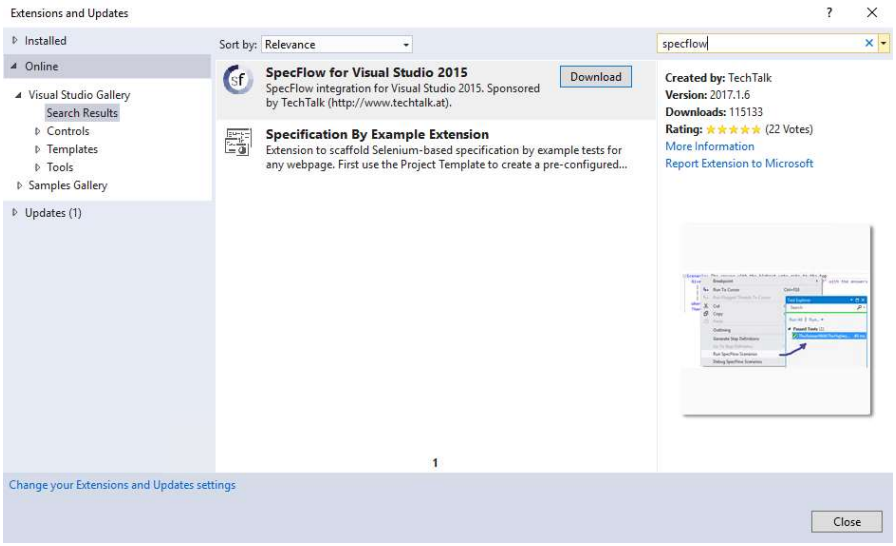
Övning Arbeta med BDD

Övningen visar hur du arbetar med BDD i Visual Studio. I övningen arbetar vi med Cucumber.

Arbetsuppgift 1: Installera SpecFlow

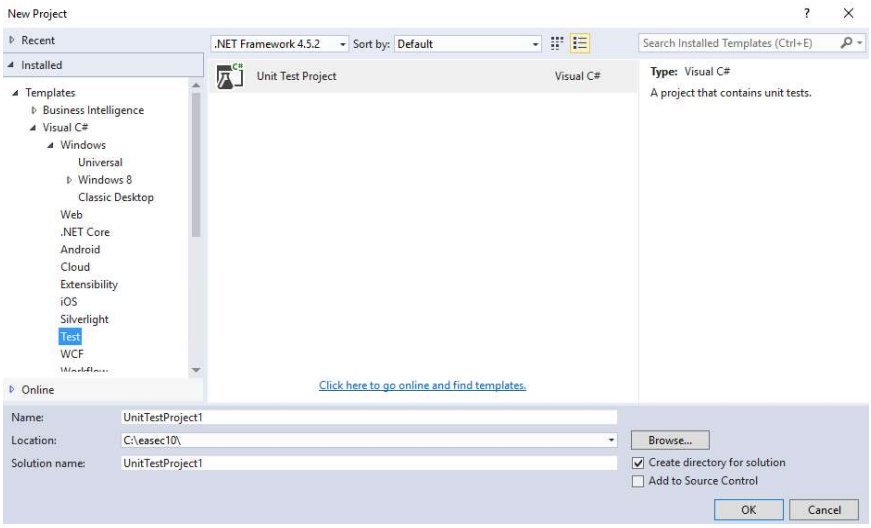
Steg 1: Öppna Visual Studio, klicka på Tools – Extensions and updates – Online, skriv in SpecFlow i sökrutan.



