test

1) Which test method successfully assert fallowing unit test.



```
[TestMethod]
public void UnitTest()
{
     double expected = 5;
     double actual = Math.Sqrt(25);
}
```

Figure A9. Question 1 of Actual Quiz for Assert Classes

a. Assert.AreSame(expected, actual);

b. Assert.AreEqual(expected, actual);

c. CollectionAssert.AreEqual(expected, actual);

d. CollectionAssert.AreEquivalent(expected, actual);

Answer-b

Explanation – Assert.AreSame() test method assert if two object reference variable point to same object. CollectionAssert() test method works on collection. Only Assert.AreSame compare expected and actual method and if they match assertion gets pass.

2) Which test method successfully assert that reference variable M1 and M2 (ManagerUnitTest) pointing to different objects.

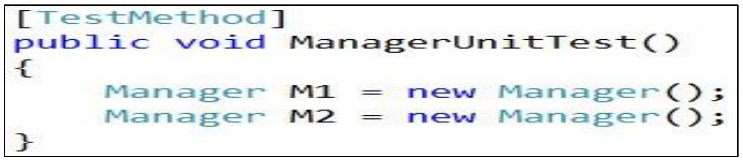

```
[TestMethod]
public void ManagerUnitTest()
{
     Manager M1 = new Manager();
     Manager M2 = new Manager();
}
```

Figure A10. Question 2 of Actual Quiz for Assert Classes

a. Assert.AreNotEqual(M1,M2);

b. Assert.IsInstanceOfType(M1,M2);

c. Assert.AreNotSame(M1,M2);

d. CollectionAssert.IsNoTSubsetOf(M1,M2);

Answer-c

Explanation – Assert.ArenotSame() test method ensure that provided reference variables does not point to same objects.

3) Assert.Fail() test method fails unit test without asserting any condition whatsoever. So what could be reason for developers to use this test method?

a. To intentionally sabotage unit tests.

b. To increase the number of fail unit test.

c. To set as reminder that particular unit test is incomplete or have flaws.

d. To correct Assert.Fail() test method

Answer-c

4) Which test method successfully assert that fallowing expression is not true.

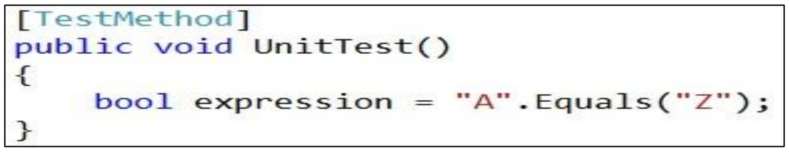

```
[TestMethod]
public void UnitTest()
{
     bool expression = "A".Equals("Z");
}
```

Figure A11. Question 4 of Actual Quiz for Assert Classes

a. Assert.IsTrue(expression);

b. Assert.IsFalse(expression);

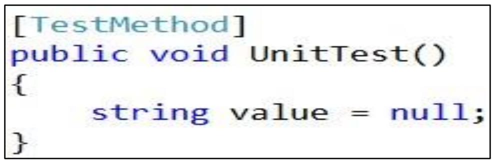
c. StringAssert.StartsWith(expression);

d. StringAssert.EndsWith(expression);

Answer-b

Explanation – Assert.IsTrue() test method will fail as this expression is not true. StringAssert class only works with strings so it is not applicable over here. Assert.IsFalse test method successfully assert if specified condition is false, hence it's right answer.

5) Which test method successfully assert that fallowing value is null.



```
[TestMethod]
public void UnitTest()
{
    string value = null;
}
```

Figure A12. Question 5 of Actual Quiz for Assert Classes

a. Assert.IsNull();

b. Assert.IsNotNull(value);

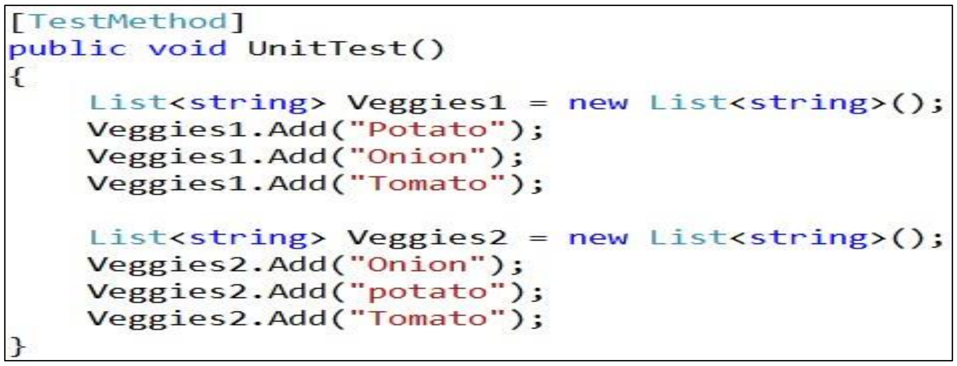
c. Assert.IsNotNull();

d. Assert.IsNull(value);

Answer-d

Explanation – Assert.IsNull() test method assert that specified value is null. So answer is either a or d. Assert.IsNull() test method expects value for assertion otherwise it will throw compile time error. Here option d passing value, hence it is right answer.

