Test Driven Development (TDD) Using MVC Web Application

In this article we will create test methods and then will write business code against test methods in MVC 5.

Bipin Patil Mar 16 2016

2 0 33.1k

TDDWithMVC.zip

Introduction

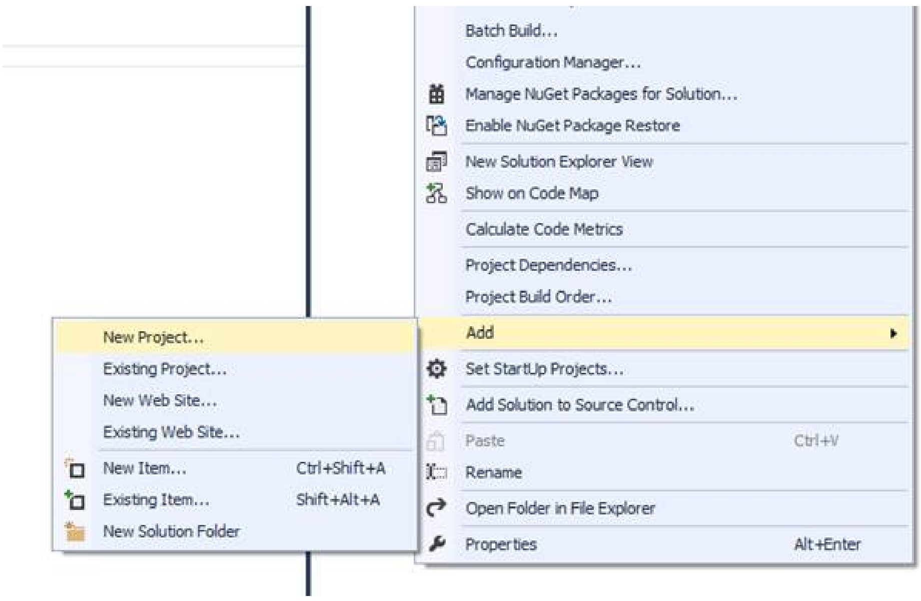
Here we going to calculate Rating for Shop using various methods. For this we may need to try different methods over time to have validation against the test method.

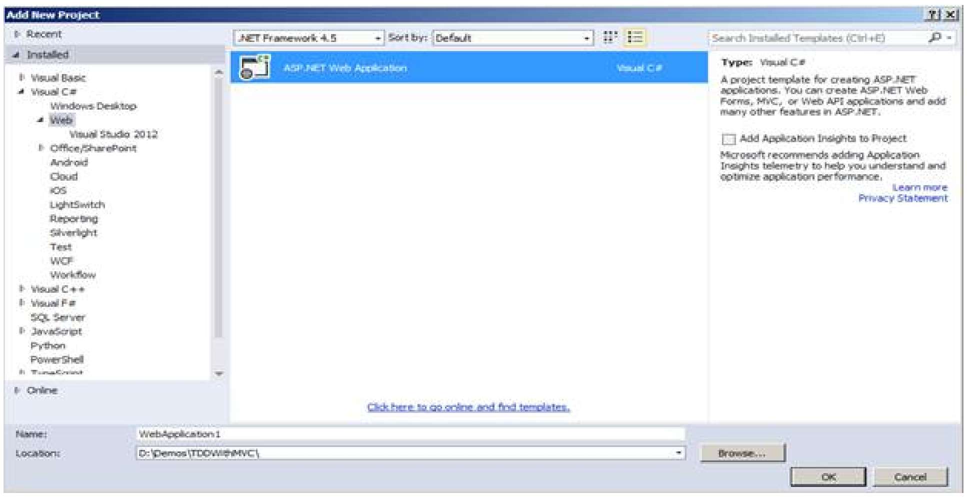
Let's start step by step:

- Create ASP.NET Test Project - Test Project "*TDDWithMVC.Tests*".
- Adding / testing default test methods.
- Adding a Test class.
- Working with Test class and test methods.
- Adding Features classes.
- Calling the Features classes and test unit test methods.
- Refactoring Unit test and business class code.
- Also how to use Strategy Design Pattern.

Create the project

Right click on Solution and add a new project.





.NET Framework 4.5 Sort by: Default

ASP.NET Web Application Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Add Application Insights to Project