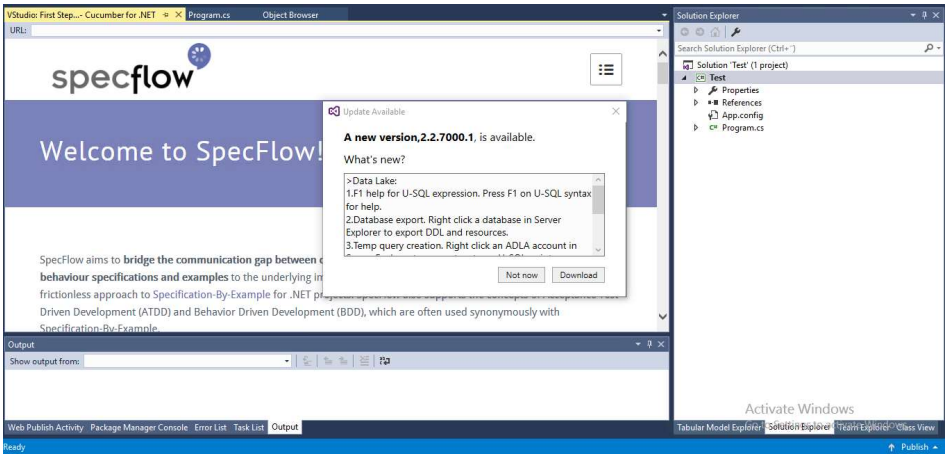
Steg 2: Klicka på Download, när SpecFlow är nerladdat, klicka på Install.

Steg 3: Klicka på Restart, för att starta om Visual Studio.

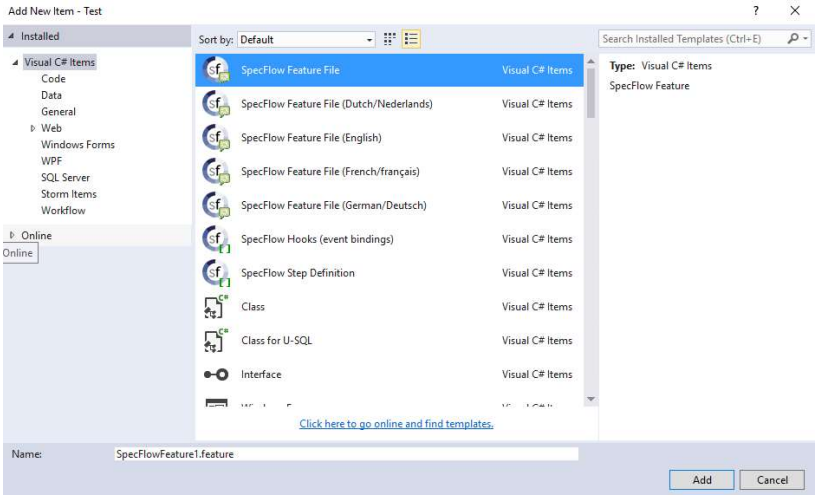
Arbetsuppgift 2: Skapa ett Unit Test Projekt och lägg till SpecFlow



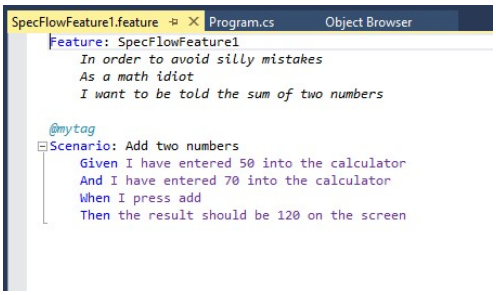
Steg 1: I Visual Studio, klicka på New – Project, under Templates klicka på Test – Test Project. Klicka på OK.



Steg 2: Om dialogfönstret Update Available dyker upp, klicka på Not now.

