**Lab**

# Test Driven Development

Contents

[Test Driven Development 1](#_Toc355859445)

[Lab Introduction 3](#_Toc355859446)

[Test Driven Development Approach 4](#_Toc355859447)

[Task 1 – Create Basic Class and Returns 4](#_Toc355859448)

[Task 2 – Create Failing Tests 5](#_Toc355859449)

[Task 3 – Make Failing Tests Pass 6](#_Toc355859450)

[Task 4 – Refactor with Confidence 8](#_Toc355859451)

[Responding to Changes 11](#_Toc355859452)

[Task 5 – Modify Code Scaffolding 11](#_Toc355859453)

[Task 6 – Create New Unit Tests 12](#_Toc355859454)

[Task 7 – Modify Code and Refactor 13](#_Toc355859455)

# Lab Introduction

In this lab you will learn the basics of doing Test Driven Development. This lab assumes you are familiar with unit testing concepts using the VS 2013 testing tools. It also assumes you have a level of competency writing basic C# code.

**Objectives**

In this Hands-On Lab, you will learn:

* Create Tests First
* Create Code after each Test
* Red to Green to Refactor
* Create Tests first for change requests

**Duration**

Suggested time to complete this Hands-On Lab is approximately 30 minutes.

**Setup – Starting Materials**

This Hands-On Lab includes the starting materials that are available under the root courseware folder on your machine. Open the folder **C:\Unit Testing\Labs\6\_TDD**

**Note**: This lab requires Visual Studio 2012

# Test Driven Development Approach

You have received the following requirements from the business:

|  |
| --- |
| **Requirement**  The user needs to send a pants color to the system and in return that system will suggest a shirt color. The following rules need to apply:   * If the pants color is red then the recommended shirt color must be pink * If the pants color is blue then the recommended shirt color must be green * If the pants color is green then the recommended shirt color must be blue * If the pants color is pink then the recommended shirt color must be yellow * If the pants color is black then the recommended shirt color must be white * Any other pants color then the recommended shirt color must be white * If the pants color is empty or white then the system should return an error |

## 

## Task 1 – Create Basic Class and Returns

1. Open Visual Studio 2012
2. Open solution file in the Root > Begin folder. You should see two projects in the solution, TestDrivenDevelopment.Code & TestDriveDevelopment.UnitTests
3. In the TestDrivenDevelopment.Code project, open the ColorCordinator.cs file. You will notice that only the class name exists but no other functionality. Based on the requirements you will need to add the basic method and property of the class. Change the class file so it looks like the following:

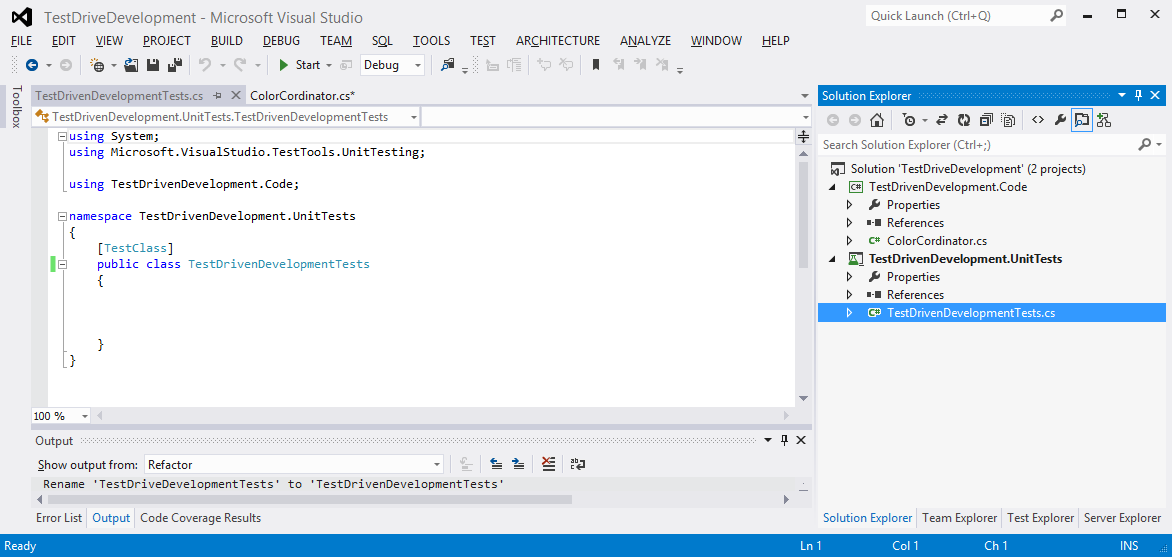
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace TestDrivenDevelopment.Code  {  public class ColorCordinator  {  public string shirtColor { get; set; }    public void RunColorCordinatorForPants(string pantsColor)  {  }  }  } |

