**Software Engineering Concepts & Methods  
Lab Exercise:   
Automated Unit Testing**

**Overview**

This exercise assumes that you have successfully completed the Manual Unit Testing exercise that tested the GetNearestDate method of the Scheduler class. In particular, that you have created the test cases required for “Basis Path” coverage. In this exercise, these test cases will be implemented as automated unit tests.

This lab exercise consists of the following sections:

[Create a Unit Test Project 2](#_Toc412309144)

[Write a Unit Test Method 3](#_Toc412309145)

[Run a Unit Test Method 4](#_Toc412309146)

[Write Four More Unit Test Methods 5](#_Toc412309147)

[Save Your Project 5](#_Toc412309148)

**IMPORTANT**

Make sure you have a method of saving your Visual Studio project at the end of the exercise for next week stuff.

# Create a Unit Test Project

In this section you will create a Unit Test project in the Shipping Solution in Visual Studio.

1. If you have not already done so:
   1. Copy the Shipping solution from Blackboard to the local hard drive
   2. If it is not already running, run Visual Studio
   3. Open your Shipping solution.
2. To create a Unit Test project:
   1. In the Solution Explorer, right mouse click on the **Shipping\_Solution**.
   2. In the popup menu select **Add** then **New Project…**
   3. In the Add New Project dialog
      1. In the list of project types on the left hand side, if necessary expand **Visual C#** and then select **Test.**
      2. In the list of project templates in the middle of the dialog, select **Unit Test Project**.
      3. Name the project **Shipping\_UnitTestProject**
      4. Click **OK**.
   4. In the Solution Explorer, rename the class UnitTest1 to **UnitTests\_Scheduler**. When prompted to rename all references to ‘UnitTest1’ click **Yes**.
3. Before you can use the Shipping\_UnitTestProject project to test methods in classes in the Shipping\_Project, you must add a reference to the Shipping\_Project.
   1. Right mouse click on **Shipping\_UnitTestProject**.
   2. In the popup menu select **Add** then **Reference**.
   3. In the Reference Manager - Shipping\_UnitTestProject dialog
      1. In the list of the left hand side, if necessary expand the **Solution** item and select **Projects**.
      2. In the middle of the dialog, check the checkbox next to the **Shipping\_Project** item - there should only be one.

*Note: the checkbox might not be visible until the mouse pointer hovers over the item.*

* + 1. Click **OK**.
  1. To make programming easier and the code more readable, you will now add two **using** directives for the Shipping\_Project and the other for Collections. In fact, the namespace of the Shipping\_Project is **Shipping**.

Copy the following lines of code and paste them immediately underneath the existing using directives in your Shipping\_UnitTestProject class.

using System.Collections.Generic;

using Shipping;

# Write a Unit Test Method

The unit test methods that you will write in this section will use the Basis Path test cases that you developed in the previous tutorial. In this section you will be shown how to create and run a test method based on one of the Basis Path test cases. You will then implement the rest of the test cases.

1. In the UnitTests\_Scheduler class, locate TestMethod1 and change its name to **Scheduler\_GetNearestDate\_ExactMatch** - naming convention ProductionClass\_Method\_TestCase.
2. The GetNearestDate method takes three arguments: requiredDate, daysFlexibility and voyagesList.

Each of these arguments has to be created with values that represent the test case. So after reading the descriptions in the following points, copy and paste the following lines of code into the Scheduler\_GetNearestDate\_ExactMatch method.

* 1. Declare the **requiredDate** and set it to any date, for example:

DateTime requiredDate = new DateTime(2015, 3, 3);

* 1. Because the Scheduler\_GetNearestDate\_ExactMatch test case is testing for an exact date match then the departure date of one of the voyages in the **voyagestList** must be the same as the required date you used above:

List<Voyage> voyagesList = new List<Voyage>();

Voyage voyage1 =

Factory.CreateVoyage(new DateTime(2015, 3, 3), "Olsen");

voyagesList.Add(voyage1);

Are any more voyages necessary?

* 1. In this test case the value of **daysFlexibility** is not used but should be set to zero for completeness:

int daysFlexibility = 0;

1. An instance of the Scheduler class must be created. This is because the GetNearestDate method is a member of the Scheduler class.