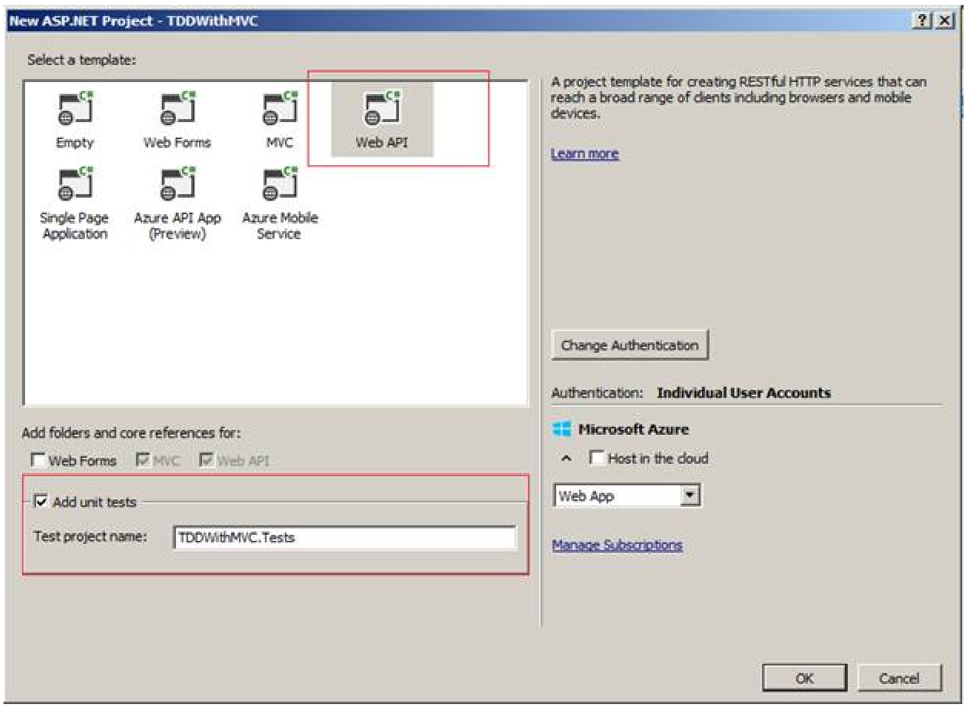
Microsoft recommends adding Application Insights telemetry to help you understand and optimize application performance.

Click here to go online and find templates.

Name: WebApplication1

Location: D:\Demos\TDDWithMVC\

Select a template Web API.



A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Authentication: **Individual User Accounts**

Add folders and core references for:

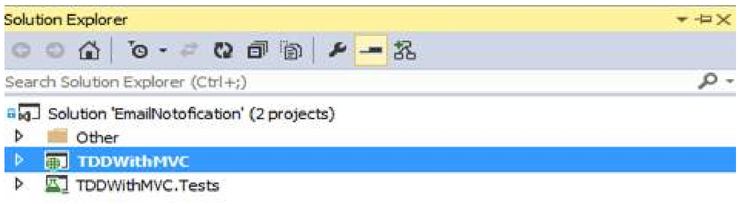
Web Forms [✓ MVC] [✓ Web API]

[✓ Add unit tests]