You now have the basic scaffolding of your business class. The ExecuteColorCordinatorForPants method will contain the logic to execute the business requirements and the shirtColor property will be used to return the results.

## Task 2 – Create Failing Tests

Now that we have our basic class, method, and property created we start creating our unit tests to meet the business requirements.

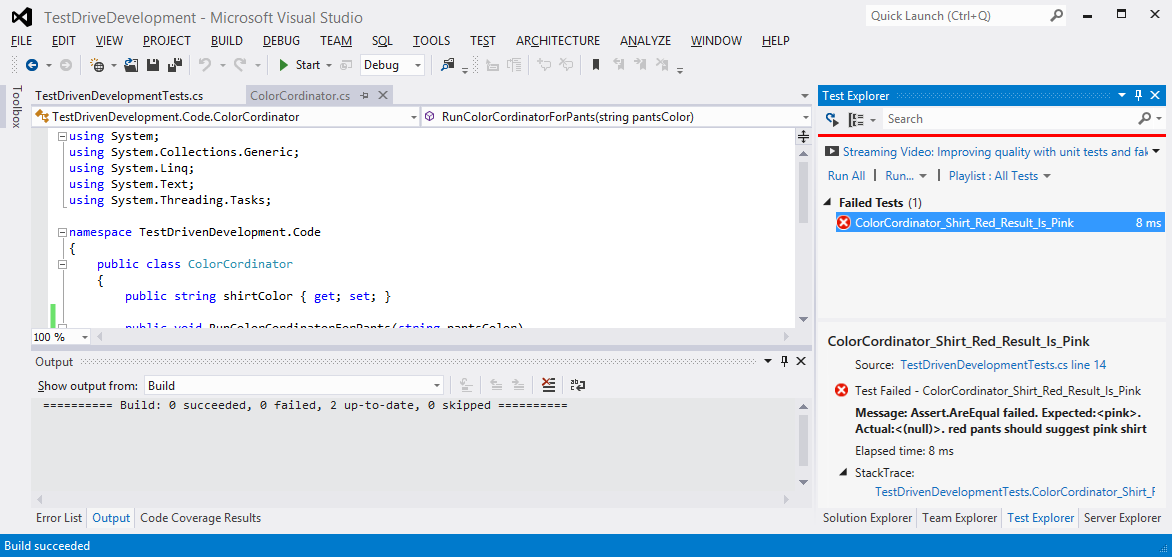
1. From the TestDriveDevelopment.UnitTests project open the TestDrivenDevelopmentTests class file. You will notice that the only code in this file in the namespace and class name.



1. Our first requirement states that “If the pants color is red then the recommended shirt color must be pink”. So we will create a test method to test for this first requirement. You can use the following code for the first method.

|  |
| --- |
| [TestMethod]  public void ColorCordinator\_Shirt\_Red\_Result\_Is\_Pink()  {  //Arrange  ColorCordinator objColorCordinator = new ColorCordinator();  //Act  objColorCordinator.RunColorCordinatorForPants("red");  var result = objColorCordinator.shirtColor;    //Assert  Assert.AreEqual("pink", result, "red pants should suggest pink shirt");  } |

1. Save the files and build your solution.
2. Open the Test Explorer and Run the test.



Obviously the test is going to fail because the RunColorCordinatorForPants method does not contain any logic or return any values.

