Lab 03

Test Automation

Hands on Lab



**Conditions and Terms of Use**

**Microsoft Confidential - For Internal Use Only**

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

**Copyright and Trademarks**

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
[*http*://www.microsoft.com/about/legal/permissions/](http://www.microsoft.com/about/legal/permissions/)

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

© 2016 Microsoft Corporation. All rights reserved.

Contents

[Introduction 4](#_Toc451510045)

[Exercise 1 – Adding Unit Tests and Measuring Code Coverage to the Build Process 4](#_Toc451510046)

[Exercise 2 – UI Test Automation](#_Toc451510049) 13

[Exercise 3 – Updating the build definition](#_Toc451510050) 16

# Introduction

**Duration: 60 minutes**

Goal: Introduction to Test Automation

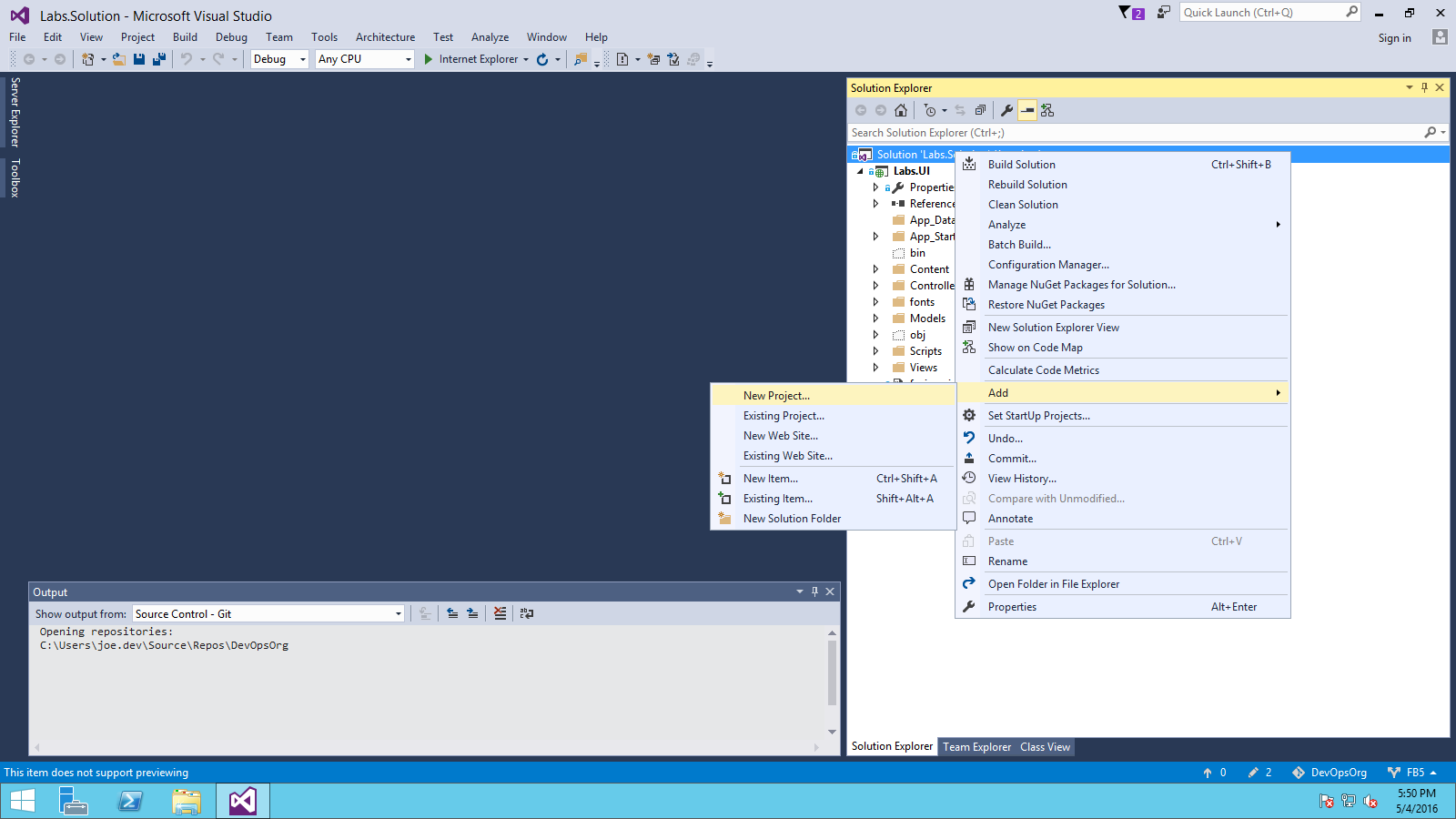
# Exercise 1 – Adding Unit Tests and Measuring Code Coverage to the Build Process

The current application committed into source control doesn’t any have kind of tests that can validate the requirements that were implemented actually met the business needs. In this first step, we will add few unit tests to the project, and make them part of the build process, allowing us to catch business logic issues before they go to the next stage of testing.

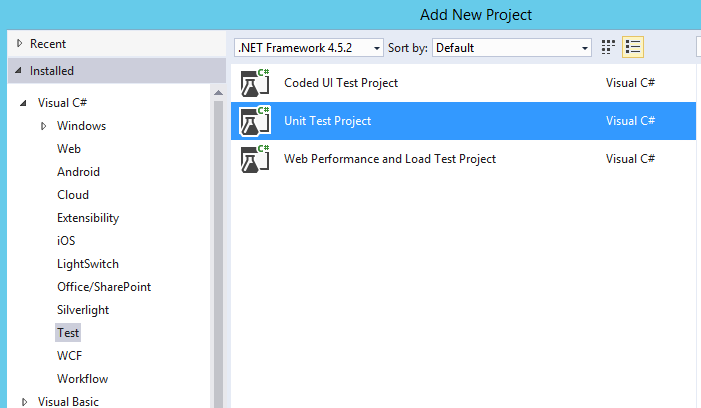
* Open your original MVC Labs.Solution from Lab 1 in Visual Studio.
* Go to your master branch.
* Undo any pending changes if you made any.
* Fetch, then pull from VSTS.
  + You can only pull if you either have no changes (undo them) or commit them locally first.
* Create a new feature branch called FB-UnitTest based off master and checkout (FB = feature branch, checkout means to swap to that branch).
* Make sure you’re working in the Feature branch, by checking the toolbar on the bottom right of Visual Studio.