Test project name: TDDWithMVC.Tests

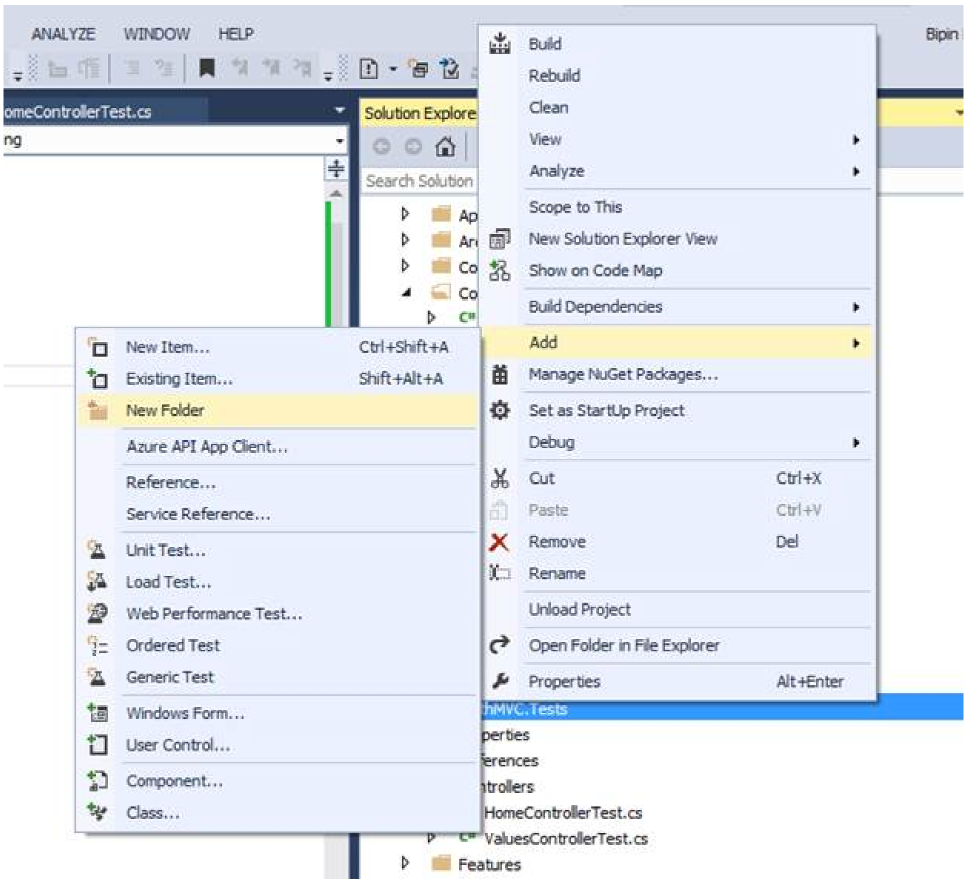
Host in the cloud

Web App

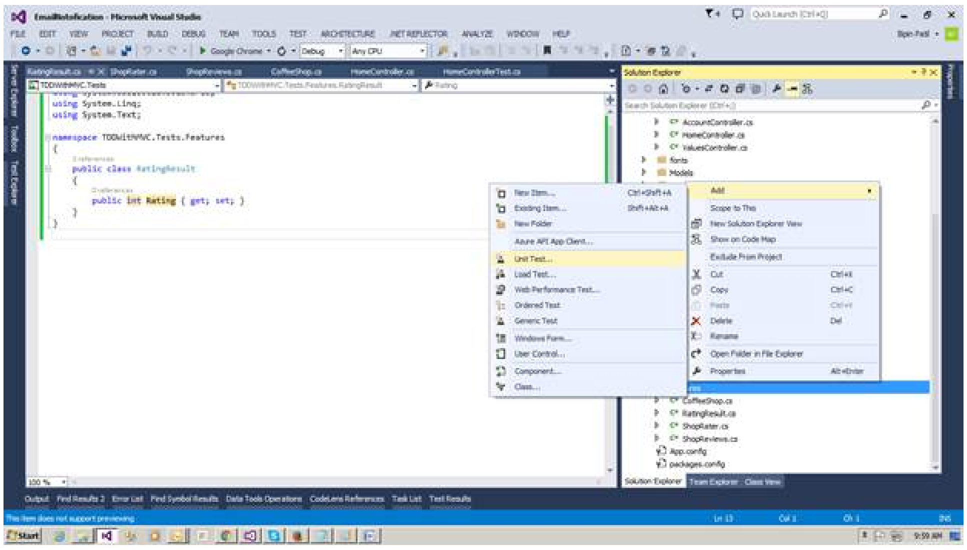


Solution 'EmailNotofication' (2 projects)
Other
TDDWithMVC
TDDWithMVC.Tests

Add new folder Name as “Features”.



Now add a Test Class to Test Project “TDDWithMVC.Tests” Names as “UnitTest1.cs”

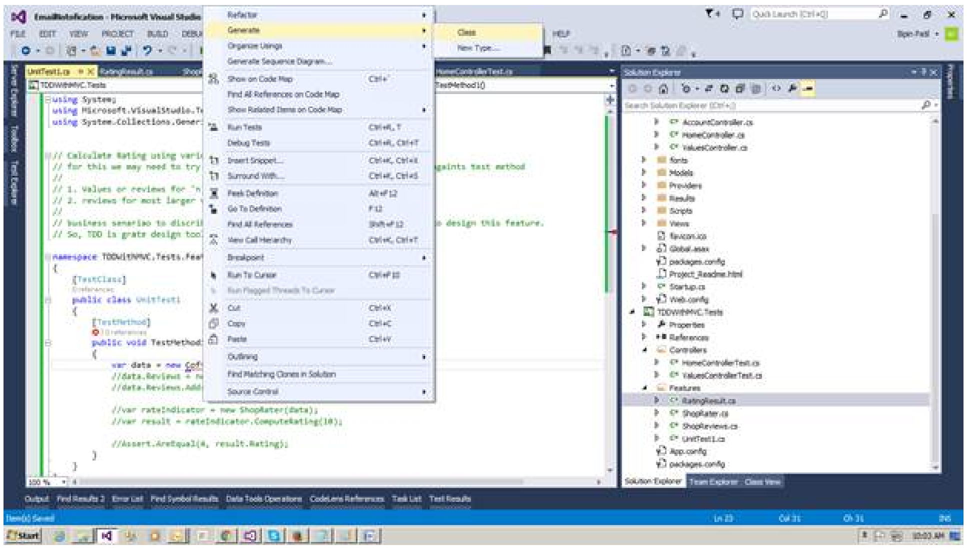

```
using System.Linq;
using System.Text;

namespace TDDWithMVC.Tests.Features
{
    public class RatingResult
    {
        public int Rating { get; set; }
    }
}
```

