Cost Effective Agile Test Practices and Test Automation using Open Source Tools SpecFlow and White

Kavitha Naveen and Perry Hunter
Tektronix
kavin@tek.com and perry.w.hunter@tektronix.com

Abstract

Budget! Cost optimization! Expensive! These are common words which are often heard when licensed automation tools are proposed for software test automation.

If licensed test tools are expensive and budget is the key constraint, don't worry; Open source tools are there to help us in creating effective and low-cost test automation. Open source tools have an added advantage of supporting Behavior-Driven Development (BDD) which is an agile software development technique that encourages collaboration between developers, testers and non-technical or business participants in a software project. Open source tools are free; just get it and use it to see the power of test automation.

This paper is about a Test Framework which was developed using open source tools to test plugins embedded in Visual Studio 2010.

Among the many open source tools available, this presentation will showcase how two tools, SpecFlow and the White library, have made test automation tasks easy to complete!

Writing elaborate test cases is a big pain, isn't it? Don't worry; BDD comes to your rescue. You can write simple and easily readable test cases in the language you speak. SpecFlow from TechTalk brings BDD to the .NET environment. It can be used to write test cases using BDD style in Visual Studio.

For GUI automation we have the White library from Thought Works. It supports automation for Visual Studio 2010 Controls and plugins embedded in Visual Studio 2010.

Biography

Kavitha Naveen is a senior software quality lead engineer at Tektronix Engineering Development India Ltd. Over the past few years, she has been involved in test automation for products used in Test and Measurement Industry. Kavitha has worked extensively on training test engineers and researching test methodologies. Kavitha has a Bachelor's of Engineering in Telecommunication from Bangalore University, India.

Perry Hunter is a senior software quality lead engineer at Tektronix in Beaverton, OR. He has worked in the fields of scientific research and software quality for approximately 25 years, and is currently supporting the signal generator group at Tektronix. He enjoys gardening, most products of fermentation and is passionate about Kendo. His first real computer was a Data General MV/8000.
Perry holds a Bachelors of Science in Oceanography from Humboldt State University.

1 Introduction

Every software test engineer will be required to write test cases at some point during the software project cycle to test the product under development. Each person follows his own method to write the test cases. Some test engineers write elaborate test cases while others write very brief test cases. Finally, when the test cases are ready, they need to be reviewed. Usually, when the test cases are sent for review there are hardly any review comments, as the software team finds reasons not to do it. Even if it is done, the review is often not very extensive. The end result is frequently test cases not getting reviewed and missing test conditions— which are defects seen later in the end product.

One of the reason for test cases not getting reviewed is they are frankly less interesting than coding to the team members. No one likes to switch context and decipher elaborate test code. Therefore if we write test cases in a more readable language, use precise words to explain the test scenario and then automate the test case line by line, then the test case review time can be substantially reduced. The quality of the test cases is improved, because the reviewers can clearly understand what is being tested and how.

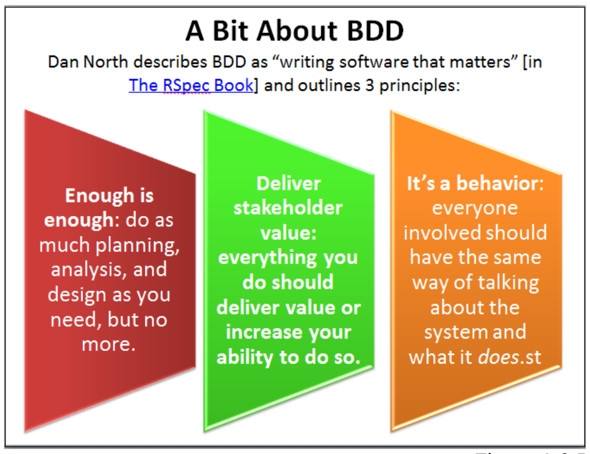
In this paper, we describe how two open source tools; i.e., SpecFlow and White, combined with Microsoft Visual Studio, help us to carry out GUI test automation using agile test practices in the Windows environment. SpecFlow helps us to write test cases using a BDD style in the .NET environment with simple sentences which are easily understood by all stakeholders. The White Library from Thought Works helps us to carry out GUI test automation.

We will also share how we automated Microsoft development tools like Visual studio 2010 and achieved nearly 90% automation for one of our projects using the open source tools. The whole process—starting from evaluating the tools, implementing the automation scripts, and running the scripts successfully for the project—took about 12 weeks' time.

2 Incorporating Agile Testing using Open Source Tools

Agile testing has become a common phrase in the software industry. But now the question is, where do I start and how do I proceed? When it comes to following agile test practices, the concentration is more on team work rather than individuals like testers or developers. Hence, our thought process also needs to change. Test engineers need to think of following agile practices right from the start

Agile for us really means doing things in a simplified but rapid way. For writing the test cases and collaborating well with the requirements stakeholders, BDD comes to the rescue. Here the test cases are written in simple, business readable language. BDD is a widespread methodology used in the agile community.



A Bit About BDD

Dan North describes BDD as "writing software that matters" [in The RSpec Book] and outlines 3 principles:

Enough is enough: do as much planning, analysis, and design as you need, but no more.

Deliver stakeholder value: everything you do should deliver value or increase your ability to do so.

It's a behavior: everyone involved should have the same way of talking about the system and what it *does*.st

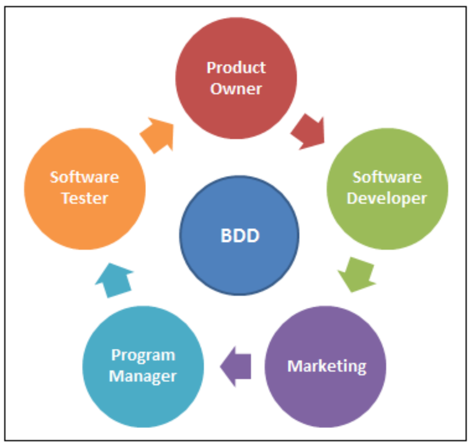


Figure 1.0 Behavior Driven Development

One of the reasons why the defects escape in the released product is due to requirements not being clear and thereby missing test cases or incorrect test cases. By using BDD to write the requirements and test cases wherein emphasis is given to the vocabulary, writing incorrect test cases reduces to a great extent.

Test cases written in BDD style reduce review time as they become automated. The test case is written in the form of Features and Scenarios.

2.1 Cucumber

Cucumber (written in the Ruby programming language) is a tool used for writing test cases in BDD style. Gherkin is the language that is used to write BDD style test cases in Cucumber.

A feature is something that your software does (or should do), and generally corresponds to a user story and a way to tell when it's finished and that it works.