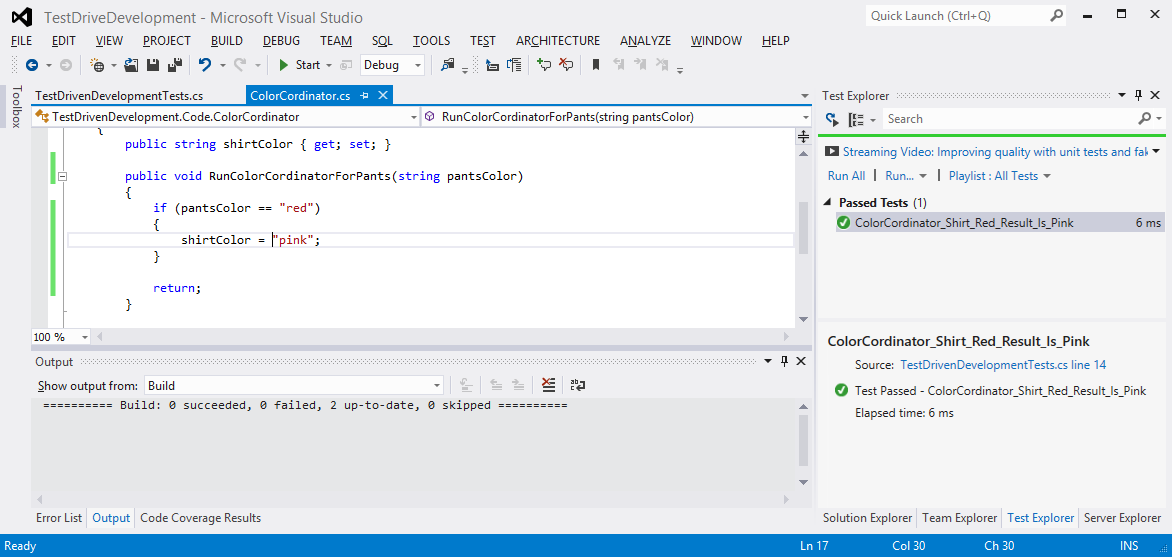
## Task 3 – Make Failing Tests Pass

Let us add the code logic necessary into our method so that the requirement is met and the test passes.

1. Open the ColorCordinator.cs file from the TestDrivenDevelopment.Code project.
2. Add the following code so that the method will pass the test. The method should look like this:

|  |
| --- |
| public void RunColorCordinatorForPants(string pantsColor)  {  if (pantsColor == "red")  {  shirtColor = "pink";  }  return;  } |

1. Save and rebuild the solution.
2. Open the Test Explorer and Run the test.



Our test successfully passed. Obviously this may not be the best way to code this method as we have to account for many different pant color requirements. This will get addressed during the refactor phase of TDD once all the required tests have passed.

1. Spend some time to complete the Test to Code process for the rest of the requirements. You should end up with a total of 7 tests.

|  |
| --- |
| **Requirement**  The user needs to send a pants color to the system and in return that system will suggest a shirt color. The following rules need to apply:   * If the pants color is red then the recommended shirt color must be pink * If the pants color is blue then the recommended shirt color must be green * If the pants color is green then the recommended shirt color must be blue * If the pants color is pink then the recommended shirt color must be yellow * If the pants color is black then the recommended shirt color must be white * Any other pants color then the recommended shirt color must be white * If the pants color is empty or white then the system should return an error |