Replace code with following code

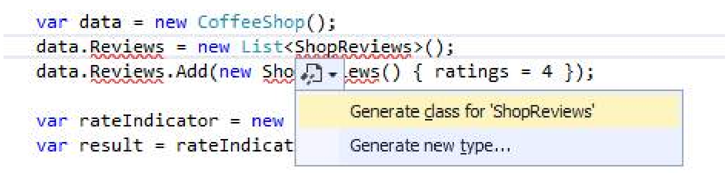
```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;

// Calculate Rating using various methods
// for this we may need to try diffrent methods over time to have validation againts tes
//
// 1. Values or reviews for 'n' number of values
// 2. Reviews for Weighted Rating
// 3. Reviews for top n no of reviews
//
// business senariao to discribe approch when to use TDD : when we do not know how to de

namespace TDDWithMVC.Tests.Features
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
            var data = new CoffeeShop();
            data.Reviews = new List < ShopReviews > ();
            data.Reviews.Add(new ShopReviews()
            {
                ratings = 4
            });

            var rateIndicator = new ShopRater(data);
            var result = rateIndicator.ComputeRating(10);

            Assert.AreEqual(4, result.Rating);
        }
    }
}
```

Now we need to create classes and methods which we are using in Test class. Right now we are considering Namespace for classes,

1. Right click or Ctrl + . on CoffeeShop and add new class,

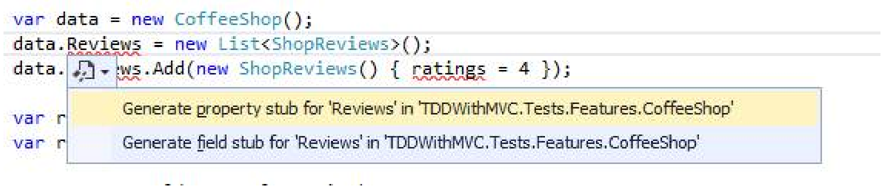


2. Add class ShopReviews as below,



```
var data = new CoffeeShop();
data.Reviews = new List<ShopReviews>();
data.Reviews.Add(new Sho    ews() { ratings = 4 });

var rateIndicator = new
var result = rateIndicat
```

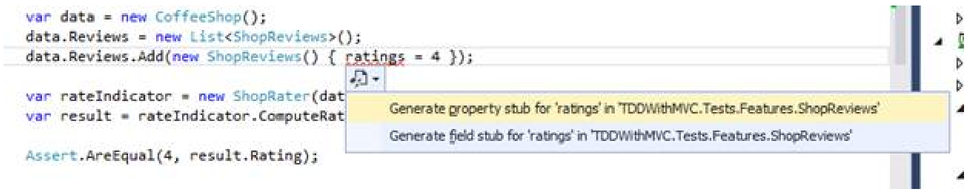
3. Generate Property as below,



```
var data = new CoffeeShop();
data.Reviews = new List<ShopReviews>();
data.   ws.Add(new ShopReviews() { ratings = 4 });

var r
var r
```

Generate property stub for 'Reviews' in 'TDDWithMVC.Tests.Features.CoffeeShop'
Generate field stub for 'Reviews' in 'TDDWithMVC.Tests.Features.CoffeeShop'

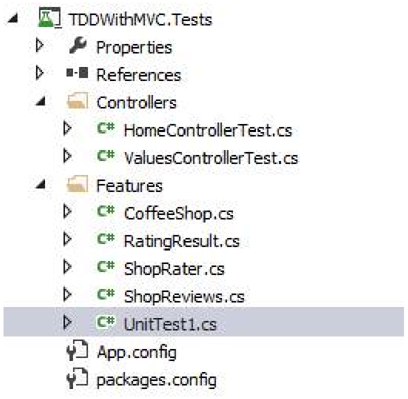
4. Add property for Rating,



```
var data = new CoffeeShop();
data.Reviews = new List<ShopReviews>();
data.Reviews.Add(new ShopReviews() { ratings = 4 });

var rateIndicator = new ShopRater(dat
var result = rateIndicator.ComputeRat

Assert.AreEqual(4, result.Rating);
```

Generate property stub for 'ratings' in 'TDDWithMVC.Tests.Features.ShopReviews'
Generate field stub for 'ratings' in 'TDDWithMVC.Tests.Features.ShopReviews'

5. In the same way add class ShopRater and add new method as ComputeRating. Now we can find the following classes added to Features folder,



```
TDDWithMVC.Tests
    Properties
    References
    Controllers
        HomeControllerTest.cs
        ValuesControllerTest.cs
    Features
        CoffeeShop.cs
        RatingResult.cs
        ShopRater.cs
        ShopReviews.cs
        UnitTest1.cs
    App.config
    packages.config
```

