System Testing

* Black box tests that validate the entire system against requirements
* Once these tests are successful, the system is ready for acceptance testing
* Typically conducted by a QA team (internal or external)
* Functional and Nonfunctional requirements are validated
* Should be performed with the system running on required environment

# Functional Testing

* Focuses on how the users will use the system
* User stories and use cases are the primary source of functional requirements

# Nonfunctional Testing

* Focuses on the quality aspects of the system

## Performance

* Validates the speed of the system
* E.g. How long does it take to respond

## Load/Stress Event Testing

* Validates that the system’s breaking point is within acceptable ranges

## Reliability and Availability Testing

* Validates that the mean time between failures (MTBF) is within acceptable ranges

## Recoverability Testing

* Validates that the system can recover from unexpected failures

## Usability Testing

* Validates that the system satisfies stated usability requirements
* Common usability attributes:
  + Accessibility: How easy is it for users to access the features of the system?
  + Responsiveness: How sluggish is the system?
  + Efficiency: How many steps are needed to accomplish a task?
  + Comprehensibility: How easy is the system to understand?

## Security Testing

* Validates the system’s level of security

## Compatibility Testing

* Validating the system’s interaction with other applications

## Installation and Installability Testing

* Validates that the system can be installed

## Serviceability Testing

* Validates how easy it is to repair and enhance the system

Acceptance Test

* Validation – are we building the right system?
* From the point of view of the business/user/customer
* Written in non-technical format (structured)
* Pass/Fail
* Helps document what the system should do
* “Living documentation”
* Typically exercises a vertical slice through the system
* Helps to define “done”

SpecFlow

* Open source tool
* Allows non-technical people to write acceptance tests
* Enables the automation of acceptance tests
* [www.specflow.org](http://www.specflow.org)
* When used in test first approaches
  + While application is not done
    - Collaboratively define what the system should do next
    - Write tests in SpecFlow
    - Repeat
      * Write code
      * until tests pass

# Specflow Structure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature File** (Written in Gherkin) | | | | |
| **Scenario 1**  (Written in Gherkin) | | | **Scenario 2** | **Scenario n** |
| **Step 1** | **Step 2** | **Step n** |  |  |
| **Test Code** | **Test Code** | **Test Code** |  |  |
| **Testing Framework (e.g. NUnit)** | | |  |  |

# Installing SpecFlow

* Use the **Tools > Extensions and Updates...**
  + Search online for SpecFlow
  + Install SpecFlow from TechTalk
  + Restart Visual Studio

# Creating a SpecFlow Feature File

* Create a new class library project
* Add a new item to the project
  + SpecFlow Feature File, the following is autogenerated:

|  |
| --- |
| MyFirstFeature.feature |
| Feature: MyFirstFeature  In order to avoid silly mistakes  As a math idiot  I want to be told the sum of two numbers  @mytag  Scenario: Add two numbers  Given I have entered 50 into the calculator  And I have entered 70 into the calculator  When I press add  Then the result should be 120 on the screen |

Gherkin

* Business readable domain specific language
* Tests are written in natural language albeit structured
* Line-oriented – lines are terminated with newline
* Indentation used to create structure
* Has fixed keywords (e.g. Feature, Scenario)

# Feature

* A small, discrete functionality of the system
* Contains 1 or more scenarios
* Used to group logically related test scenarios

# Feature File

* Start with keyword: Feature
* Followed by a short description, typically verb-noun phrase similar to a use-case
* Followed by optional free text description

|  |
| --- |
| CalculateBMI.feature |
| Feature: CalculateBMI  *In order to stay fit*  *As a fitness freak*  *I want to calculate my BMI* |

## Some Standard Formats

* Role/Person Centric
  + As a... I want... So that

|  |
| --- |
| As a Fitness Freak,  I want to calculate my BMI,  So that I can stay fit. |

* Value Centric
  + In order to... As a ... I want...

|  |
| --- |
| In order to stay fit,  As a Fitness Freak,  I want to calculate my BMI. |

* Requirement Centric (requires the system do something)
  + *<System name>* shall allow *<users>* to *<some objective>* *<some action>*

|  |
| --- |
| BMI Calculator shall allow Fitness Freaks  To calculate their BMI  So that they can stay fit. |

# Scenario

* Concrete examples of expected behavior
* Describes a particular situation
* Each scenario should be ***independent*** and ***isolated***
* Normal (“happy”) paths, Alternative paths, Error, Exception (“sad”) paths, Edge cases