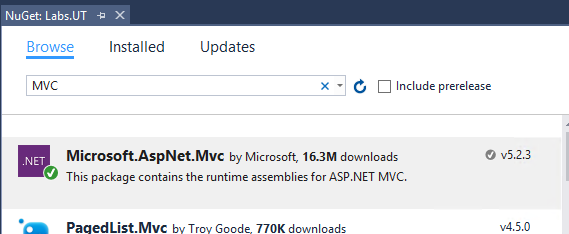
* Right click on the Labs.Solution node and select Add, then click new project:



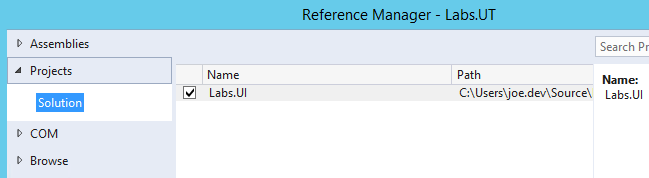
* Select Unit Test project under the Test section, name it **Labs.UT**



* Delete the UnitTest1.cs file.
* Add the Microsoft.AspNet.Mvc NuGet package to the unit test project:



* Add a project reference in the Unit Test project to the Labs.UI project



* Right click on the unit test project and add a unit test class and name it HomeControllerTests.cs
* Add a Test method using the following code

[TestMethod]

public void TestIndex\_ShouldReturnHomeView()

{

//AAA

//Arrange

string expectedViewName = "Index";

string returnedViewName;

HomeController controller = new HomeController();

//Act

ViewResult result = controller.Index() as ViewResult;

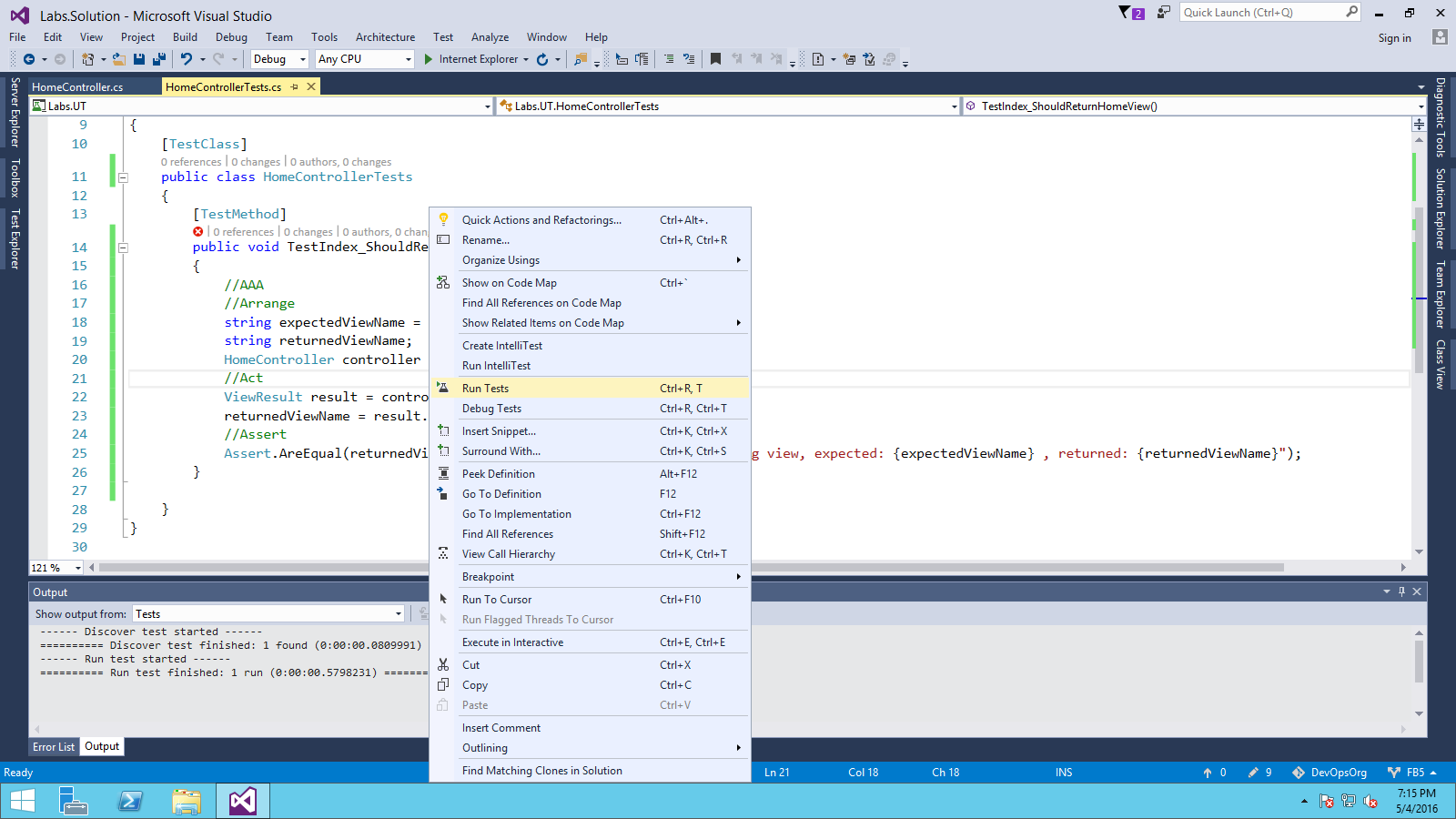
returnedViewName = result.ViewName;

//Assert

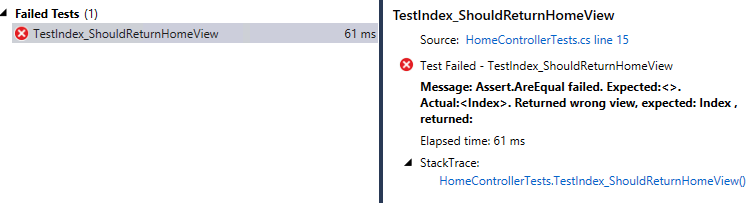
Assert.AreEqual(returnedViewName, expectedViewName, $"Returned wrong view, expected: {expectedViewName} , returned: {returnedViewName}");

}

* Build the solution
* Right click inside the unit test and code and click on “Run Tests”



* The test execution will fail showing you the following error:



* The reason is the controller is not returning the viewname, which is not necessarily an error, but it’s not a good practice. Let’s change that.
* Change the Index method in the controller to be:

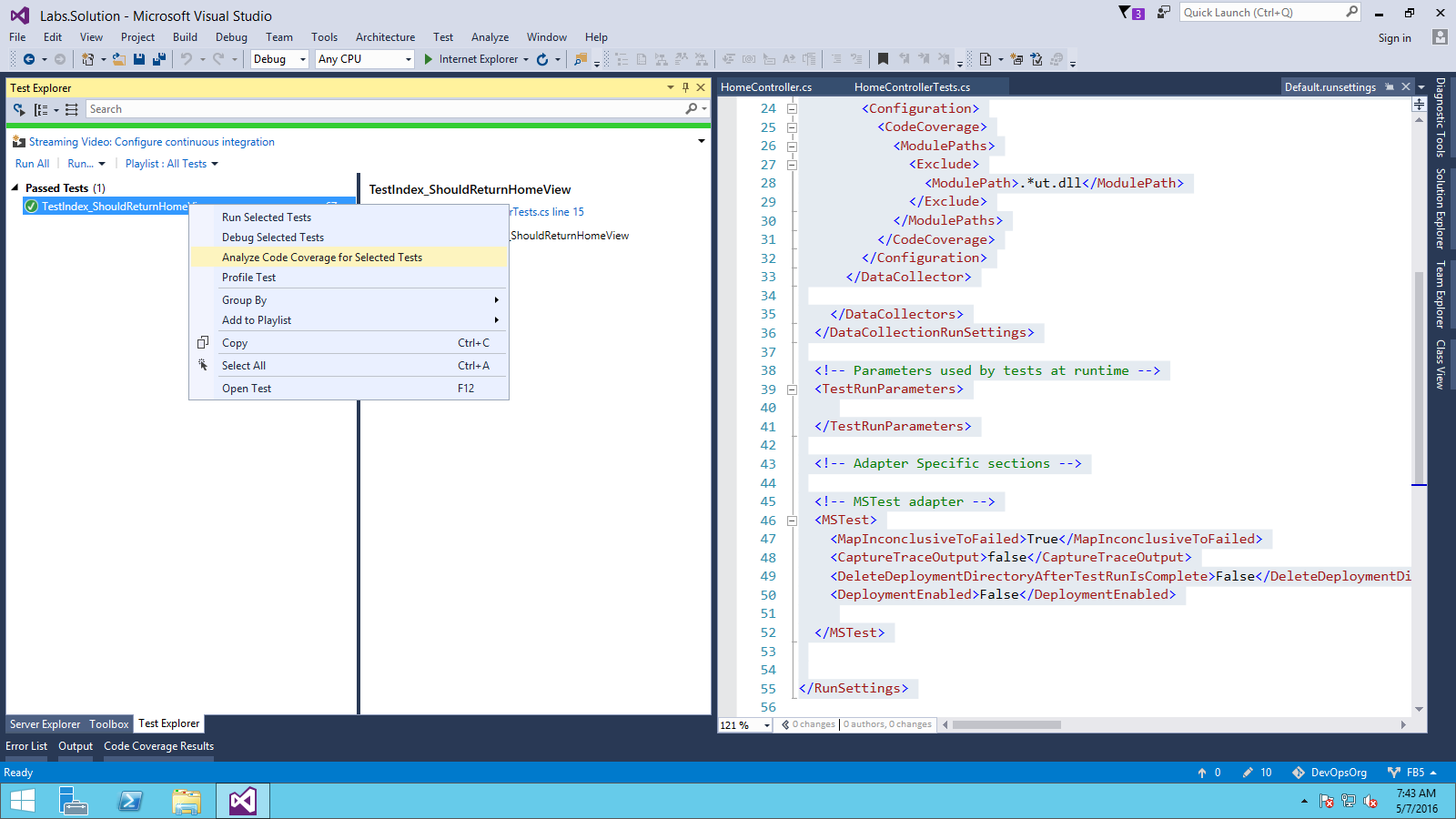
public ActionResult Index()

{

return View("Index");

}

* Build the project and re-run the test. Your test should pass.
* Right click on the Test Run in the Test Explorer window and click on Analyze Code coverage for Selected Tests.



* Notice how the code coverage results include the Unit test project, which is not desirable.
* Add an XML file to your Visual Studio solution and rename it to **Default.runsettings**. (The filename doesn’t matter, but the extension must be .runsettings.)
* Replace the content of that XML file with the following

<?xml version="1.0" encoding="utf-8"?>

<RunSettings>

<!-- Configurations that affect the Test Framework -->

<RunConfiguration>

<MaxCpuCount>1</MaxCpuCount>

<!-- Path relative to solution directory -->

<ResultsDirectory>.\TestResults</ResultsDirectory>

<!-- [x86] | x64

- You can also change it from menu Test, Test Settings, Default Processor Architecture -->

<TargetPlatform>x86</TargetPlatform>

<!-- Framework35 | [Framework40] | Framework45 -->

<TargetFrameworkVersion>Framework45</TargetFrameworkVersion>

<!-- Path to Test Adapters -->

<TestAdaptersPaths>%SystemDrive%\Temp\foo;%SystemDrive%\Temp\bar</TestAdaptersPaths>

</RunConfiguration>

<!-- Configurations for data collectors -->

<DataCollectionRunSettings>

<DataCollectors>

<DataCollector friendlyName="Code Coverage" uri="datacollector://Microsoft/CodeCoverage/2.0" assemblyQualifiedName="Microsoft.VisualStudio.Coverage.DynamicCoverageDataCollector, Microsoft.VisualStudio.TraceCollector, Version=11.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">