## 

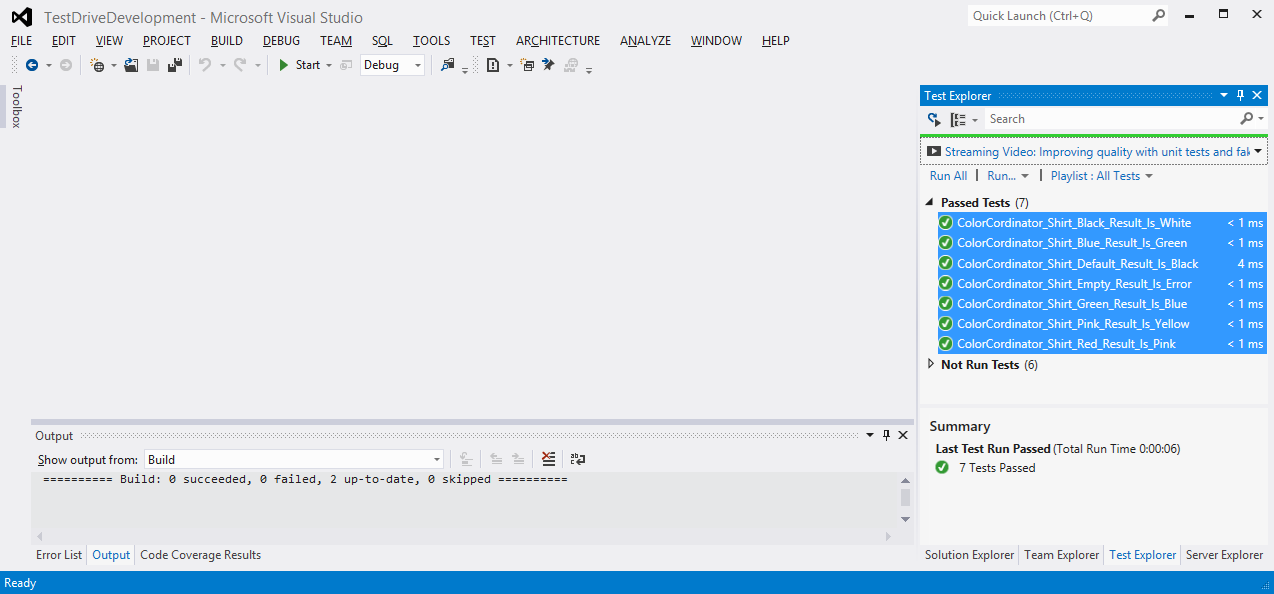
## Task 4 – Refactor with Confidence

Now that we have all our tests and code completed for our requirement we can safely refactor our code to make it more efficient. In this case we are going to remove the multiple if statements with a switch statement.

1. Open the ColorCordinator.cs file from the TestDrivenDevelopment.Code project.
2. Refactor the code to something like this:

|  |
| --- |
| public void RunColorCordinatorForPants(string pantsColor)  {  string \_shirtcolor = "";  //check for empty or white and throw and error if true  if (pantsColor == "" || pantsColor == "white")  {  \_shirtcolor = "error";  this.shirtColor = \_shirtcolor;  return;  }  //sort thru pants colors and recommend the shirt color accordingly  switch (pantsColor)  {  case "red":  \_shirtcolor = "pink";  break;  case "blue":  \_shirtcolor = "green";  break;  case "green":  \_shirtcolor = "blue";  break;    case "pink":  \_shirtcolor = "yellow";  break;    case "black":  \_shirtcolor = "white";  break;  default:  \_shirtcolor = "black";  break;  }  //set the global property so it can be returned on the object  this.shirtColor = \_shirtcolor;  return;  }  } |

1. Save and rebuild the solution.
2. Open the Test Explorer and Run the test.



This should have resulting in passing all 7 of your tests. The TDD process allowed us to refactor our code in confidence as we now have a way to check our code changes against a group of unit tests to ensure the code is meeting the requirements.

# Responding to Changes

Our initial requirements are completed and our code is now in production. The feature is successful and the business has decided to improve the functionality and has added a change request. Below is the original requirement with the modified functionality added to account for recommending a shoe color(s).

|  |
| --- |
| **Change**  The user needs to send a pants color to the system and in return that system will suggest a shirt color and shoe color. The following rules need to apply:   * If the pants color is red then   + the recommended shirt color must be pink   + the recommended shoe colors are black and brown * If the pants color is blue then   + the recommended shirt color must be green   + the recommended shoe colors are black and white * If the pants color is green then   + the recommended shirt color must be blue   + the recommended shoe colors are black * If the pants color is pink then   + the recommended shirt color must be yellow   + the recommended shoe colors are black * If the pants color is black then   + the recommended shirt color must be white   + the recommended shoe colors are black * Any other pants color then   + the recommended shirt color must be white   + the recommended shoe colors are black * If the pants color is empty or white then the system should return an error |

## Task 5 – Modify Code Scaffolding

Because our ColorCordinator class is not returning any show color recommendations we need to make some minor modifications to the code in order to create new tests.

1. Open the ColorCordinator.cs file from the TestDrivenDevelopment.Code project.
2. Add a ListShoes property to our class.

|  |
| --- |
| public string shirtColor { get; set; }  public IList<string> ListShoes { get; set; }  *(the rest of the code…)* |

We now have a property that will be used to pass back the shoes. This is enough to create new unit tests.