The general format of a feature is:

```
Feature: <short description>
  <story>
  <scenario 1>
  ...
  <scenario n>
```

A scenario is made up of 3 sections related to the 3 types of steps:

1. ***Given***: This sets up preconditions, or context, for the scenario.
2. ***When***: This is the action, the behavior that we're focused on.
3. ***Then***: This checks results… it verifies that the right thing happened as a result of the *When* steps.

A simple example to illustrate writing Test Cases in Cucumber BDD Style would be as given below

```
Feature: Bug Reports

As an SQA Engineer
I want to search defects
So that I can keep my team informed

Scenario: Finding severe Defects
  Given there are 5 defects on "Project A"
  And 2 of those are marked "Critical"
  When I search for defects marked "Critical" on "Project A"
  Then I should see 2 defects
```

All the test cases written above are in plain English. Note the simple sentences used to describe the feature and the test case scenarios. When test cases are written in this style the entire can more easily understand the given conditions, the action conditions and the expected result conditions.

If the right tool exists to enable and simplify writing BDD style test cases and to help us follow agile test practices, then the burden on the team to follow agile practices will be drastically reduced.

Once the right tool is selected for test case development, then the other tasks like test script development and functional testing will fall in place. The team will be following Agile Test practices right from test case development.

In the next section we will see how the open source tool SpecFlow is used for writing test cases in BDD style.

3 Selecting SpecFlow for Behavior Driven Development in the .NET environment

As explained in previous section, BDD helps us to write test cases in plain English and then convert the same test cases to test scripts. In the .NET environment, SpecFlow comes in handy for writing BDD style test cases. It is an open source tool that supports the implementation of the ideas behind agile methodologies like BDD. It lets you write specifications using 100%-Cucumber-compatible Gherkin Syntax within the Visual Studio environment. In the upcoming sections, we shall see the advantages of SpecFlow.

3.1 SpecFlow is Open Source

The budget gets tightened often in organizations due to reasons like recession, low profit margins, and managers always demanding “quicker, cheaper, better” solutions. SpecFlow being an open source tool is an added advantage due to these reasons. It is a lightweight installation and integrates with Visual Studio automatically once you install the software. Using open source tools helps us lower cost of ownership and reduces dependencies on specific products. It promotes faster time to market, flexibility to customize, and ease of procurement. Another advantage is the open source community which is there to support us. There is a well-led core team who quickly responds to our queries and provides solutions for us to move forward. There is continual improvement in the tool and iterative release process with a large number of developers and test engineers contributing to the tool by reporting defects and enhancement requests. Most of the expensive commercial test tools are used by a limited number of developers and testers and hence the scope for improvement is limited when compared to open source tools which are widely used. So the quality and reliability are continuously improving in open source tools.

3.2 SpecFlow Supports Agile Testing

In agile testing we need to be as quick as possible right from writing test cases to testing the product. The test team needs to be ready to run the tests on a daily basis. Instead of writing elaborate test cases, writing test cases in BDD style helps us have acceptance test cases for the requirements. The acceptance criteria are well defined in the test case itself. When the test cases are executed, there is valid proof that all the requirements have been captured along with the acceptance criteria due to the well-defined test conditions that are specified in the test case.

In software development life cycle conventionally, the testing cycle would come at the end of the project life cycle. A linear and sequential approach which can be viewed as the waterfall model is followed i.e. after all the features of the product are implemented, testing activities would start. In this model until the final stage of the development life cycle the working software with all the features implemented is not available. Defects will get accumulated and the cost of defect fixes at the end of the project life cycle becomes very huge. The development team and testing team are under severe pressure to complete their tasks during the end of the project. The probability of the working features starting to fail is high. Side effects get introduced due to defect fixes, and the whole team will be in a state of chaos. This leads to failure of the project. Hence many companies have changed their traditional approach of following water fall model, by exploring new ways which includes agile testing. Agile testing ensures quality of the product as well as the time to market. Following agile test practices as early as possible in the development cycle saves time and money of the organization

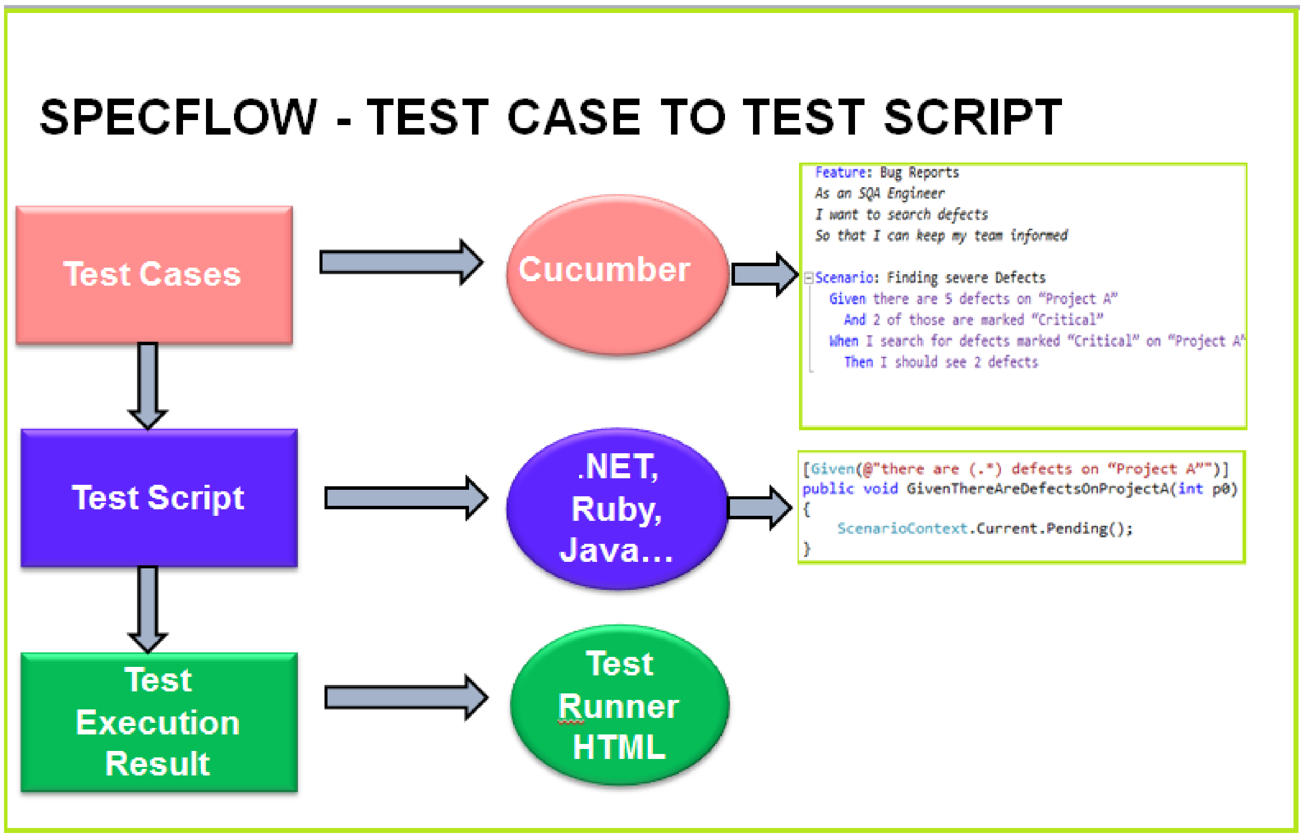
Due to the flexible processes and the ability to continuously improve the product during the development stage, agile testing has achieved significant success in recent days.