<Configuration>

<CodeCoverage>

<UseVerifiableInstrumentation>True</UseVerifiableInstrumentation>

<AllowLowIntegrityProcesses>True</AllowLowIntegrityProcesses>

<CollectFromChildProcesses>True</CollectFromChildProcesses>

<CollectAspDotNet>false</CollectAspDotNet>

<ModulePaths>

<Exclude>

<ModulePath>.\*ut.dll</ModulePath>

</Exclude>

</ModulePaths>

</CodeCoverage>

</Configuration>

</DataCollector>

</DataCollectors>

</DataCollectionRunSettings>

<!-- Parameters used by tests at runtime -->

<TestRunParameters>

</TestRunParameters>

<!-- Adapter Specific sections -->

<!-- MSTest adapter -->

<MSTest>

<MapInconclusiveToFailed>True</MapInconclusiveToFailed>

<CaptureTraceOutput>false</CaptureTraceOutput>

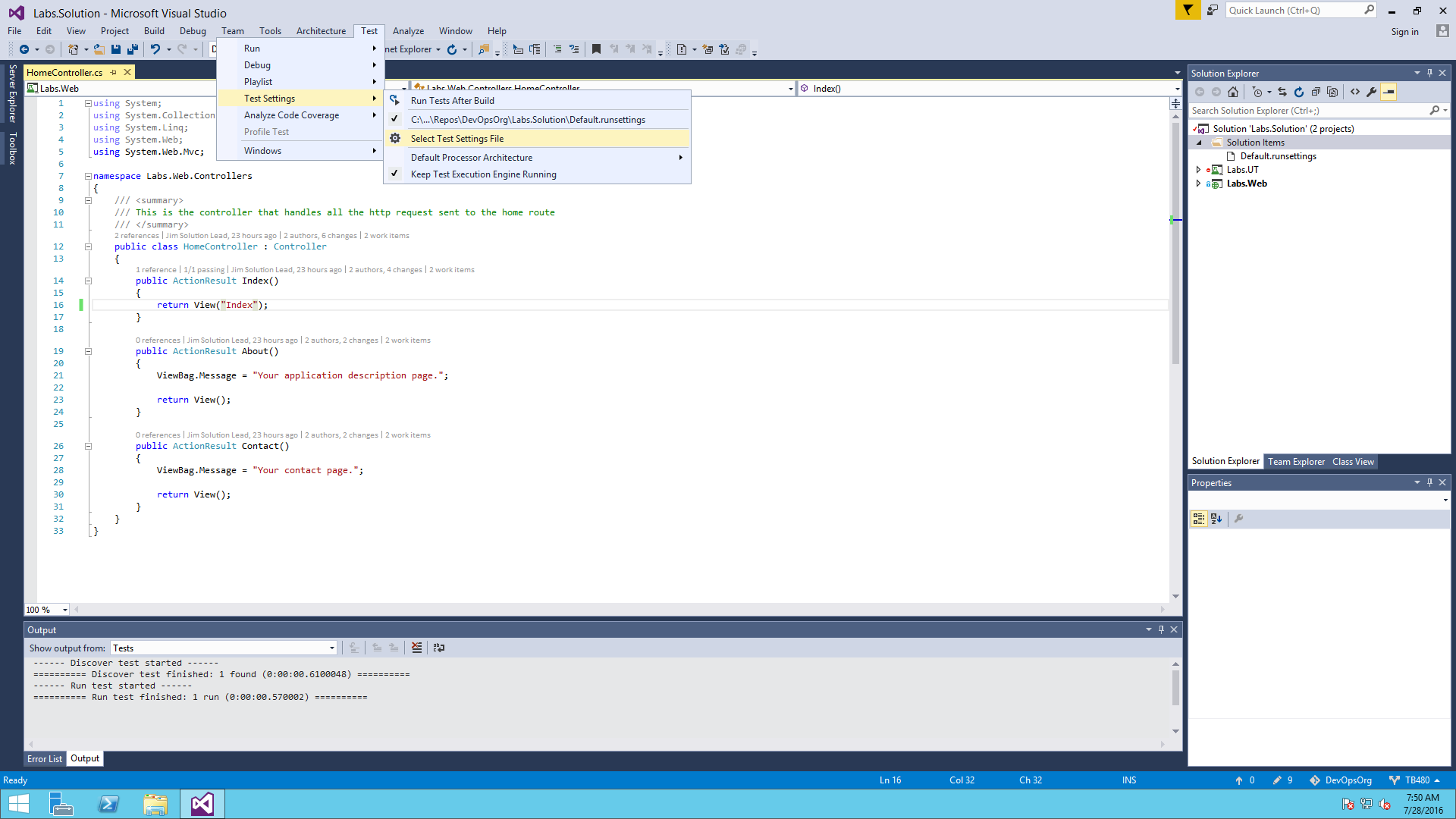
<DeleteDeploymentDirectoryAfterTestRunIsComplete>False</DeleteDeploymentDirectoryAfterTestRunIsComplete>

<DeploymentEnabled>False</DeploymentEnabled>

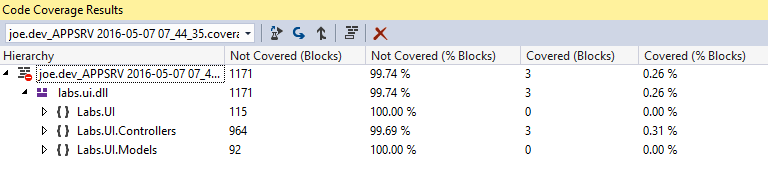
</MSTest>

</RunSettings>

**From the Test Menu at the top in visual studio, select the newly added file as the one that will be used to run the unit tests**. Notice how we excluded the Unit Test Project from the code coverage.

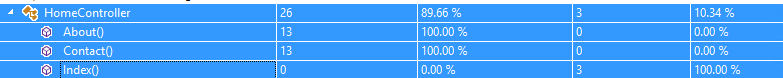


* Run the unit test once more and Analyze code coverage once more, you should get a result that looks as follow:

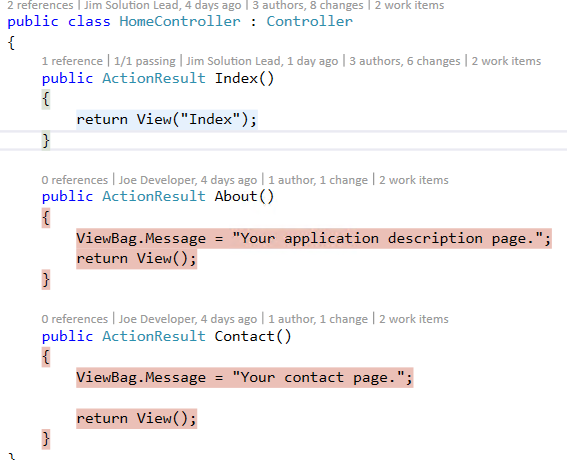


(You could potentially fine tune the **.RunSettings** file to focus on areas that should be tested. For example, we could exclude code that is auto generated by visual studio.)

* Click on the Home Controller class from the test coverage and notice how we see which methods are covered and which of them are not.