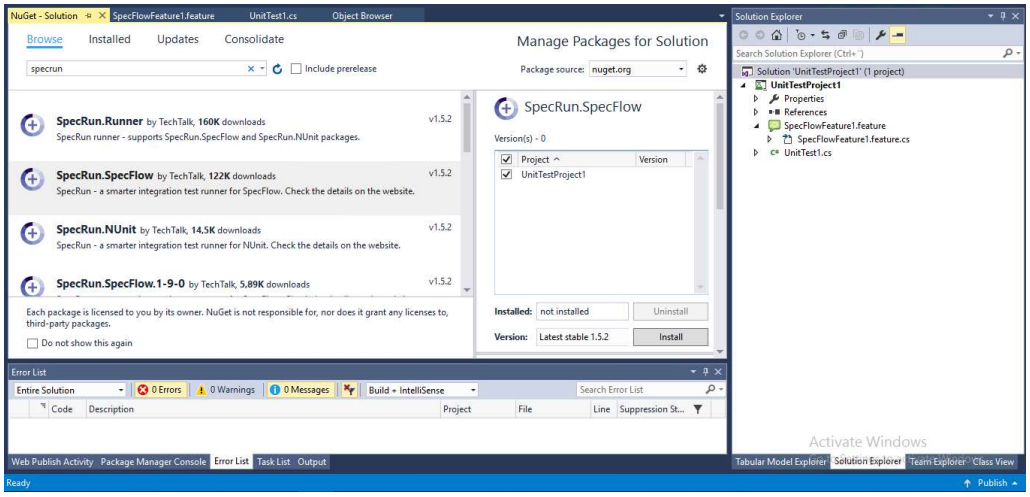


Steg 3: Högerklicka på ditt projekt, välj Add – New Item, markera SpecFlow Feature File, klicka på Add.

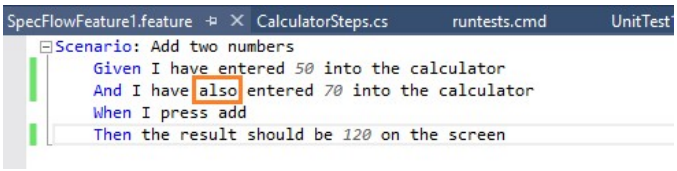


Ett scenario visas, med BDD-terminologi.

Steg 4: Lägg till SpecFlow+ Runner till ditt projekt med NuGet, högerklicka på din lösning och välj alternativet Manage NuGet Packages för Solution.



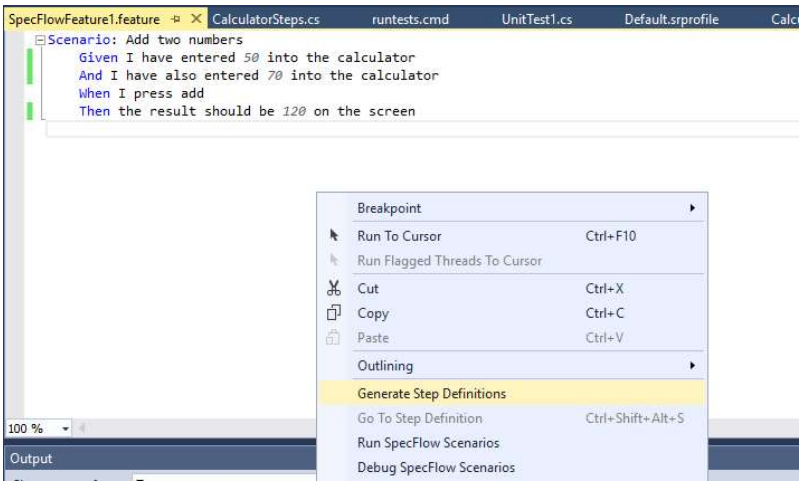
Steg 5: Klicka på Browse, skriv sedan in specrun i sökrutan, klicka på SpecRun.SpecFlow, klicka i boxruta för ditt projekt, klicka sedan på Install. Klicka på OK ett antal gånger för att bekräfta förändringar för din fil.



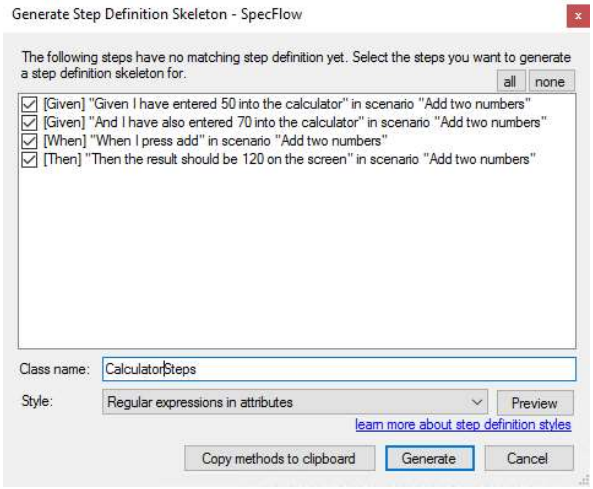
Steg 6: I Solution Explorer, dubbelklicka på filen SpecFlowFeature1.feature, lägg till also på andra raden, så att raden blir så här:

And I have ***also*** entered 70 into the calculator

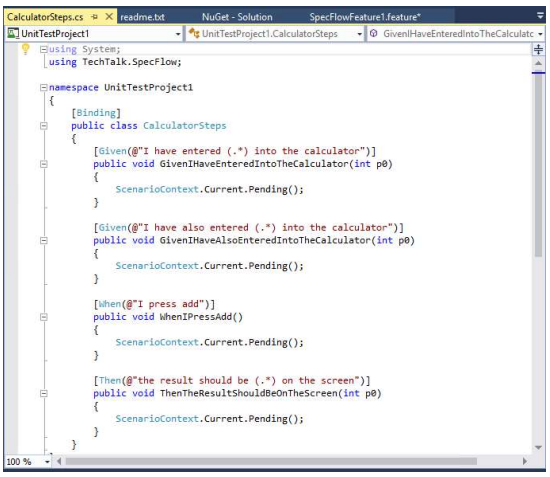
Detta lilla ord är väldigt viktigt, annars kommer inte ett andra given att genereras. Syftet med given är att sätta systemet i ett känt läge, innan det kan användas. When används för att beskriva nyckelfunktioner som användaren utför. Syftet med then är att observera resultatet, istället för flera when, kan and och but användas. Om flera given behövs, använd also.



Steg 7: Högerklicka i code editor, välj alternativet Generate Step Definitions.



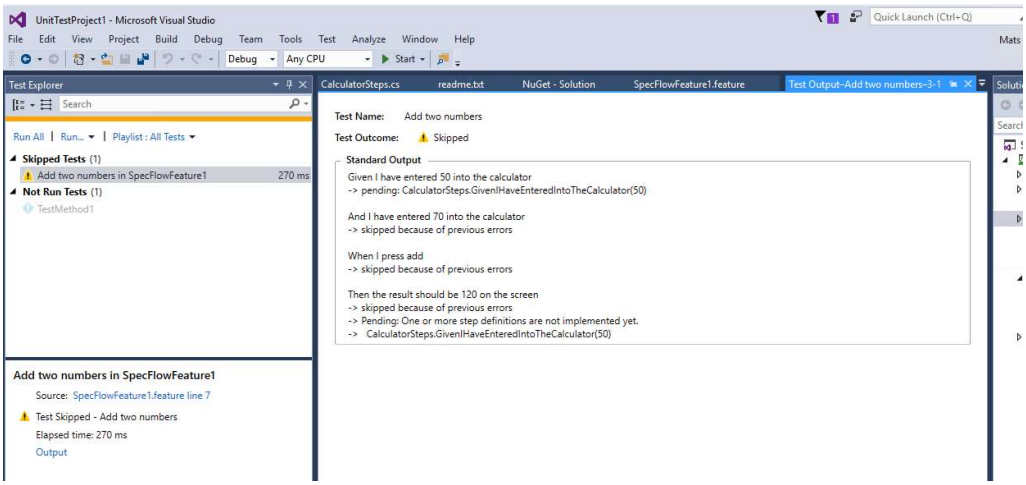
Steg 7: Ändra namn på klass till: CalculatorSteps, klicka först på Generate, därefter på Save.



Steg 9: Klicka på Build – Build Solution.