With SpecFlow, we don't have to use another test tool to capture the test cases. The aim of agile testing to complete tasks quickly and be ready for product testing can be easily appreciated when SpecFlow is used for writing test cases and converting them to test scripts. It provides a seamless developer experience by leveraging existing infrastructure and frictionless integration into the .NET platform. SpecFlow is compatible with the Gherkin language. It allows us to write test scripts in any .NET language. We use existing unit-testing frameworks as the runtime for scenario execution.

SpecFlow consists of three parts.

- Feature file converter to produce test fixtures (integrated into Visual Studio through a single-file generator)
- Runtime framework (handles step bindings, events and tracing)
- Integration with unit test frameworks (NUnit, MSTest, xUnit.Net, MbUnit)

The flow of data in SpecFlow from test case to test script and result is explained in the following diagram.



```
Feature: Bug Reports
As an SQA Engineer
I want to search defects
So that I can keep my team informed

Scenario: Finding severe Defects
  Given there are 5 defects on "Project A"
    And 2 of those are marked "Critical"
  When I search for defects marked "Critical" on "Project A"
    Then I should see 2 defects
```

```
[Given(@"there are (.*) defects on "Project A"")]
public void GivenThereAreDefectsOnProjectA(int p0)
{
    ScenarioContext.Current.Pending();
}
```

Figure 2.0 - Test Framework Architecture

3.2.1 Writing Test Cases Using SpecFlow in Microsoft Visual Studio IDE

SpecFlow is a plugin embedded in Visual Studio. SpecFlow and Visual Studio complement each other and help us to manage and automate our tests. SpecFlow helps in BDD, while the various test functionalities embedded in Visual Studio 2010 control test development and execution.

In the upcoming sections, we shall see step by step how to create test cases and execute them using SpecFlow. When SpecFlow is installed it gets added as a plugin in Visual Studio IDE as shown below.

Now we just need to add a reference to the TechTalk. SpecFlow DLL in the Visual Studio project that will contain your features.

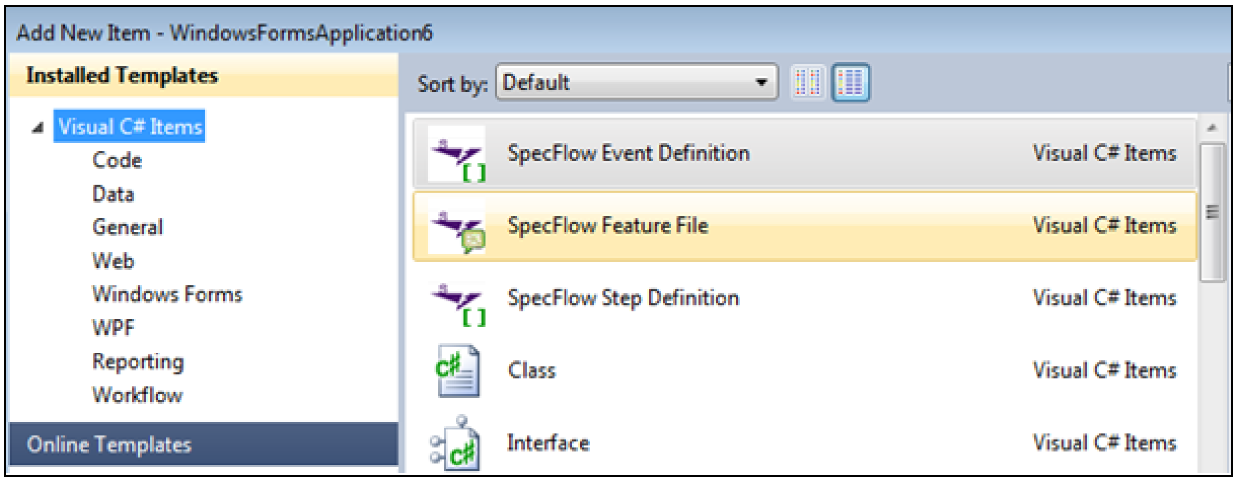

Add New Item - WindowsFormsApplication6
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
SpecFlow Event Definition   Visual C# Items
SpecFlow Feature File   Visual C# Items
SpecFlow Step Definition   Visual C# Items
Class   Visual C# Items
Interface   Visual C# Items

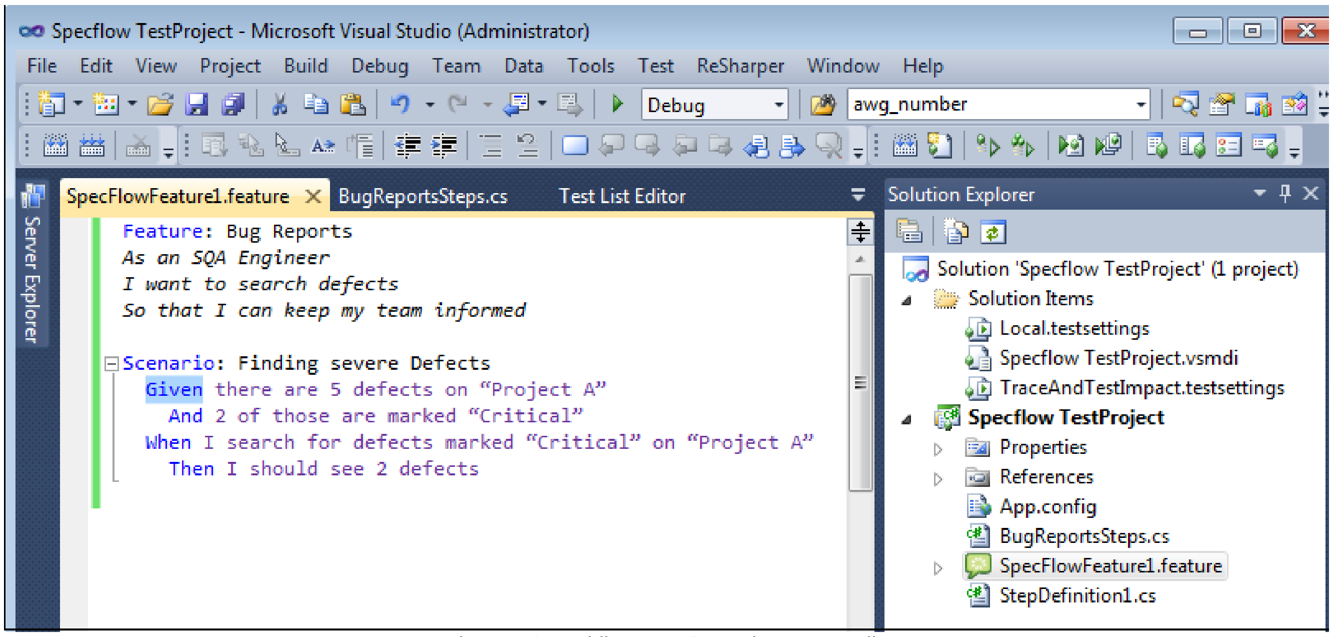
Figure 3.0 – Adding SpecFlow Feature File