# Scenario in the Feature File

* Starts with keyword: Scenario
* Followed by scenario title
* Followed by scenario steps
  + Setup initial state
    - **Given**
  + Perform action(s)
    - **When**
  + Check end state
    - **Then**

|  |
| --- |
| CalculateBMI.feature |
| Feature: CalculateBMI  *In order to stay fit*  *As a fitness freak*  *I want to calculate my BMI*  Scenario: Optimal BMI  Given I navigate to the BMI page  And I have entered 120 as the weight  And I have entered 66 as the height  When I press Submit  Then the result should be BMI:19.4 and BMI message:optimal |

# Tags

* Use: @
* Categorizes the scenarios
* Tags are applied to features and scenarios
* There can be multiple tags
* @ignore – ignores the tagged scenario(s)

|  |
| --- |
| CalculateBMI.feature |
| Feature: CalculateBMI  *In order to stay fit*  *As a fitness freak*  *I want to calculate my BMI*  *@Normal\_Flow*  Scenario: Optimal BMI  Given I navigate to the BMI page  And I have entered 120 as the weight  And I have entered 66 as the height  When I press Submit  Then the result should be BMI:19.4 and BMI message:optimal  *@Exceptions\_Flow*  Scenario: Invalid weight  Given I navigate to the BMI page  And I have entered -1 as the weight  When I press Submit  Then "Weight is invalid" should be displayed next to the weight input box |

# Comments

* Use: #
* A single line

# Data Tables

* Allows tabular data to be passed to an automation step
* Add colon (:) at the end of the step
* Use pipes (|) to build the table
* The first column specifies the column name (becomes the parameter in code)

|  |
| --- |
| BasicSearchForStories.feature |
| ...  Scenario: Successful Search  Given I'm on the EStR main page  And the search field is empty  When I type in Ghost  And Tent  And Sissy  And then I issue the search command  Then there should be at least one story displayed |
| ...  Scenario: Successful Search  Given I'm on the EStR main page  And the search field is empty  When I type in:  | searchTerm |  | Ghost |  | Tent |  | Sissy |  And then I issue the search command  Then there should be at least one story displayed |

# Scenario Outlines

* Execute the scenario multiple times with a different set of data each time
* Use pipes (|) to build the table
* The first column specifies the column name (becomes the parameter in code)

|  |
| --- |
| CalculateBMI.feature |
| ...  Scenario: Optimal BMI  Given I navigate to the BMI page  And I have entered <weight> as the weight  And I have entered 66 as the height  When I press Submit  Then the result should be BMI:19.4 and BMI message:optimal  Examples:  | *weight* |  | 120 |  | 130 |  *...* |

# Background

* Provides state setup to the scenarios in a feature
* Executed before each scenario

|  |
| --- |
| CalculateBMI.feature |
| Feature: CalculateBMI  *In order to stay fit*  *As a fitness freak*  *I want to calculate my BMI*  Background:  Given I navigate to the BMI page  *@Normal\_Flow*  Scenario: Optimal BMI  Given I have entered 120 as the weight  And I have entered 66 as the height  When I press Submit  Then the result should be BMI:19.4 and BMI message:optimal  *@Exceptions\_Flow*  Scenario: Invalid weight  Given I have entered -1 as the weight  When I press Submit  Then "Weight is invalid" should be displayed next to the weight input box |

Coding the Automation Steps

* Needed Nuget Packages
  + **Specflow.NUnit**
    - **SpecFlow** package will also be installed

# Binding

* Hooking up the steps to coded tests

# Generating the Test Class

* Right-click the scenario and then select “Generate Step Definitions”
  + Name the class
  + Select the style
    - Preferred style: Method name – pascal case

## Setting the Default Style

* Open **App.config**
  + Add <trace stepDefinitionSkeletonStyle="MethodNamePascalCase" /> between the <specFlow> tags

# Adding New Steps

* Be careful not to overwrite previous steps!
* Generate Step Definitions > **Copy methods to clipboard**

# Running the Scenarios

* Need a test runner (Use Extensions and Updates to install NUnit Test Adapter)
* Build the solution to have the tests appear in the ***test explorer***
* Run or debug as usual

# Sharing Step Definitions

* SpecFlow will automatically match existing step definitions

## Parameterized Step Definitions

* If a step has common items across scenarios, then it may be possible to parameterize it.
  + For example
    - Scenario 1 has: When I type in *Ghost*
    - Scenario 2 has: When I type in *Tent*
  + The *Ghost* and *Tent* can be parameterized.
    - public void WhenITypeIn\_SEARCHTERM(string searchTerm)
* The parameterization is CAPITALIZED in the method name.