Arbetsuppgift 3: Kör test

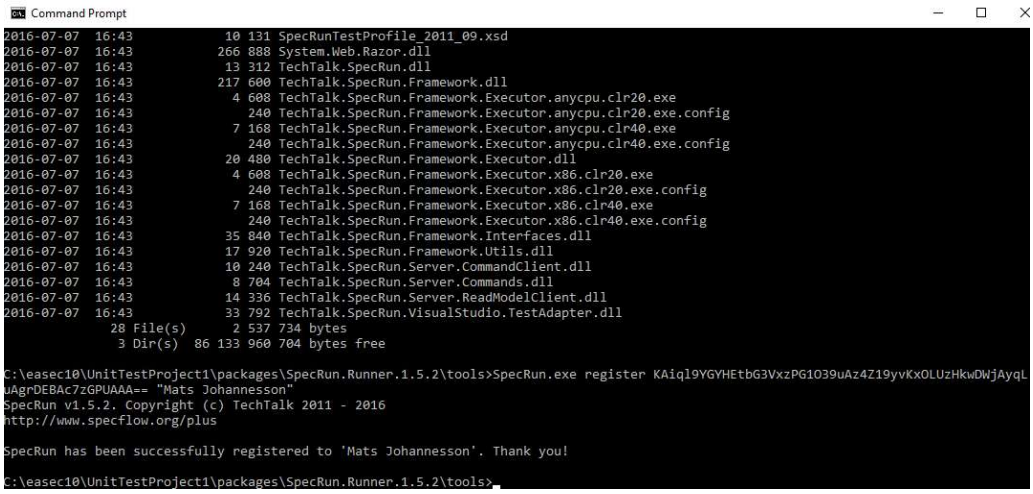
Steg 1: Klicka på Test – Windows – Test Explorer.



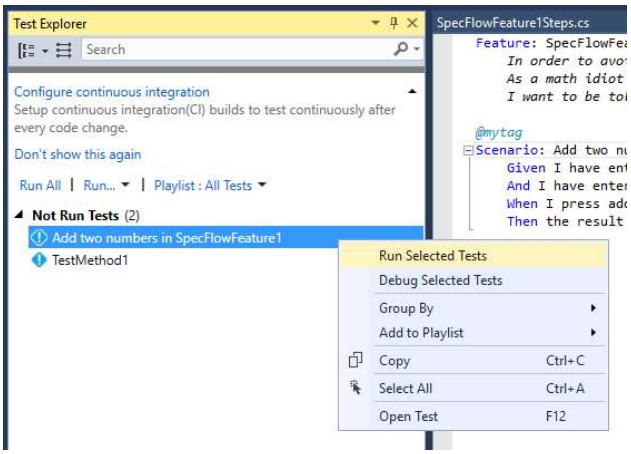
Steg 2: Klicka på Add two numbers in SpecFlowFeature1. För att köra test behöver du en demolicens.

Steg 3: Öppna webläsare, skriv in följande URL: http://specflow.org/plus/request-trial/ och klicka på Enter. Fyll i dina uppgifter.

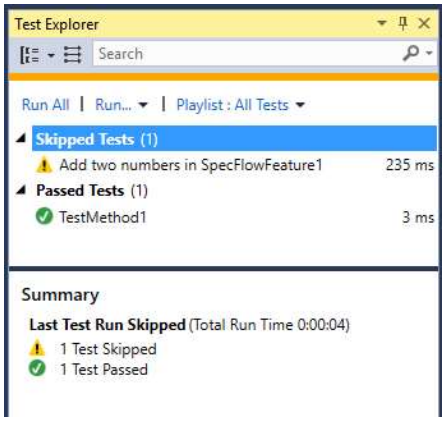
Följ instruktionerna för att installera licensen, finns med i meddelandet.



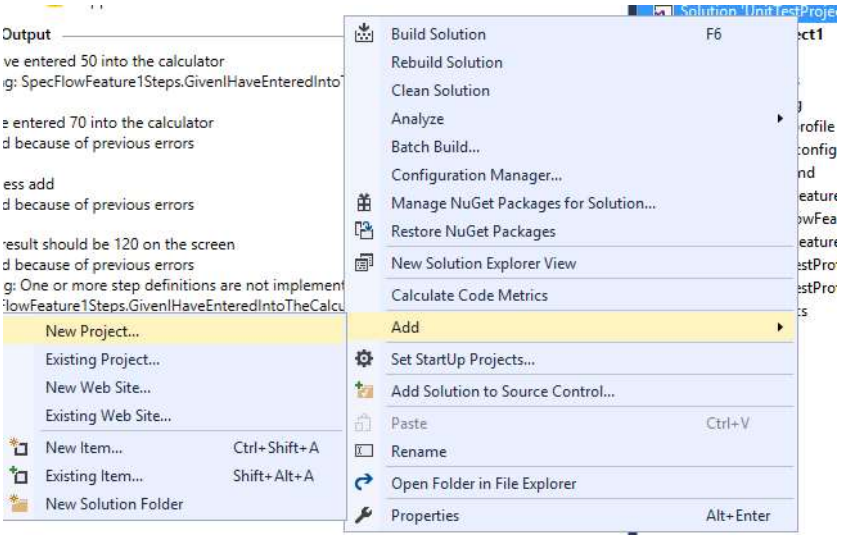
Steg 4: Spara din lösning, stäng ner Visual Studio.