3.2.2 Add SpecFlowFeature1

To write the test case, follow these steps:

- Select SpecFlow Feature File in the Installed Templates Folder as shown in the Figure 3.0
- SpecFlowFeature1.feature gets added in the project folder of Visual Studio
- A template automatically gets inserted in the SpecFlowFeature1.feature file.

Select the SpecFlowFeature1.feature file and modify the template to write the test case in BDD Style as shown in the following Figure 4.0.



```
Feature: Bug Reports
As an SQA Engineer
I want to search defects
So that I can keep my team informed

Scenario: Finding severe Defects
  Given there are 5 defects on “Project A”
    And 2 of those are marked “Critical”
  When I search for defects marked “Critical” on “Project A”
    Then I should see 2 defects
```

Figure 4.0 – Adding Test Cases in Feature File

3.2.3 Run Test and Watch It Fail

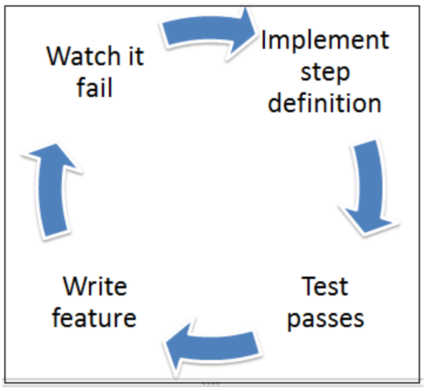
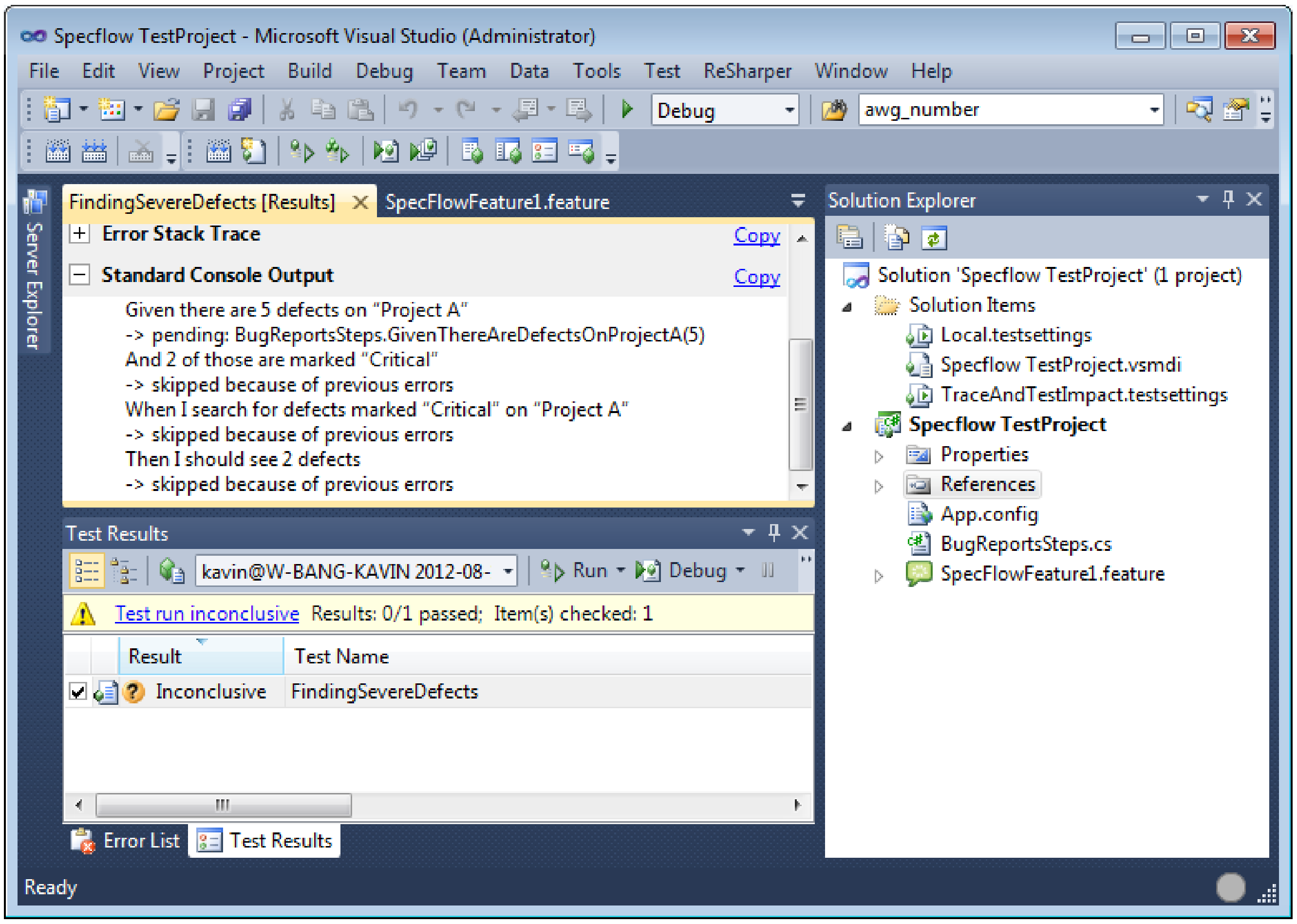