Copy the following line of code and paste it in to the Scheduler\_GetNearestDate\_ExactMatch method.

Scheduler myScheduler = Factory.CreateScheduler();

1. Now the GetNearestDate method can be executed by the following line of code. The return value is a string whose value will be tested in a subsequent statement in the Scheduler\_GetNearestDate\_ExactMatch method.

Copy the following line of code and paste it as the next line in the Scheduler\_GetNearestDate\_ExactMatch method.

string nearestDate =  
 myScheduler.GetNearestDate(

requiredDate, daysFlexibility, voyagesList);

1. Static or class methods on the Assert class are used to test whether or not the result of an operation is correct. In this case, you want to test if the return value - nearestDate - has the value you expect. So you use the **AreEqual** method.

The first argument is the **expected** value which in this specific test case is the text string:

Exact date: 03-Mar-2015 Olsen

The variable nearestDate is the actual value and is the second argument of the AreEqual method.

It is also usual to include a descriptive message in the event that the expected value and actual values are not equal. This is the third argument in the AreEqual method.

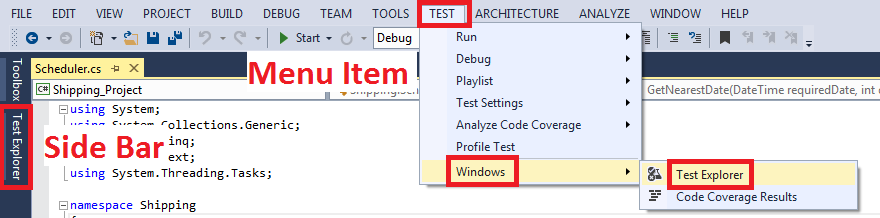
Copy the following code and paste it as the next line in the Scheduler\_GetNearestDate\_ExactMatch method:

Assert.AreEqual(  
 "Exact date: 03-Mar-2015 Olsen", // Expected value  
 nearestDate, // Actual value  
 "Exact Date Match NOT found"); // Error message

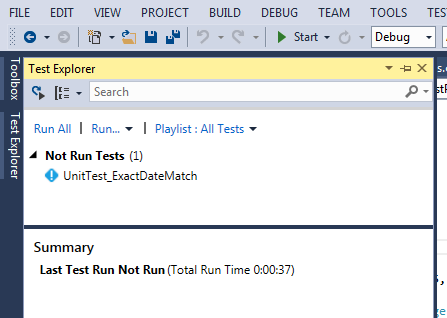
# Run a Unit Test Method

In this section you will run the Scheduler\_GetNearestDate\_ExactMatch unit test method that you created in the previous section. If the code in that method is not correct then you will have to debug and edit it.

1. Before you run any unit tests you must rebuild the solution. There are a couple of different ways of doing this:
   1. Right mouse click on Shipping\_Solution and select **Rebuild Solution**.
   2. Click on the BUILD menu item and select **Rebuild Solution**.
2. Now you need to open the Test Explorer. There are two ways (possibly more) of doing this: either click on **Test Explorer** the side bar on the upper left hand side of Visual Studio or use the TEST menu item. Both methods are shown in the following screen shot.



The Test Explorer will occupy all of the left hand side of Visual Studio. It should look something like the following screen shot.



1. To run the Scheduler\_GetNearestDate\_ExactMatch test method, right mouse click on it and then select **Run Selected Tests**.
2. If there is a problem with your test method then you might need to set break points in the code in order to debug it. After you have set the break points you would then right mouse click on the method and select **Debug Selected Tests**.

# Write More Unit Test Methods

In this section you will write additional Test Methods for other test cases.

1. Using the Scheduler\_GetNearestDate\_ExactMatch test method as a template, write unit test methods for the other test cases for the GetNearestDate method of the Scheduler class. There should be **four** more. With the one already written that makes five in total.
2. Write a unit test method to check that an exception of type System.NullReferenceException is thrown when the value of the voyagesList argument is null.

# Save Your Project

Save your project files on to your network drive. Alternatively use a memory stick or removable hard drive. You will need your project for the next lab exercise.