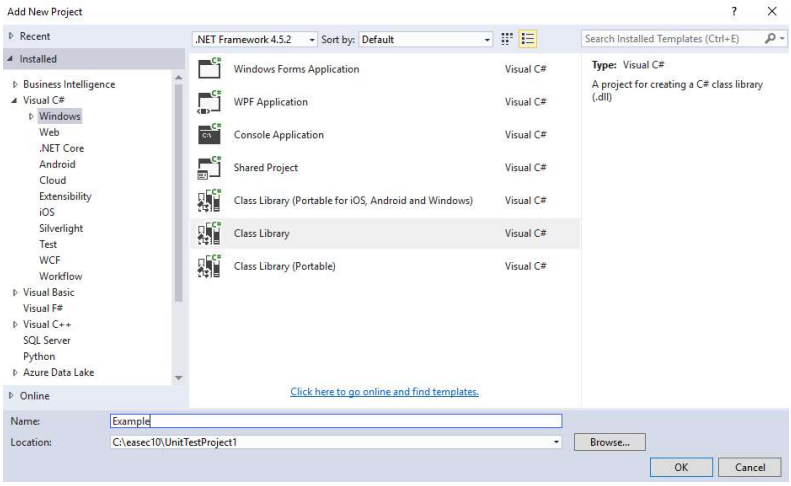
Steg 5: Dubbelklicka på din lösning, högerklicka på Add two numbers ..., välj alternativet Run Selected Tests.



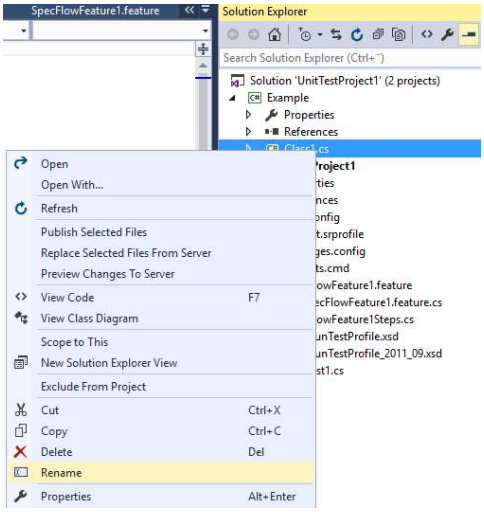
När testet kommer till första metoden, kommer testet att avbrytas.



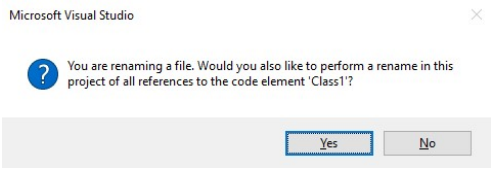
Steg 6: I din lösning, lägg till ett nytt projekt, genom att högerklicka på din lösning, välj Add – Project.

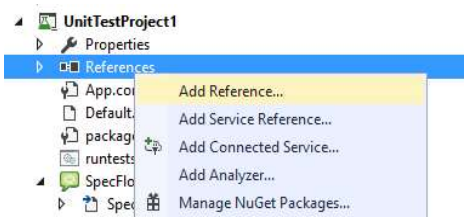


Steg 7: Klicka på Windows – Class Library, skriv in Example i rutan för namn, klicka på OK. Är en hållare för klassen.