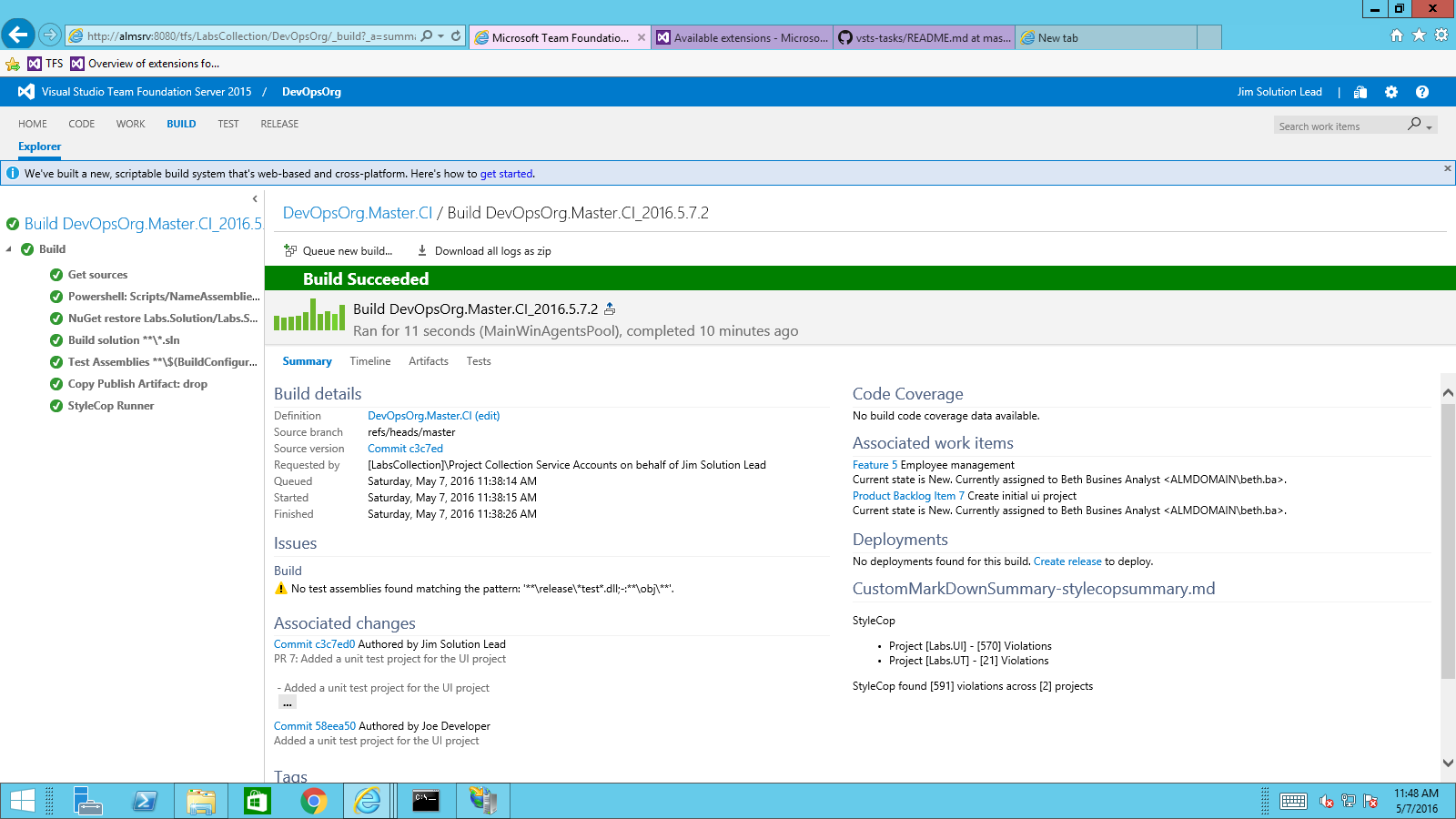


* After you’ve selected the 3 methods click on the button “**Show Code Coverage Coloring**”, the go back to the HomeController.cs class to notice how the parts of the code that are not covered by unit tests are highlighted in Red and the ones covered are highlighted in Blue:



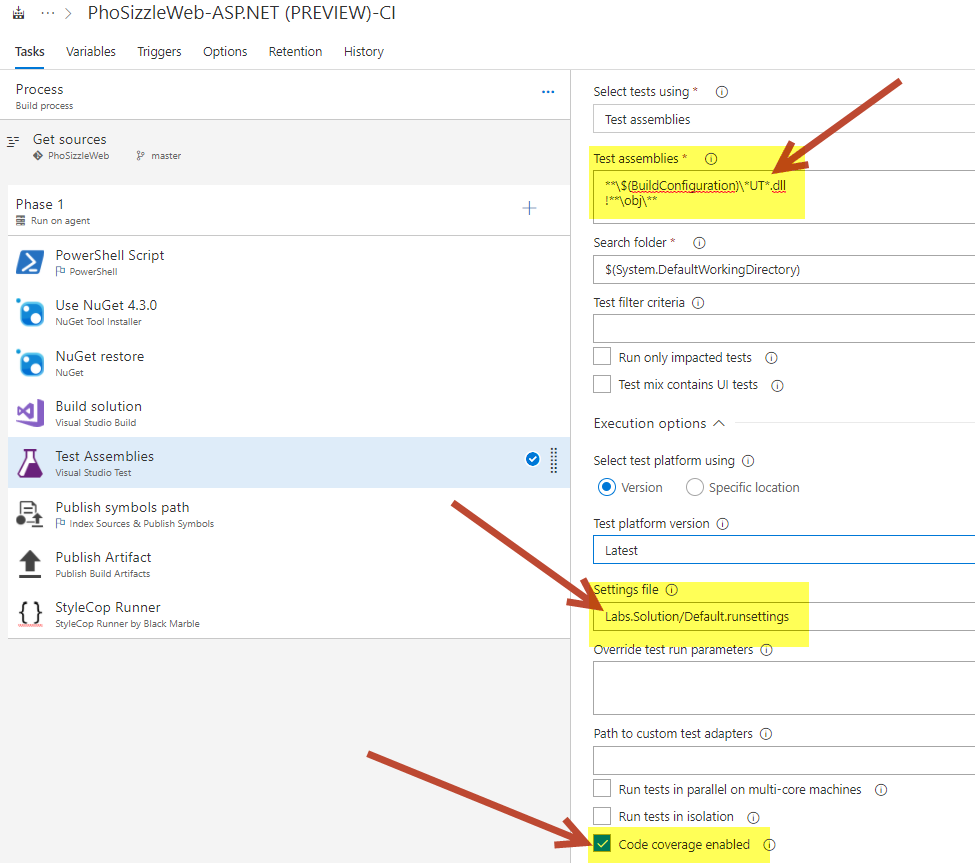
So far, our code coverage and unit tests are still local, we need to push them to the repository and start measuring code coverage as part of the build.

* Click on the Git Changes Indicator in the lower right part of the screen. Stage the changes, commit them and push them to the repository with the following message. “**Added a unit test project for the UI project**”
* Create a pull request in VSTS.
* Complete the pull request and merge it into master.
* A build will be automatically queued to merge the changes to the master branch.
* Open the build output summary, it will show the following results:

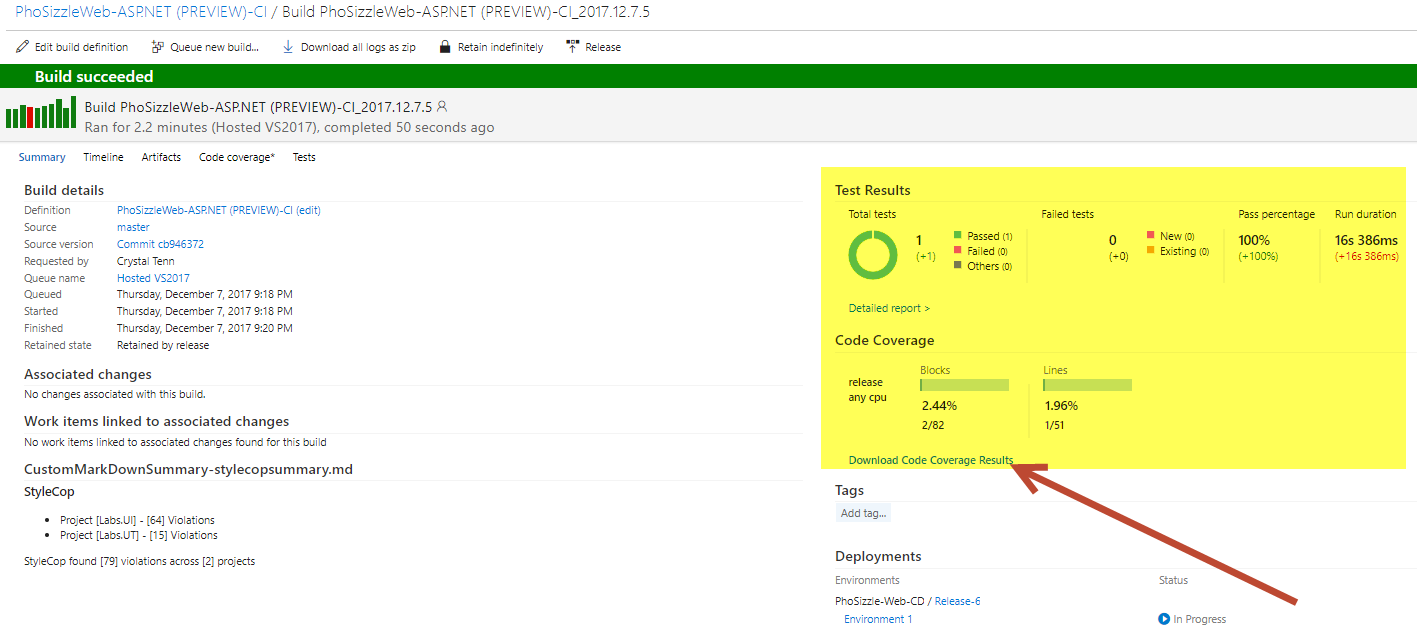


There is no code coverage and there isn’t any unit test being run. We need to modify the build definition to change that.

* Go to VSTS and the build definition and click edit.
* Select the Unit Test Task and modify the test assembly property to: \*\*\$(BuildConfiguration)\\*UT.dll;-:\*\*\obj\\*\*
* Change the settings file as well to points to the Default.Settings file.



* Queue another build and open the results once it has completed.
* It should show the following results:



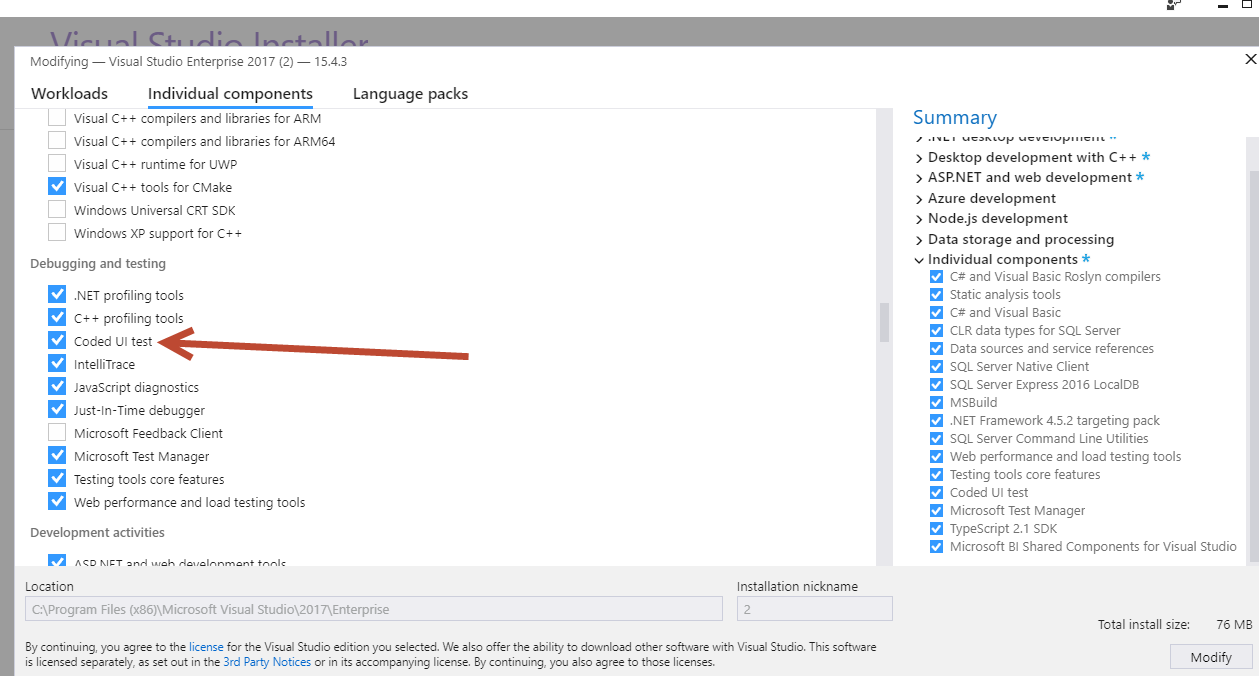
Notice how the test results and the code coverage results are displayed, but also how they can be downloaded locally for further analysis.

Adding unit tests to the project is a first step towards measuring the quality of the delivered products and making sure that the implemented features. During the next exercises we will cover other aspects of testing that can help a team gain the necessary confidence before they deliver their product to the end users.

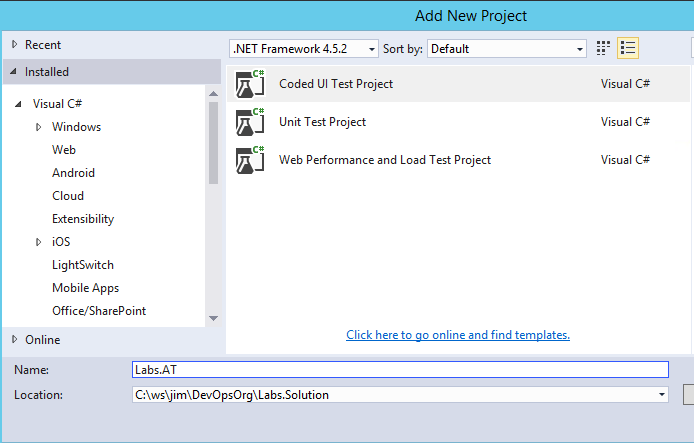
# Exercise 2 – UI Test Automation

In this exercise we will automate the execution of a test case. Note, this can only be done with Visual Studio Enterprise versions. Visual Studio Community and Professional do not come with Coded UI.

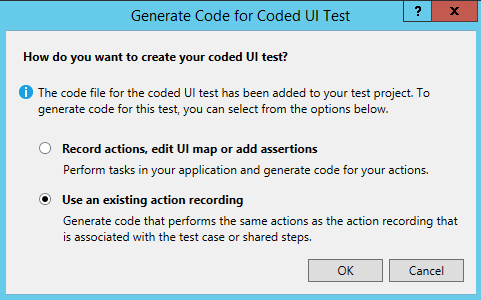
If you have Visual Studio 2017 Enterprise, you may need to activate the Coded UI and Performance Test modules. Please go to the Visual Studio Installer and verify they are installed before starting.