Figure 5.0 – Outside in Development in BDD


Specflow TestProject - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Test ReSharper Window Help
Debug
awg_number
Server Explorer
FindingSevereDefects [Results]
SpecFlowFeature1.feature
Error Stack Trace
Copy
Standard Console Output
Copy

```
Given there are 5 defects on "Project A"
-> pending: BugReportsSteps.GivenThereAreDefectsOnProjectA(5)
And 2 of those are marked "Critical"
-> skipped because of previous errors
When I search for defects marked "Critical" on "Project A"
-> skipped because of previous errors
Then I should see 2 defects
-> skipped because of previous errors
```

Test Results
kavin@W-BANG-KAVIN 2012-08-
Run
Debug

Test run inconclusive Results: 0/1 passed; Item(s) checked: 1

Result	Test Name
Inconclusive	FindingSevereDefects

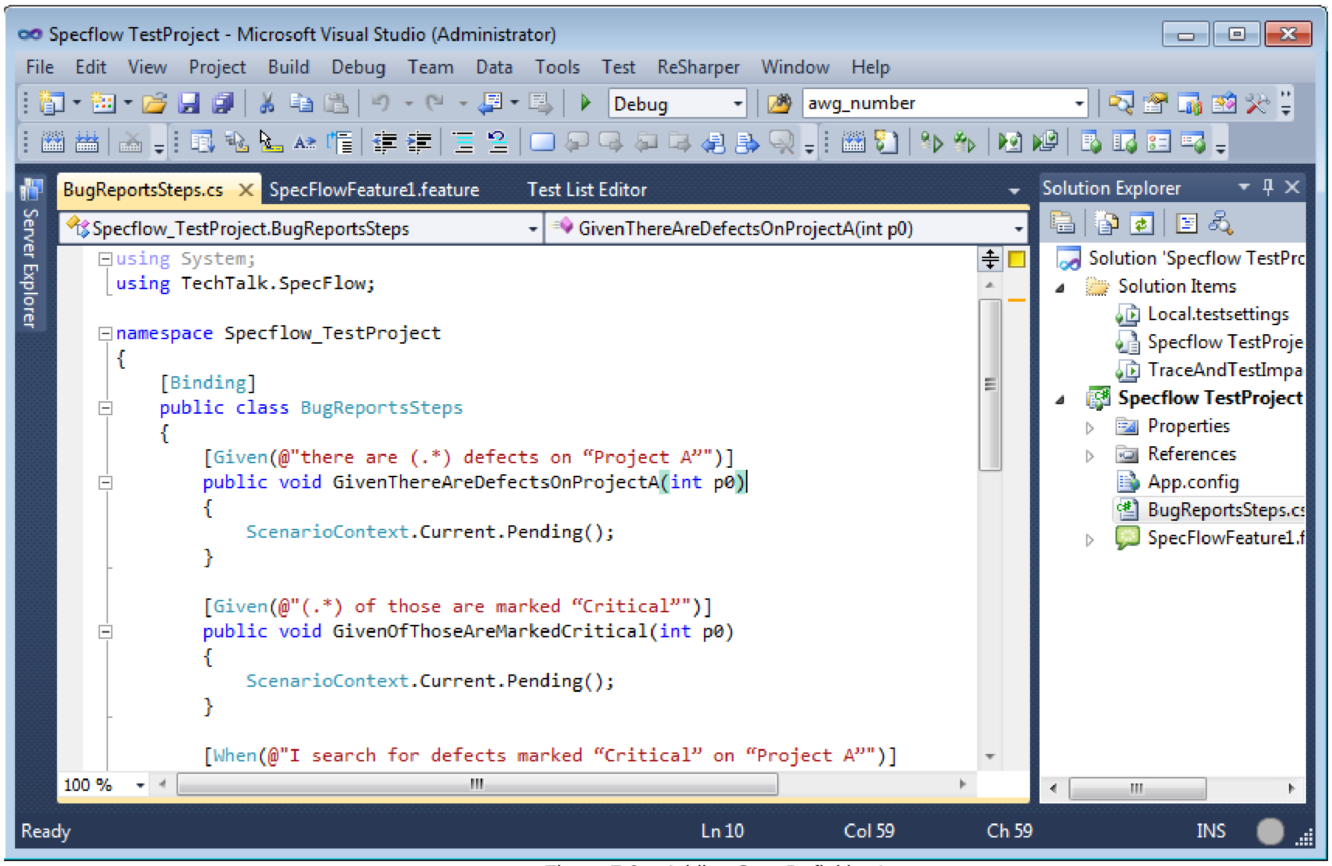
Figure 6.0 – Test Execution Results

3.2.4 Write Step Definitions

- Select SpecFlow StepDefinition file in the Installed Templates Folder
- StepDefinition1.cs gets added in the project folder of Visual Studio
- A template automatically gets inserted in the StepDefinition1.cs file

The next step would be to bind the test cases to test scripts. This would involve writing code for each of the test Steps in StepDefinition1.cs file.

For the example above, the Step Definition for the Given statement would look like the code shown in Figure 7.0



```
using System;
using TechTalk.SpecFlow;

namespace Specflow_TestProject
{
    [Binding]
    public class BugReportsSteps
    {
        [Given(@"there are (.*) defects on “Project A”")]
        public void GivenThereAreDefectsOnProjectA(int p0)
        {
            ScenarioContext.Current.Pending();
        }

        [Given(@"(.*) of those are marked “Critical”")]
        public void GivenOfThoseAreMarkedCritical(int p0)
        {
            ScenarioContext.Current.Pending();
        }

        [When(@"I search for defects marked “Critical” on “Project A”")]
```

Figure 7.0 – Adding Step Definition1

Similarly, each statement in the test case needs to be bound with a step definition.

3.3 Report Format in SpecFlow

SpecFlow has an excellent report format. When you execute the SpecFlow commands from the command line, a neat reporting structure is displayed. Each line of the test case is displayed along with the test script execution time and the pass/fail results. The report format is explained in section-4.2.3

After executing the test scripts, we just need to send the report to the entire team to show the results of automation. The reporting format is very easy to read, and it serves as a useful tool to understand the results starting from test cases.