## Task 6 – Create New Unit Tests

We have a new property to create unit tests against. The ListShoes property will return a list of strings as the recommended shoe color can have more than one value.

1. Open the TestDrivenDevelopmentTests.cs file from the TestDrivenDevelopment.UnitTests project.
2. Create a new test method that will be used to check to make sure the ListShoes value meets the requirements when the RunColorCordinatorForPants is called. In this test method we will check the shoe colors for the pink pants.

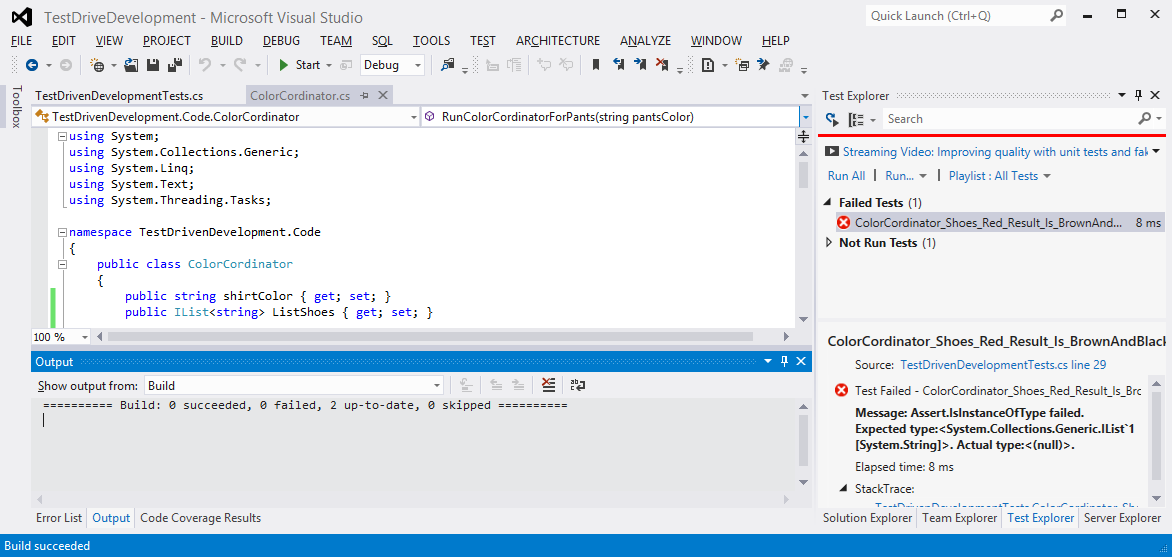
|  |
| --- |
| [TestMethod]  public void ColorCordinator\_Shoes\_Red\_Result\_Is\_BrownAndBlack()  {  //Arrange  ColorCordinator objColorCordinator = new ColorCordinator();  //Act  objColorCordinator.RunColorCordinatorForPants("red");  var result = objColorCordinator.ListShoes;  //Assert  Assert.IsInstanceOfType(result, typeof(IList<string>));  Assert.IsTrue(result.Contains("black"), "shoe color must contain black");  Assert.IsTrue(result.Contains("brown"), "shoe color must contain brown");  Assert.IsTrue((result.Count == 2), "only 2 results should be returned");  objColorCordinator = null;  } |

This method will check the ListShoes property separately than the shirtColor property. This assumes that our other shirtColor tests are still valid.

Our assertions in the test method are checking for 3 things.

* Is the ListShoes an IList<string> type
* Do the returned results contain black and brown
* The result count is equal to 2

1. Save the files and build your solution.
2. Open the Test Explorer and Run the test.



The test failed as expected because no logic is in our code to return a shoe color. We will now need to go thru the TDD process of modifying code to get the tests to work and refactor as appropriate.

## Task 7 – Modify Code and Refactor

We now need to modify our code to include the proper logic to recommend a shoe color along with the shirt color. In this task we will leave it to you to modify the code as you see fit. However please make sure you follow the following process.

1. Develop your code so that the test passes.
2. Continue the cycle of creating tests and modifying code to make those tests pass until all requirements are met.
3. Refactor your code and ensure all tests past after the code refactor.

|  |
| --- |
| You can see an example of the finished code and tests in the **6\_TDD\End** folder |