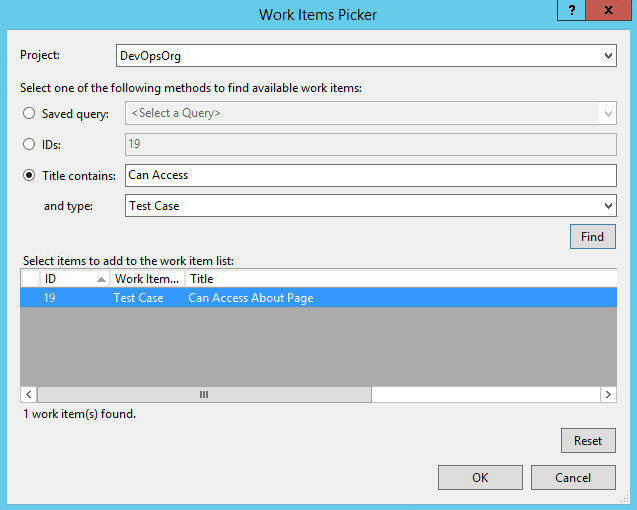
* Open your original MVC Labs.Solution from Lab 1 in Visual Studio.
* Go to your master branch.
* Undo any pending changes if you made any.
* Fetch, then pull from VSTS.
  + You can only pull if you either have no changes (undo them) or commit them locally first.
* Create a new feature branch called FB-AutomatedTest based off master and checkout (FB = feature branch, checkout means to swap to that branch).
* Make sure you’re working in the Feature branch, by checking the toolbar on the bottom right of Visual Studio.
* Right click on the **Labs.Solution** node, add a new **CodedUI** project and name is **Labs.AT**:



* You will be given a choice to either record an automated UI test or use an existing one. Since our goal is to automate an existing manual test we will use an existing recording:



* Use the search UI to find the test case you created in the previous step. Either use the ID, or enter a partial text from the test case:



* Once you find the test case, select it and click “Ok”. This step will generate code from the test case execution you have generated from running the test using Microsoft Test Manager.
* Rename the test class to Sprint1SuiteAutomationTest.cs
* Rename the test method to CUT\_TestAboutPageAccess
* The test method should be as follow:

[TestMethod]

public void CUT\_TestAboutPageAccess()

{

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

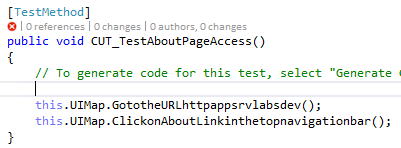
this.UIMap.GototheURLhttpappsrvlabsdev();

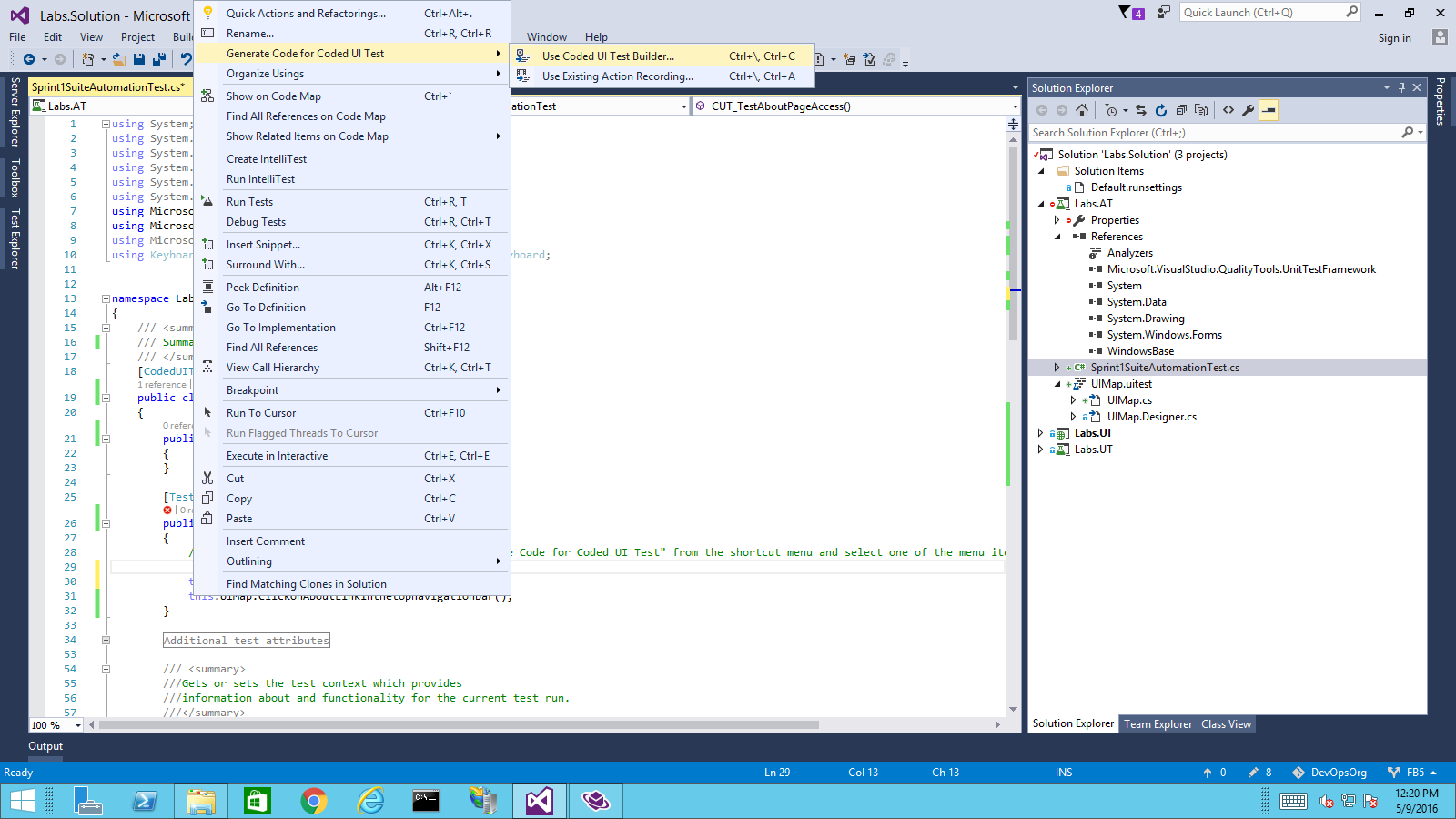
this.UIMap.ClickonAboutLinkinthetopnavigationbar();

}

* Execute the test, and **make sure you don’t move the mouse while the test is running**.
* Open the Test Explorer window and notice that the test has failed. The reason it failed is because when the test was recorded the browser windows was already opened.

We will use the recording feature now to add the missing part.

* Make sure that all the browser windows are closed.
* Insert a blank line at the start of the test method:
* Position the cursor at that blank line.
* Right click to show the action menu, click on “Generate Code for Coded UI Test” then Click on “Use Coded UI Test Builder”.



* The recorder will launch, but the recording hasn’t started yet. Click on the button that initiates the recording.  
  