4 Selecting White for UI Automation

4.1 White is Open Source

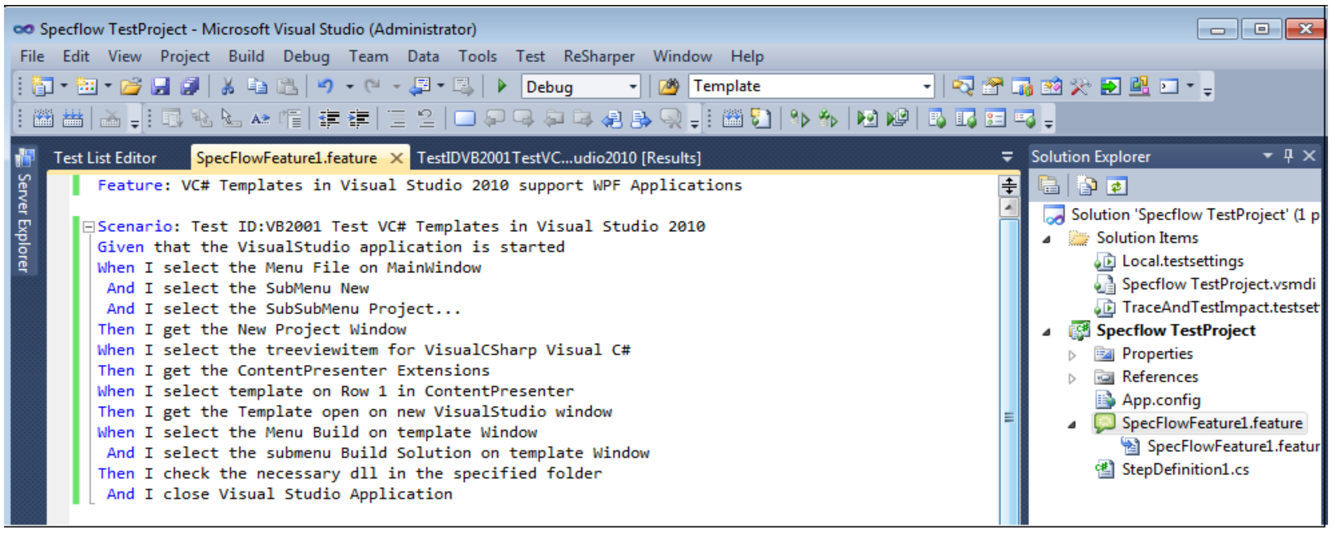
White is another open source tool. It is based on Microsoft's automation API and provides a simple way to automate a GUI developed on Windows.

4.2 White Supports VS2010 Controls

In the Test and Measurement Industry the GUI plays a key role as it is the first item the customers get to interact with. Hence all our applications involve a GUI in some form. The primary goal for us is to ensure that the GUI works as expected in every release of the software. If we can automate even part of the GUI testing, then the work load on test engineers gets reduced.

Selecting the right tool for automation is the key element for success. In one of the applications that we work, we had added a plugin to Visual Studio 2010. The plugin enables customers to insert customized project templates in Visual Basic and Visual C#. Hence automating our plugin involved selecting the various controls in VS2010, like tool bars, menus and tree nodes, for automation. Automating the plugin involved automating Visual Studio 2010 itself. While evaluating the White library from ThoughtWorks, we found that the controls in Visual Studio 2010 are recognized and automated easily. Hence, the White library is the right choice.

Here is an example test case as shown in Figure 8.0 for Automating Visual Studio.

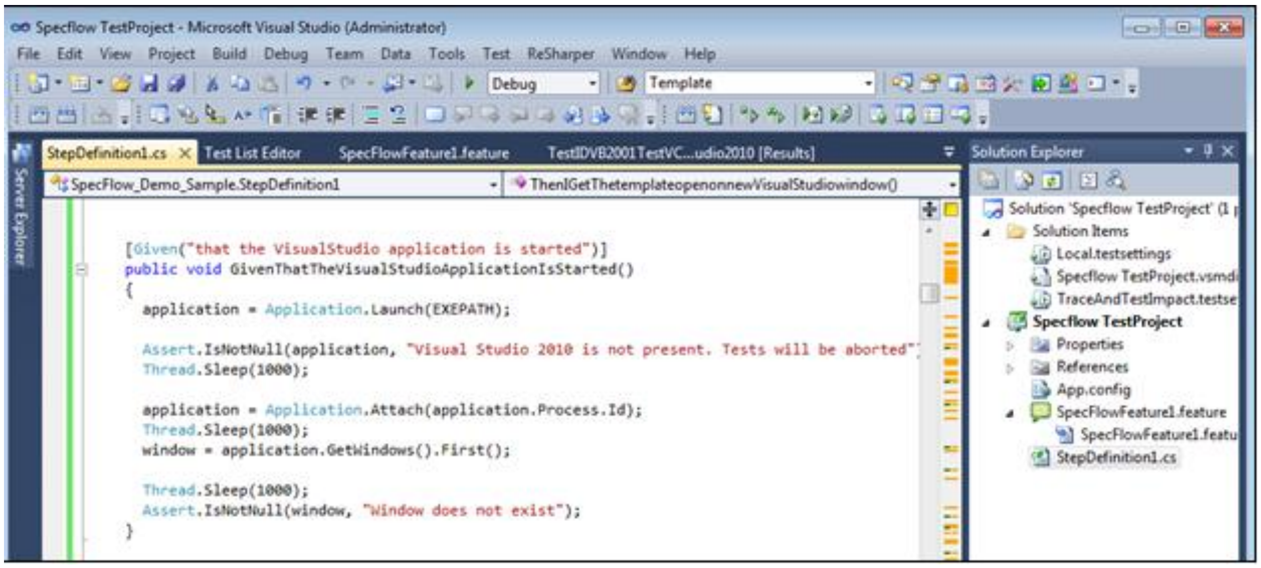

Specflow TestProject - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Test ReSharper Window Help
Debug
Template
Server Explorer
Test List Editor
SpecFlowFeature1.feature
TestIDVB2001TestVC...udio2010 [Results]
Solution Explorer
Solution 'Specflow TestProject' (1 p
Solution Items
Local.testsettings
Specflow TestProject.vsmdi
TraceAndTestImpact.testset
Specflow TestProject
Properties
References
App.config
SpecFlowFeature1.feature
SpecFlowFeature1.featur
StepDefinition1.cs

```
Feature: VC# Templates in Visual Studio 2010 support WPF Applications

Scenario: Test ID:VB2001 Test VC# Templates in Visual Studio 2010
Given that the VisualStudio application is started
When I select the Menu File on MainWindow
 And I select the SubMenu New
 And I select the SubSubMenu Project...
Then I get the New Project Window
When I select the treeviewitem for VisualCSharp Visual C#
Then I get the ContentPresenter Extensions
When I select template on Row 1 in ContentPresenter
Then I get the Template open on new VisualStudio window
When I select the Menu Build on template Window
 And I select the submenu Build Solution on template Window
Then I check the necessary dll in the specified folder
 And I close Visual Studio Application
```