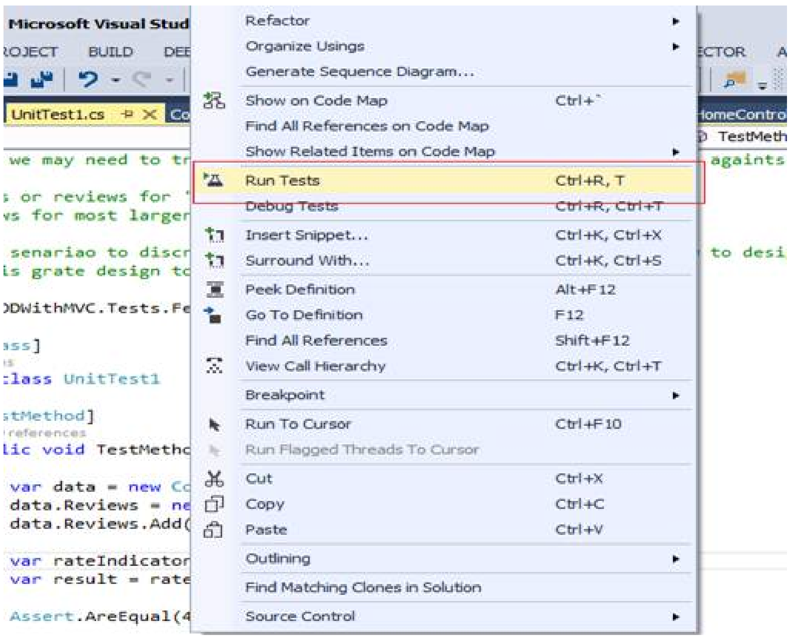
Now we will find below code to the classes,

```
using System.Collections.Generic;

namespace TDDWithMVC.Tests.Features
{
    public class CoffeeShop
    {
        public List < ShopReviews > Reviews
        {
            get;
            set;
        }
    }
}

namespace TDDWithMVC.Tests.Features
{
    public class ShopReviews
    {
        public int ratings
        {
            get;
            set;
        }
    }
}


namespace TDDWithMVC.Tests.Features
{
    class ShopRater
    {
        private CoffeeShop data;

        public ShopRater(CoffeeShop data)
        {
            // TODO: Complete member initialization
            this.data = data;
        }

        public RatingResult ComputeRating(int p)
        {
            return new RatingResult();
        }
    }
}
```

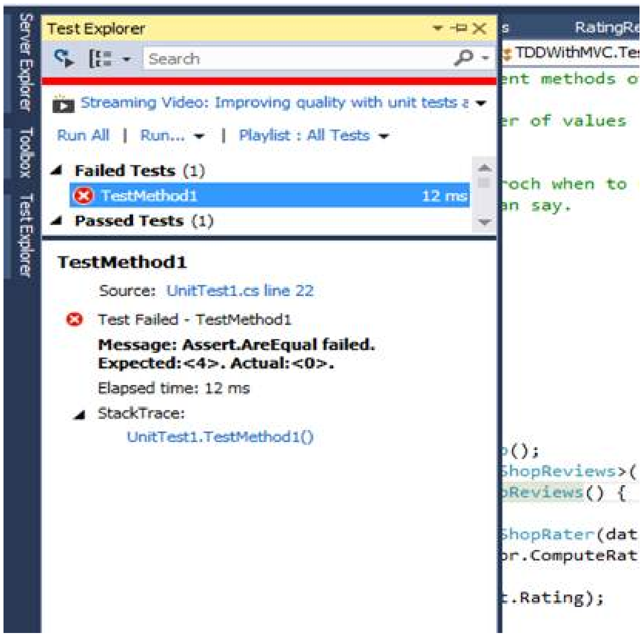
6. Now Build the project as it is stable to build the code.

7. It's time to Run the Unit test as it is not going to pass but as it is our first step to create TDD .

8. Right click on Test method and Run test.


Refactor
Organize Usings
Generate Sequence Diagram…
Show on Code Map Ctrl+`
Find All References on Code Map
Show Related Items on Code Map
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
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining
Find Matching Clones in Solution
Source Control

9. Test has failed; you can see result into Test Explorer.


Test Explorer
Search
Streaming Video: Improving quality with unit tests a
Run All | Run… | Playlist : All Tests
Failed Tests (1)
TestMethod1 12 ms
Passed Tests (1)

TestMethod1

Source: UnitTest1.cs line 22

Test Failed - TestMethod1

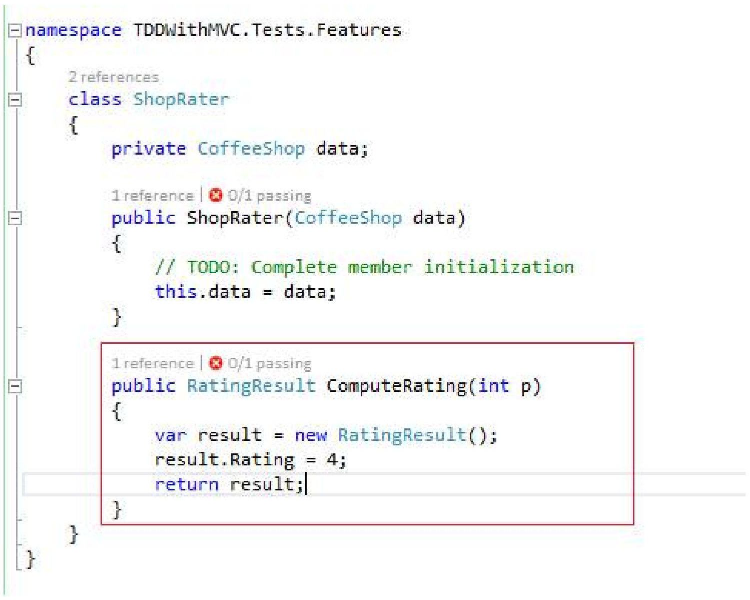
Message: Assert.AreEqual failed. Expected:<4>. Actual:<0>.