## Multiple and String Parameters in a Single Step Definition

* Each parameter is CAPITALIZED in the method name
* For example
  + When I type *Tent* and then *Ghost*
  + public void WhenIType\_FIRSTSEARCHTERM\_AndThen\_SECONDSEARCHTERM(string firstSearchTerm, string secondSearchTerm)

# Creating a step definition with a Data Table

* The data table is passed to the method with data type: Table

|  |
| --- |
| When I type in:  | searchTerm |  | Ghost |  | Tent |  | Sissy | |
| [When]  public void WhenITypeIn(Table table)  {  var term1 = table.Rows[0][0];  var term2 = table.Rows[1][0];  var term3 = table.Rows[2]["searchTerm"];  var hasSearchTermColumn = table.ContainsColumn("searchTerm");  ScenarioContext.Current.Pending();  } |

# Scenario Outline Step Definitions

* Same idea as parameterized step definitions
* Be careful with quotations in the scenario

# Sharing and Maintaining State between Step Definitions

* Can use ScenarioContext to store key-value pairs
  + ScenarioContext.Current.Add("key", value);
  + var value = ScenarioContext.Current["key"];
* Can create attributes in the feature class if all steps are in the same class

Automated Functional UI Testing a Website (Tutorial)

# Using SpecFlow and Selenium

1. Create the class library C# project: BMISystemAcceptanceTesting
2. Install the SpecFlow.NUnit Nuget package
3. Add a feature file: CalculateBMI.feature

|  |
| --- |
| Feature: CalculateBMI  In order to stay fit  As a fitness freak  I want to calculate my BMI |

1. Add the background and one normal flow scenario:

|  |
| --- |
| Background:  Given I navigate to the BMI page  @Normal\_Flow  Scenario: Optimal BMI  Given I have entered 120 as the weight  And I have entered 66 as the height  When I press Submit  Then the result should be BMI:19.4 and BMI message:optimal |

1. Add the default step generation to App.config between the <specFlow> tags:
   * <trace stepDefinitionSkeletonStyle="MethodNamePascalCase" />
2. Generate the step definitions and modify the parameterized methods:

|  |
| --- |
| using System;  using TechTalk.SpecFlow;  namespace SpecflowFeature  {  [Binding]  public class CalculateBMISteps  {  [Given]  public void GivenINavigateToTheBMIPage()  {  ScenarioContext.Current.Pending();  }    **[Given]**  **public void GivenIHaveEntered\_WEIGHT\_AsTheWeight(int weight)**  **{**  **ScenarioContext.Current.Pending();**  **}**    **[Given]**  **public void GivenIHaveEntered\_HEIGHT\_AsTheHeight(int height)**  **{**  **ScenarioContext.Current.Pending();**  **}**    [When]  public void WhenIPressSubmit()  {  ScenarioContext.Current.Pending();  }  **[Then]**  **public void ThenTheResultShouldBe\_BMIVALUE\_And\_BMIMESSAGE(string bmiValue, string bmiMessage)**  **{**  **ScenarioContext.Current.Pending();**  **}**  }  } |

1. Code the first step:

|  |
| --- |
| [Given]  public void GivenINavigateToTheBMIPage()  {  BMIPage.Initialize();  BMIPage.GoTo();  } |

1. To the solution, create a new C# class library project: BMITestFramework
2. Add a reference from BMIAcceptanceTesting to BMITestFramework
3. Add a public static class to BMITestFramework:

|  |
| --- |
| namespace BMITestFramework  {  public static class BMIPage  {  }  } |

1. Generate the method stubs for BMIPage.Initialize() and BMIPage.Goto()
2. Add code to BMI:

|  |
| --- |
| namespace BMITestFramework  {  public static class BMIPage  {  private const string Url = "http://einstein.etsu.edu/~roachj/bmi/";  public static void Initialize()  {  Chrome.Create();  }  public static void GoTo()  {  Chrome.Goto(Url);  }  }  } |

1. Create a new C# class library project: WebDriverFramework
2. Add a reference from BMITestFramework to WebDriverFramework
3. Add a public static class to WebDriverFramework:

|  |
| --- |
| namespace WebDriverFramework  {  public static class Chrome  {  }  } |