Figure 8.0 – Example Feature file for Visual Studio Automation

4.2.1 Step Definitions using White

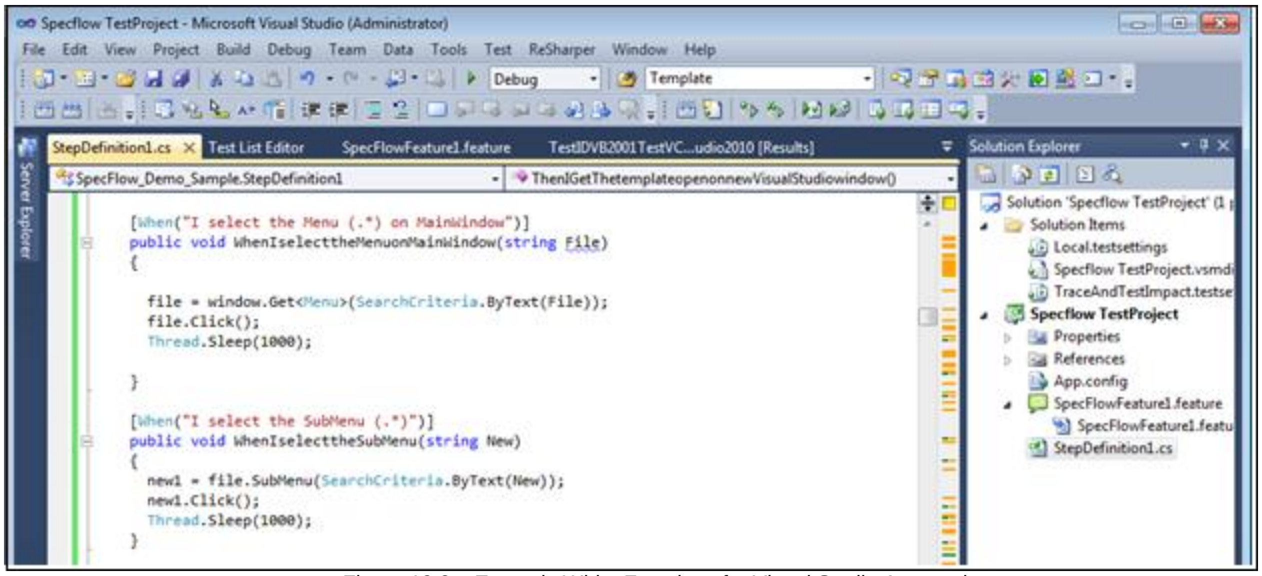
To start the Visual Studio Application, the step definition using White functions would be like the code shown below in Figure 9.0



```
[Given("that the VisualStudio application is started")]
public void GivenThatTheVisualStudioApplicationIsStarted()
{
  application = Application.Launch(EXEPATH);

  Assert.IsNotNull(application, "Visual Studio 2010 is not present. Tests will be aborted"
  Thread.Sleep(1000);

  application = Application.Attach(application.Process.Id);
  Thread.Sleep(1000);
  window = application.GetWindows().First();

  Thread.Sleep(1000);
  Assert.IsNotNull(window, "Window does not exist");
}
```

Figure 9.0 – Example Step Definition file for Visual Studio Automation

To select the File Menu in Visual Studio 2010 IDE, the step definition using White Functions would be like the code shown below in Figure 10.0



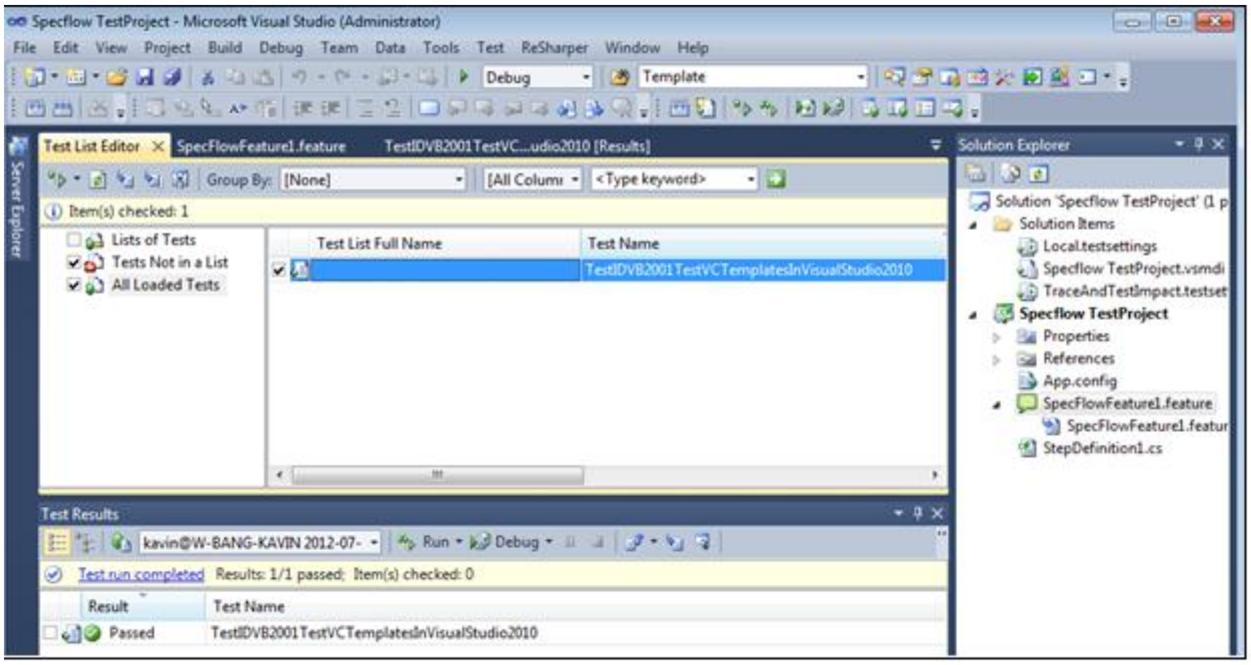
```
[When("I select the Menu (.*) on MainWindow")]
public void WhenIselecttheMenuonMainWindow(string File)
{

  file = window.Get<Menu>(SearchCriteria.ByText(File));
  file.Click();
  Thread.Sleep(1000);

}

[When("I select the SubMenu (.*)")]
public void WhenIselecttheSubMenu(string New)
{
  new1 = file.SubMenu(SearchCriteria.ByText(New));
  new1.Click();
  Thread.Sleep(1000);
}
```

Figure 10.0 – Example White Functions for Visual Studio Automation

4.2.2 Running the Tests in the Visual Studio IDE

To run the tests in Visual Studio IDE itself, we need to create a test list as described in http://msdn.microsoft.com/en-us/library/ms182462.aspx.