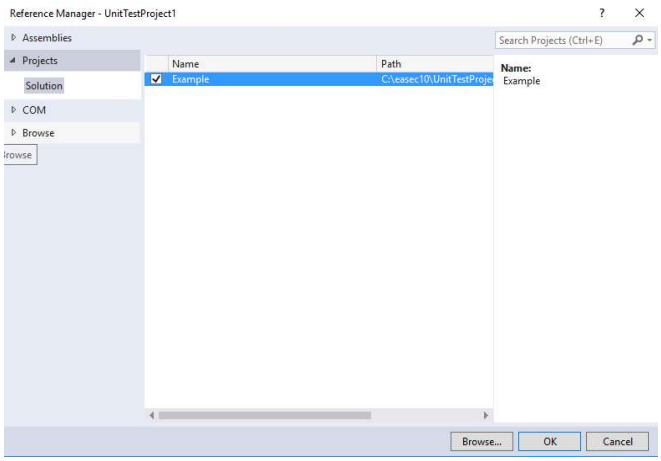


Steg 8: Högerklicka på .cs fil i det nya projektet, döp om den till Calculator.cs, välj att döpa om alla referenser, genom att klicka på Yes i dialogruta som kommer upp.





Steg 9: I Solution Explorer, expandera ditt andra projekt (UnitTestProject1), högerklicka på References och välj Add Reference.



Steg 10: I Reference Manager, under Solution, klicka i boxruta till vänster om Example, klicka på OK.

Steg 11: I Solution Explorer, dubbelklicka på filen CalculatorSteps.cs, leta upp:

```
[Given(@"I have entered (.*) into the calculator")]
public void GivenIHaveEnteredIntoTheCalculator(int p0)
```

Ändra `(int p0)` till `(int number)` .

Steg 12: Leta upp: `ScenarioContext.Current.Pending();` , ersätt med:

```
{
        calculator.FirstNumber = number;
}
```

Steg 12: I Solution Explorer, dubbelklicka på Calculator.cs, lägg till följande i klassen:

```
public int FirstNumber { set; private get; }
```

Steg 13: Lokalisera funktion för det andra Given statement, ändra parameter p0 till number, som tidigare.

Steg 14: Leta upp: `ScenarioContext.Current.Pending();` , ersätt med:

```
{
    calculator.SecondNumber = number;
}
```

Steg 15: Skifta över till Calculator.cs, lägg till följande i klassen:

```
public int SecondNumber { set; private get; }
```

Arbetsuppgift 4: Binda When

Steg 1: Skifta över till CalculatorSteps.cs, definiera en variabel för att lagra resultat, detta görs i början av klassen, innan några steg.

```
private int result { get; set; }
```

Steg 2: Lokalisera funktion för When, ersätt med:

```
public void WhenIPressAdd()
{
    result = calculator.Add();
}
```

Steg 3: Skifta över till Calculator.cs, lägg till följande:

```
public int Add()
{
    return FirstNumber + SecondNumber;
}
```

Arbetsuppgift 5: Binda Then

Then behöver verifiera att resultatet som returneras av metoden Add(), är samma som det förväntade resultatet som är definierat för testet. För att implemetera denna kod:

Steg 1: Lägg till using VisualStudio.TestTools.UnitTesting; , överst i koden, lägg även till using Example.

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

Steg 2: Lägg till följande, innan `private int result { get; set; }`:

```
private Calculator calculator = new Calculator();
```

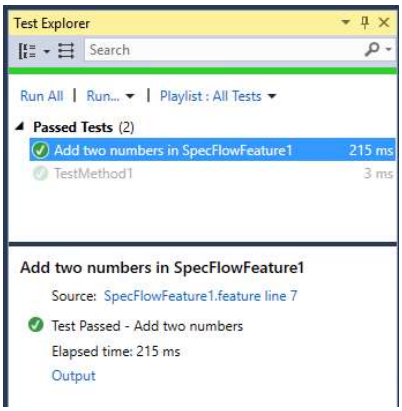
Steg 3: Lokalisera funktion för Then, ändra namnet på p0 parameter till expectedResult och editera definition till:

```
public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
{
    Assert.AreEqual(expectedResult, result);
}
```