Elapsed time: 12 ms

StackTrace:

UnitTest1.TestMethod1()

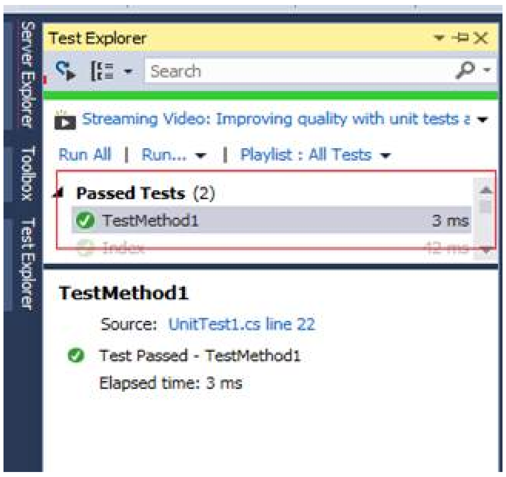
10. Now we will add login to pass the Unit test to fulfill business logic,



```
namespace TDDWithMVC.Tests.Features
{
    2 references
    class ShopRater
    {
        private CoffeeShop data;

        1 reference | 0/1 passing
        public ShopRater(CoffeeShop data)
        {
            // TODO: Complete member initialization
            this.data = data;
        }

        1 reference | 0/1 passing
        public RatingResult ComputeRating(int p)
        {
            var result = new RatingResult();
            result.Rating = 4;
            return result;
        }
    }
}
```

```
public RatingResult ComputeRating(int p)
{
    var result = new RatingResult();
    result.Rating = 4;
    return result;
}
```

11. Run the test and here we go,


Server Explorer
Toolbox
Test Explorer
Test Explorer
Search
Streaming Video: Improving quality with unit tests a
Run All | Run... | Playlist : All Tests

Passed Tests (2)
TestMethod1 3 ms

TestMethod1
Source: UnitTest1.cs line 22
Test Passed - TestMethod1
Elapsed time: 3 ms

Test has passed

But this is killing me as we have hard coding computation to just pass the Unit test. Every time we cannot ask to change values as input changes As tenant we can add more tests and test conditions and for that we need to change the ComputeRating code to work correctly. Going forward we need to insure that we are adding code, condition, and features and that we are not breaking the code; that is what is the real value of the test.

Now change the clean code and do the necessary changes to follow coding conventions.

```
namespace TDDWithMVC.Tests.Features
{
    class ShopRater
    {
        private CoffeeShop _coffeeShop;

        public ShopRater(CoffeeShop coffeeShop)
        {
            // TODO: Complete member initialization
            this._coffeeShop = coffeeShop;
        }

        public RatingResult ComputeRating(int numberOfReviews)
        {
            var result = new RatingResult();
            result.Rating = 4;
            return result;
        }
    }
}
```

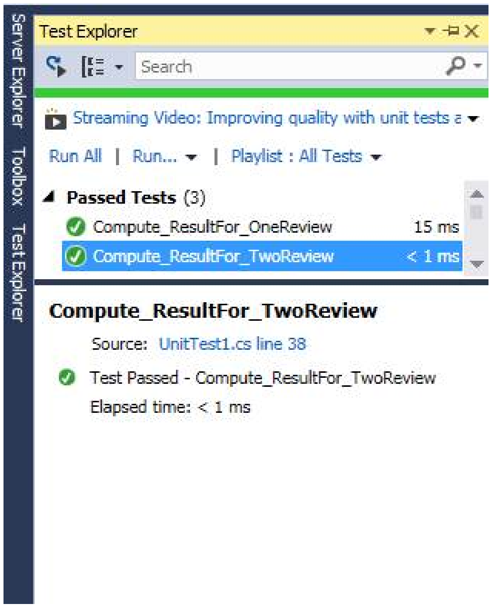
12. Add a new method to check multiple rating,

```
[TestMethod]
public void Compute_ResultFor_OneReview()
{
    // Arrange
    var data = new CoffeeShop();
    data.Reviews = new List < ShopReviews > ();
    data.Reviews.Add(new ShopReviews()
    {
        ratings = 4
    });

    var rateIndicator = new ShopRater(data);
    // Act
    var result = rateIndicator.ComputeRating(10);

    // Assert
    Assert.AreEqual(4, result.Rating);
}

[TestMethod]
public void Compute_ResultFor_TwoReview()
{
    // Arrange
    var data = new CoffeeShop();
    data.Reviews = new List < ShopReviews > ();
    data.Reviews.Add(new ShopReviews()
    {
        ratings = 4
    });
    data.Reviews.Add(new ShopReviews()
    {
        ratings = 8
    });

    var rateIndicator = new ShopRater(data);
    // Act
    var result = rateIndicator.ComputeRating(10);

    // Assert
    Assert.AreEqual(4, result.Rating);
}
```