* Click on the browser window in the task bar.
* By default the browser opens and goes to a different web page. We will change the page it goes to by modifying the generated code later.
* Click on the Generate code button



* Name the method OpenBrowserAndGoToTestHomePage, then click on Add and Generate.
* Close the Action Recorder.
* The code should now be as shown below. Delete the second line:

[TestMethod]

public void CUT\_TestAboutPageAccess()

{

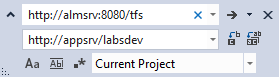
this.UIMap.OpenBrowserAndGoToTestHomePage();

~~this.UIMap.GototheURLhttpappsrvlabsdev();~~

this.UIMap.ClickonAboutLinkinthetopnavigationbar();

}

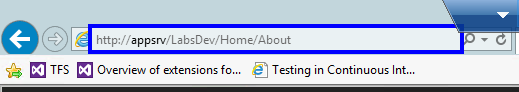
* Right click on the **OpenBrowserAndGoToTestHomePage** method and click on “Go to Defintion” (or use the F12 key).
* Close any open browser window and run the test once more. (Make sure you don’t use the mouse while the test is running)
* Right click on the URL variable and click on “Go to Defintion”
* Change the value to <http://appsrv/labsdev>
* Change it in the comment section as well. You can use the Find and Replace menu in visual studio:



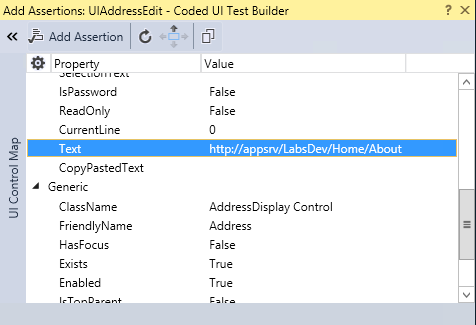
* Close any open browser window and re-run the test.
* Open the Test explorer window and notice that the test now passes.
* While the test is passes, it doesn’t any validation.
* Insert a blank line at the very end of the test method.
* Initiate the Action recorder once more.
* Navigate to the about page, by opening the home page URL then clicking on the About link. then click on the assertion icon:



* While clicking on the mouse left button, target the browser URL and notice how the select URL is selected by a blue rectangle:

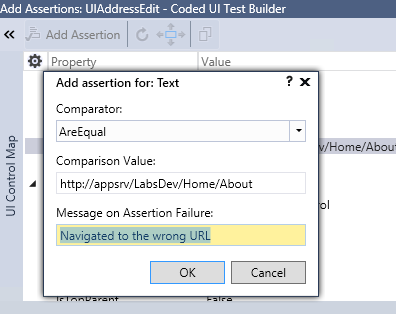


* Use the Assertion window to validate the URL text:



* Click “Add Assertion”.
* Enter the following message in the Assertion failure:

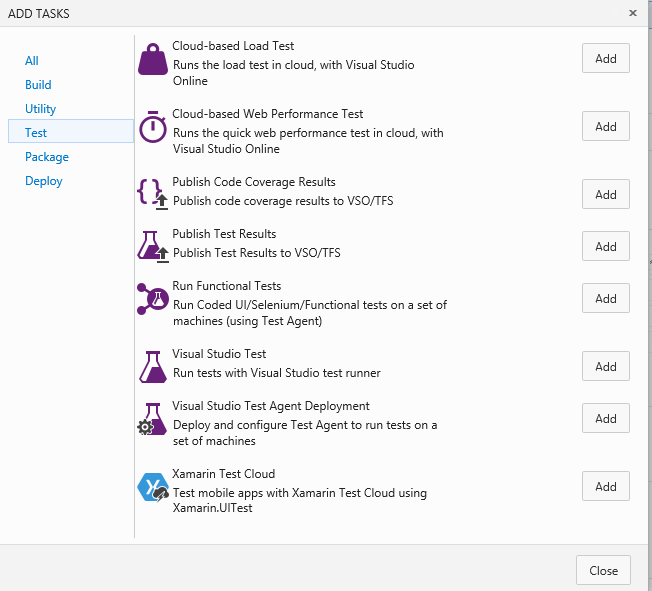
“Navigated to the wrong URL”



* Click on the Generate Code icon. Name the method “ValidateNavigationToAboutPage”
* Close the Action recorder.
* Close any browser window and re-run the test.
* Open the test explorer and make sure that the test passes.
* Stage, Commit and Push the changes to the server with the following message: ‘Added a UI Test automation project’
* Create a Pull request to merge into the master branch.
* Login from the second virtual machine using:
  + Username: almdomain\Administrator
  + Password: Workshop!1
* Approve and Complete the pull request.

# Exercise 3 – Updating the build definition

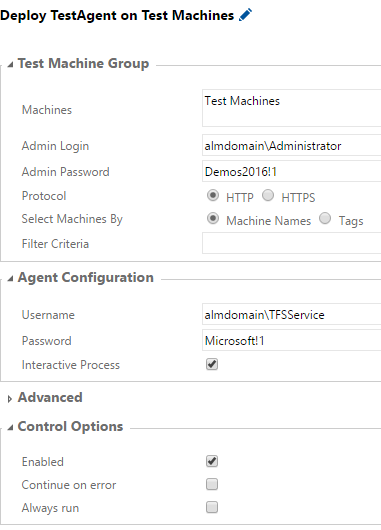
* **Log off from the AppSrv machine and log in to the TFS virtual machine.**
* Edit the DevOps.Master.CI build definition and add the two following steps, in the following order



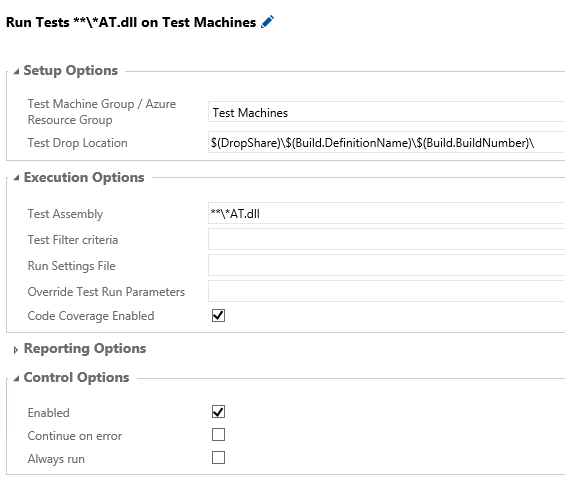
1

2

* Use the following settings for **Visual Studio Test Agent Deployment** Task:



* Use the following settings for **Run Functional Tests** Task:



* Modify the “Copy and Publish Build Artifacts”:
  + In the content section enter the following (each in a separate line)  
    \*\*\Labs.UI

\*\*\Labs.UT

\*\*\Labs.AT

* **Save the build definition and queue a new build**. (Make sure that you have logged off from the AppSrv machine because it will be used for the automated UI test).
* Open the build summary and notice how both tests are being used to automatically validate the code that gets merged into the master branch.