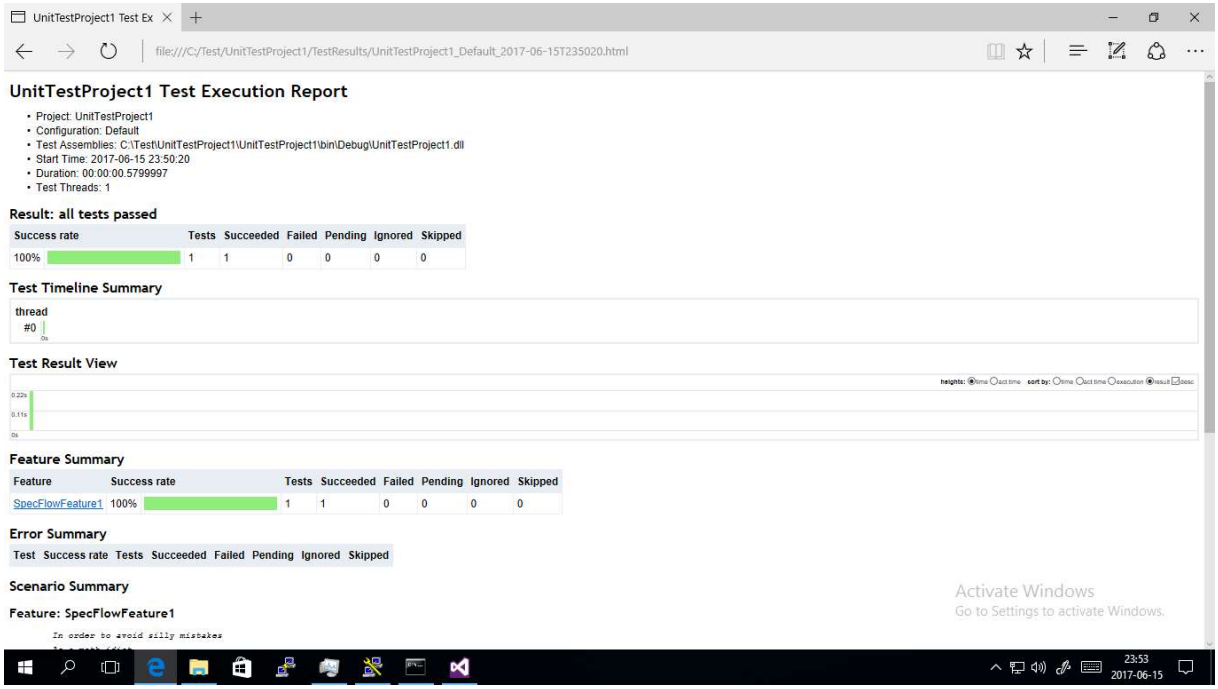
Steg 4: Klicka på Build – Rebuild Solution.

Arbetsuppgift 6: Kör om testet

Steg 1: Kör om testet, detta skall gå igenom nu.



Steg 2: I Outputfönstret finns en länk, markera och klipp ut denna, öppna webbläsare och klistra in länken i adressfältet, klicka på Enter.



CalculatorSteps.cs

```
using System;
using TechTalk.SpecFlow;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Example;


namespace UnitTestProject1
{
    [Binding]
    public class CalculatorSteps
    {
        private Calculator calculator = new Calculator();
        private int result { get; set; }
        [Given(@"I have entered (.*) into the calculator")]
```

```
        public void GivenIHaveEnteredIntoTheCalculator(int number)
        {
            calculator.FirstNumber = number;
        }


        [Given(@"I have also entered (.*) into the calculator")]
        public void GivenIHaveAlsoEnteredIntoTheCalculator(int number)
        {
            calculator.SecondNumber = number;
        }


        [When(@"I press add")]

        public void WhenIPressAdd()
        {
            result = calculator.Add();
        }


        [Then(@"the result should be (.*) on the screen")]
        public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
        {
            Assert.AreEqual(expectedResult, result);
        }

    }
}
```

Calculator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Example
{
    public class Calculator
    {
        public int FirstNumber { set; private get; }
        public int SecondNumber { set; private get; }

        public int Add()
        {
            return FirstNumber + SecondNumber;
        }

    }
}
```