13. Now refector code the code in ShopRater.cs to pass both the Test,

```
public RatingResult ComputeRating(int numberOfReviews)
{
    var result = new RatingResult();
    result.Rating = (int) _coffeeShop.Reviews.Average(x => x.ratings);
    return result;
}
```

Add Namespace using System.Linq;



Server Explorer Toolbox Test Explorer

Test Explorer

Search

Streaming Video: Improving quality with unit tests a

Run All | Run... | Playlist : All Tests

Passed Tests (3)
Compute_ResultFor_OneReview 15 ms
Compute_ResultFor_TwoReview < 1 ms

Compute_ResultFor_TwoReview
Source: UnitTest1.cs line 38
Test Passed - Compute_ResultFor_TwoReview
Elapsed time: < 1 ms

Now it's time to reactor the test code as test code is also as important as business code,

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;
using System.Linq;


// Calculate Rating using various methods
// for this we may need to try diffrent methods over time to have validation againts
//
// 1. Values or reviews for 'n' number of values
// 2. reviews for most larger values
//
// business senariao to discribe approch when to use TDD : i do not know how to desi
// So, TDD is grate design tool we can say.

namespace TDDWithMVC.Tests.Features
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Compute_ResultFor_OneReview()
        {
            // Arrange
            //var data = new CoffeeShop();
            //data.Reviews = new List<ShopReviews>();
            //data.Reviews.Add(new ShopReviews() { ratings = 4 });

            // Arrange
            var data = BuildReview(rating: 4);

            var rateIndicator = new ShopRater(data);
            // Act
            var result = rateIndicator.ComputeRating(10);

            // Assert
            Assert.AreEqual(4, result.Rating);
        }


        [TestMethod]
        public void Compute_ResultFor_TwoReview()
        {
            // Arrange
            //var data = new CoffeeShop();
            //data.Reviews = new List<ShopReviews>();
            //data.Reviews.Add(new ShopReviews() { ratings = 4 });
            //data.Reviews.Add(new ShopReviews() { ratings = 8 });

            // Arrange
```

```
                var data = BuildReview(rating: new []
                {
                    4,
                    8
                });

                var rateIndicator = new ShopRater(data);
                // Act
                var result = rateIndicator.ComputeRating(10);

                // Assert
                Assert.AreEqual(6, result.Rating);
            }

            private CoffeeShop BuildReview(params int[] rating)
            {
                var coffeeShop = new CoffeeShop();
                coffeeShop.Reviews = rating.Select(x => new ShopReviews
                    {
                        ratings = x
                    })
                    .ToList();
                return coffeeShop;
            }

        }
    }
```

Now we can consider more test scenarios for negative reviews or how to deal with odd numbers, But we will mainly focus on Design (TDD is primarily).

Now add a new test to calculate weighted average of two reviews.

Add new test method as follows

```
[TestMethod]
public void Compute_ResultFor_WeightedReview()
{
    // Arrange
    var data = BuildReview(3, 9);

    var rateIndicator = new ShopRater(data);
    // Act
    var result = rateIndicator.WeightedReviewRating(10);

    // Assert
    Assert.AreEqual(5, result.Rating)
}
```

Add do implementation to class “ShopRater” so that test passes code as follows,

```
public RatingResult WeightedReviewRating(int numberOfReviews)
{
    var review =  coffeeShop.Reviews.ToArray();
    var result = new RatingResult();
    var counter = 0;
    var toatl = 0;

    for (int i = 0; i < review.Count(); i++)
    {
        if (i < review.Count() / 2)
        {
            counter += 2;
            toatl += review[i].ratings * 2;
        } else
        {
            counter += 1;
            toatl += review[i].ratings;
        }
    }