1. Generate the method stubs for Chrome.create() and Chrome.Goto(Url)
2. Download ***ChromeDriver – WebDriver*** for Chrome (<https://sites.google.com/a/chromium.org/chromedriver/downloads>) and save it somewhere convenient.
3. Install the ***Selenium WebDriver*** packages using Nuget to project ***WebDriverFramework***
4. Add code to Chrome:

|  |
| --- |
| namespace WebDriverFramework  {  public static class Chrome  {  private static IWebDriver \_page = null;  public static void Create()  {  \_page = new ChromeDriver(@"***<Path to the webdriver executable>***");  }  public static void Goto(string url)  {  \_page.Navigate().GoToUrl(url);  }  }  } |

1. Build the solution and then run the test from the test explorer
2. Modify the Binding class to close the browser:

|  |
| --- |
| [Binding]  public class CalculateBMISteps  {  **[Before]**  **public static void Setup()**  **{**  **BMIPage.Initialize();**  **}**  **[After]**  **public static void TearDown()**  **{**  **BMIPage.EndTest();**  **}**  [Given]  public void GivenINavigateToTheBMIPage()  {  **~~BMIPage.Initialize();~~**  BMI.Goto();  } |

1. Add the EndTest() method:

|  |
| --- |
| ***public static class BMIPage***  public static void EndTest()  {  Chrome.Quit();  } |

1. Add the Quit() method:

|  |
| --- |
| ***public static class Chrome***  public static void Quit()  {  \_page.Dispose();  \_page.Quit();  } |

1. Build the solution and then run the test from the test explorer
2. Code the second step:

|  |
| --- |
| [Given]  public void GivenIHaveEntered\_WEIGHT\_AsTheWeight(int weight)  {  BMIPage.Weight = Convert.ToString(weight);  } |

1. Add Weight property to the BMIPage class:

|  |
| --- |
| public static string Weight  {  set { Chrome.Weight = value; }  } |

1. Add Weight property to the Chrome class :

|  |
| --- |
| public static string Weight  {  set  {  var weightElement =  \_page.FindElement(  By.CssSelector("input[name=\"weight\"]"));  weightElement.Clear();  weightElement.SendKeys(value);  }  } |

1. Build and then run the test
2. Code the third step:

|  |
| --- |
| ***public class CalculateBMISteps***  [Given]  public void GivenIHaveEntered\_HEIGHT\_AsTheHeight(int height)  {  BMIPage.Height = Convert.ToString(height);  } |
| ***public static class BMIPage***  public static string Height  {  set { Chrome.Height = value; }  } |
| ***public static class Chrome***  public static string Height  {  set  {  var heightElement =  \_page.FindElement(  By.CssSelector("input[name=\"height\"]"));  heightElement.Clear();  heightElement.SendKeys(value);  }  } |

1. Build and run the test
2. Code the fourth step:

|  |
| --- |
| [When]  public void WhenIPressSubmit()  {  BMIPage.Submit();  } |

1. Add the Submit class to BMIPage:

|  |
| --- |
| public static void Submit()  {  Chrome.ClickSubmit();  } |

1. Add the ClickSubmit() method to Chrome:

|  |
| --- |
| public static void ClickSubmit()  {  \_page.FindElement(By.CssSelector("input[type=\"submit\"]")).Click();  } |

1. Build and run the test
2. Code the assert step:

|  |
| --- |
| [Then]  public void ThenTheResultShouldBe\_BMIVALUE\_And\_BMIMESSAGE(string bmiValue, string bmiMessage)  {  var actualBmiValue = BMI.MainPage.BMIValue;  var actualBmiMessage = BMI.MainPage.BMIMessage;  Assert.That(actualBmiValue, Is.EqualTo(bmiValue));  Assert.That(actualBmiMessage, Is.EqualTo(bmiMessage));  } |
| ***public static class BMIPage***  public static string BMIValue  {  get { return Chrome.BMIValue; }  }  public static string BMIMessage  {  get { return Chrome.BMIMessage; }  } |
| ***public static class Chrome***  public static string BMIValue  {  get  {  return \_page.FindElement(By.XPath("/html/body/div/p[1]")).Text;  }  }  public static string BMIMessage  {  get  {  return \_page.FindElement(By.XPath("/html/body/div/p[2]")).Text;  }  } |

1. Build and run the test