6) Which test method successfully assert that collection “Veggies1” and “Veggies2” contains same items regardless of their order.



```
[TestMethod]
public void UnitTest()
{
    List<string> Veggies1 = new List<string>();
    Veggies1.Add("Potato");
    Veggies1.Add("Onion");
    Veggies1.Add("Tomato");

    List<string> Veggies2 = new List<string>();
    Veggies2.Add("Onion");
    Veggies2.Add("potato");
    Veggies2.Add("Tomato");
}
```

Figure A13. Question 6 of Actual Quiz for Assert Classes

a. CollectionAssert.AreNotEqual (Veggies1, Veggies2);

b. CollectionAssert.AreNotEquivalent(Veggies1, Veggies2);

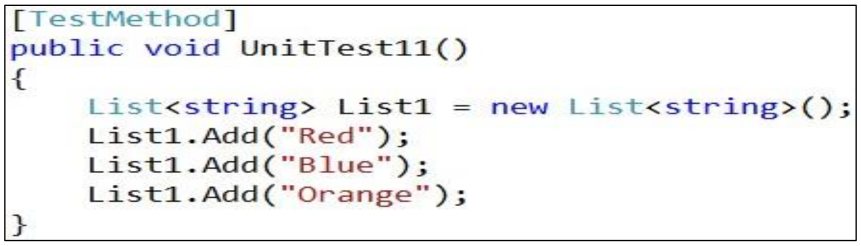
c. CollectionAssert.AreEquivalent(Veggies1, Veggies2);

d. CollectionAssert.AreEqual(Veggies1, Veggies2);

Answer-c

Explanation - CollectionAssert.AreNotEqual() will succeed assertion because it considers both collections different but it does not ensure that both collections contains same items. CollectionAssert.AreNotEquivalent () will fail assertion as both collections are equivalent and that is why CollectionAssert.AreEquivalent() will succeed assertion and will ensure that both collections contain same items regardless of its order.

7) Which test method syntax is correct for asserting that List1 has “Blue” items



```
[TestMethod]
public void UnitTest11()
{
    List<string> List1 = new List<string>();
    List1.Add("Red");
    List1.Add("Blue");
    List1.Add("Orange");
}
```

Figure A14. Question 7 of Actual Quiz for Assert Classes

a. CollectionAssert.Contains("List1", Blue);

b. CollectionAssert.Contains(List1, "Blue");

c. CollectionAssert.Contains(List1, Blue);

d. CollectionAssert.Contains("List1", "Blue");

Answer-b

Explanation – List1 is collection of type string and we do not enclose collection into inverted comma. Blue is string here so it will get enclose with inverted comma, hence b is correct option.

8) Which test method successfully assert that any specified collection does not have any duplicate.

a. CollectionAssert.AllItemsAreNotNull()

b. CollectionAssert.AllItemsAreInstanceOfType()

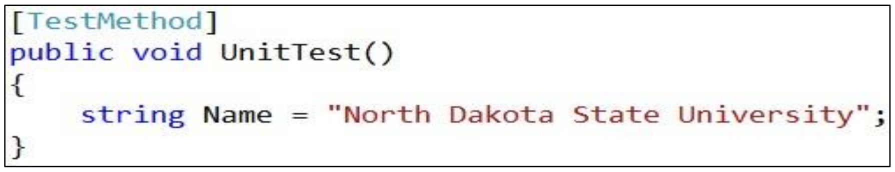
c. CollectionAssert.All ItemsAreUnique()

d. CollectionAssert.IsSubsetOf()

Answer-c

Explanation – CollectionAssert.AllItemsAreUnique ensure that all items in specified collection are unique.

9) Which test method successfully assert that "Name" string ends with "University"?



```
[TestMethod]
public void UnitTest()
{
    string Name = "North Dakota State University";
}
```

Figure A15. Question 9 of Actual Quiz for Assert Classes

a. Assert.lsTrue(“University”);

b. CollectionAssert.StartsWith(“University”);

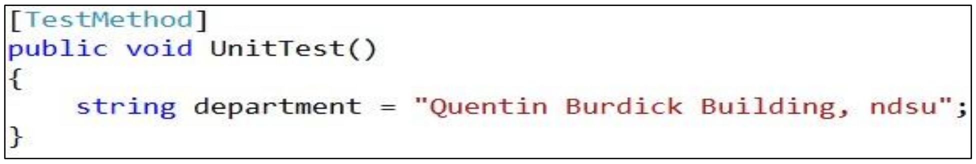
c. CollectionAssert.Contains(“University”);

d. CollectionAssert.EndsWith(“University”);

Answer-d

Explanation – Here both CollecttionAssert.Contains() and CollectionAssert.EndsWith() test method pass assertion but only CollectionAssert.EndsWith() ensure that given string ends with specified string.

10) Which test method successfully assert that “Department” string contains “Building”



```
[TestMethod]
public void UnitTest()
{
    string department = "Quentin Burdick Building, ndsu";
}
```

Figure A16. Question 10 of Actual Quiz for Assert Classes

a. Assert.Istrue(“Building”);

b. StringAssert.StartsWith(Department, “Building”);

c. StringAssert.Contains(Department, “Building”);

d. StringAssert.EndsWith(Department, “Building”);

Answer-c

Explanation – StringAssert.Contains() test method ensure that given string contains another given string hence option c is correct**.**