    result.Rating = toatl / counter;
    return result;
}
```

Now a new business change algorithm and anticipating those changes will make it difficult to do changes; for instance, if I add a new method to ShopRater, So let’s refractor code to make changes easier (it's easy to change algorithm and still I am able test if methods pass).

So now we are going to relay on ShopRater to compute actual rating.

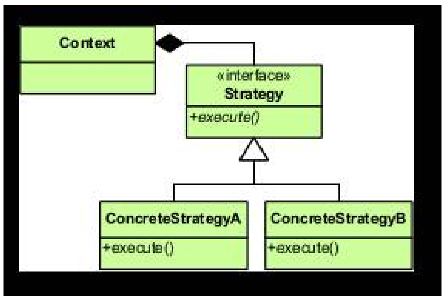
So ComputeResult is relay on IShopRaterAlgorithm which is passed as a parameter which looks as follows,

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TDDWithMVC.Tests.Features
{
    public interface IShopRaterAlgorithm
    {
        RatingResult Compute(IList < ShopReviews > shopReviews);
        //RatingResult ComputeRating(int numberOfReviews);
        //RatingResult WeightedReviewRating(int numberOfReviews);
    }

    public class SimpleRatingAlgorithm: IShopRaterAlgorithm
    {
        public RatingResult Compute(IList < ShopReviews > reviews)
        {
            var result = new RatingResult();
            result.Rating = (int) reviews.Average(x => x.ratings);
            return result;
        }
    }

    public class WeightedRatingAlgorithm: IShopRaterAlgorithm
    {
        public RatingResult Compute(IList < ShopReviews > reviews)
        {
            var review = reviews.ToArray();
            var result = new RatingResult();
            var counter = 0;
            var toatl = 0;

            for (int i = 0; i < review.Count(); i++)
            {
                if (i < review.Count() / 2)
                {
                    counter += 2;
                    toatl += review[i].ratings * 2;
                } else
                {
                    counter += 1;
                    toatl += review[i].ratings;
                }
            }

            result.Rating = toatl / counter;
            return result;
        }
    }
}
And ShopRater.cs like
using System.Collections.Generic;
using System.Linq;

namespace TDDWithMVC.Tests.Features
{
    public class ShopRater
    {
        private CoffeeShop _coffeeShop;

        public ShopRater(CoffeeShop coffeeShop)
        {
            // TODO: Complete member initialization
            this._coffeeShop = coffeeShop;
        }

        public RatingResult ComputeResult(IShopRaterAlgorithm algorithm, int noOfReviews
        {
            var filterReviews =  coffeeShop.Reviews.Take(noOfReviewsToUse);
            return algorithm.Compute(filterReviews.ToList());
        }
    }
}
```

Now Unit test methods will always call method ComputeResult depending on Algorithm,

```
var result = rateIndicator.ComputeResult(new SimpleRatingAlgorithm(), 10);
Or
var result = rateIndicator.ComputeResult(new WeightedRatingAlgorithm(), 10);
```

And this is Strategy Design Pattern: a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern defines a family of algorithms, encapsulates each algorithm, and makes the algorithms interchangeable within that family.



In short:

1. So I have moved code from ShopRater and to Algorithm classes which is assigning specific responsibility to specific classes. So I have algorithm which focuses on computing result.

2. ShopRater which has code needed to reproduce that result.

3. So Test class does not require us to determine which method to call; we always call CompteResult and we pass algorithm which requires to perform computation. Important thing is that we can introduce a new algorithm without changing code inside ShopRater or any of the existing algorithms so that we can extend.

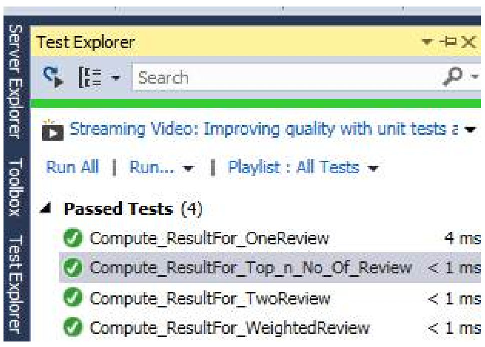
   Now add a method which will compute the average of the first few reviews,

```
[TestMethod]
public void Compute_ResultFor_Top_n_No_Of_Review()
{
    // Arrange
    var data = BuildReview(3, 3, 3, 5, 9, 9);
    var rateIndicator = new ShopRater(data);
    // Act
    var result = rateIndicator.ComputeResult(new SimpleRatingAlgorithm(), 3);
    // Assert
    Assert.AreEqual(3, result.Rating);
}
```

   Then got to ShopRater and change method as below,

```
public RatingResult ComputeResult(IShopRaterAlgorithm algorithm, int noOfReviewsToUs
{
    var filterReviews =  coffeeShop.Reviews.Take(noOfReviewsToUse);
    return algorithm.Compute(filterReviews.ToList());
}
```

Run the test cases



Test Explorer

Streaming Video: Improving quality with unit tests a

Run All | Run... | Playlist : All Tests

Passed Tests (4)

Compute_ResultFor_OneReview	4 ms
Compute_ResultFor_Top_n_No_Of_Review	< 1 ms
Compute_ResultFor_TwoReview	< 1 ms
Compute_ResultFor_WeightedReview	< 1 ms

We are through and also we can move classes and business code to specific projects and folder structures.

Read more articles on **MVC**:

- Validations In ASP.NET MVC 5.0: Part 13
- Getting Started with ASP.NET MVC 5

MVC   MVC Web Application   Test Driven Development   web application