Once the test list gets created, run the test by right clicking on the test list name and selecting Run. If everything goes well the Test Results will be displayed at the bottom of the screen as shown below in Figure 11.0



Item(s) checked: 1

Lists of Tests
Tests Not in a List
All Loaded Tests

Test List Full Name	Test Name
	TestIDVB2001TestVCTemplatesInVisualStudio2010

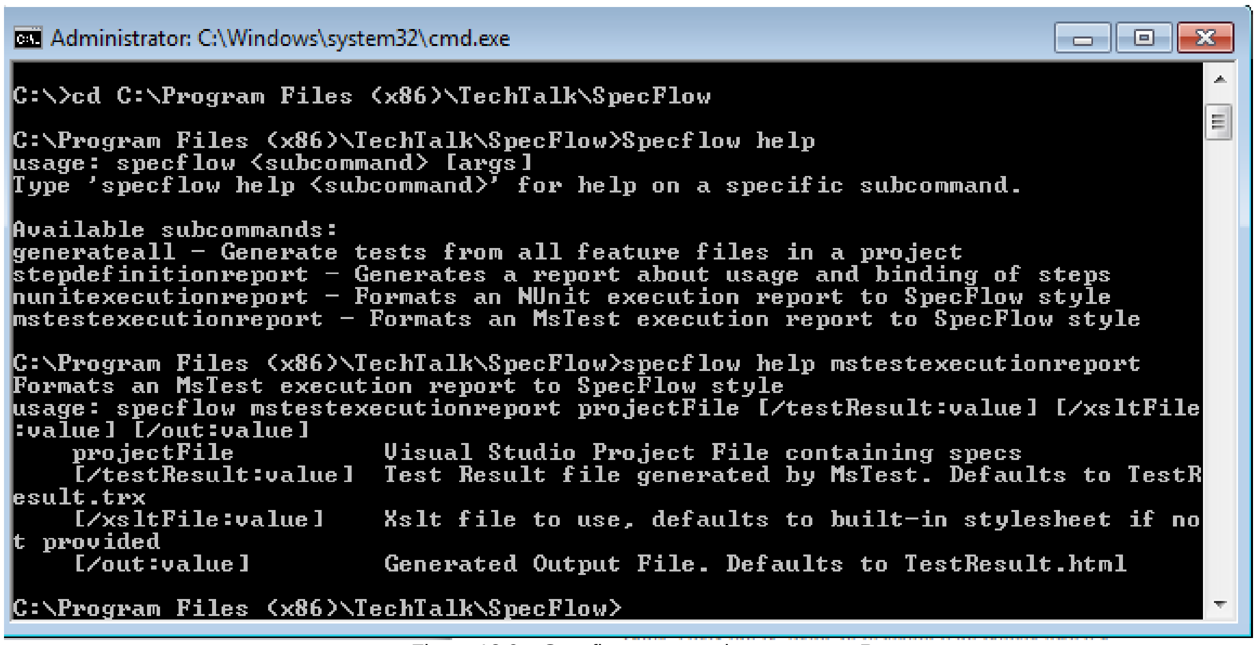
Test run completed Results: 1/1 passed; Item(s) checked: 0

Result	Test Name
Passed	TestIDVB2001TestVCTemplatesInVisualStudio2010

Figure 11.0 – Test Execution in Visual Studio IDE

4.2.3 Report Generation in .html format

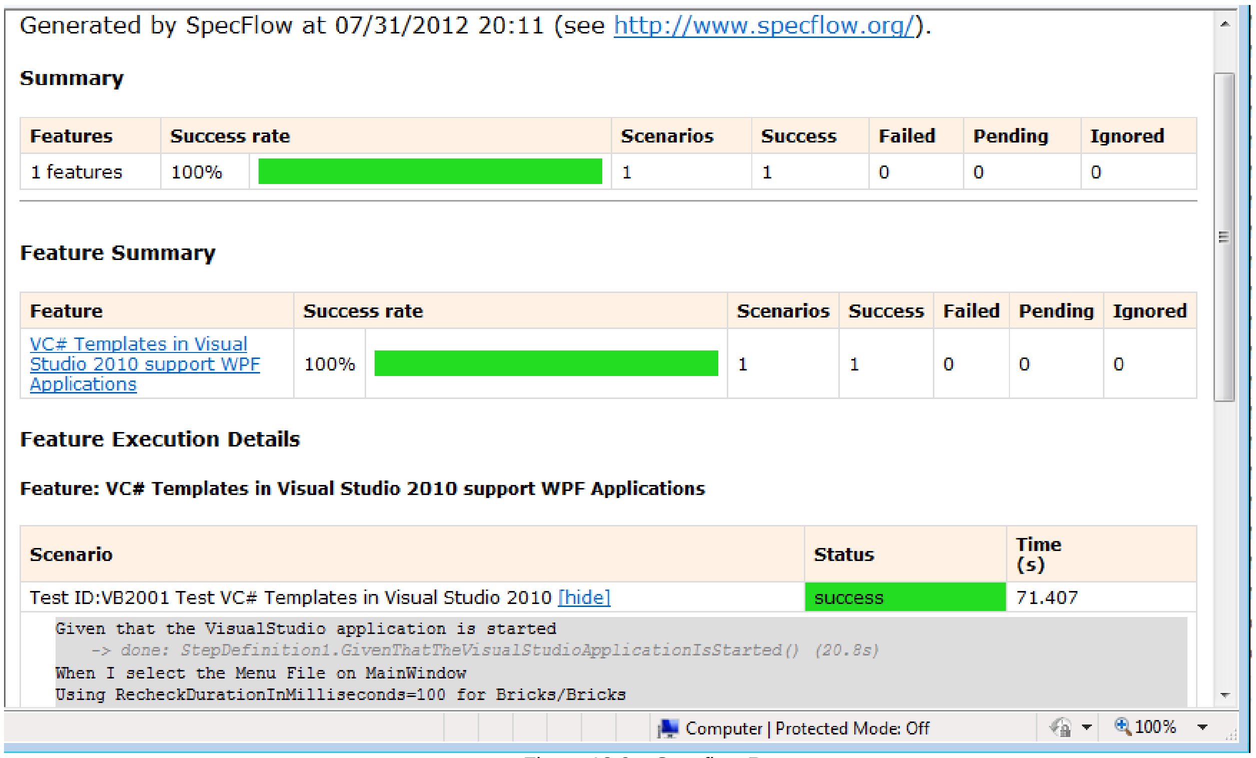
The Test Results output from Visual Studio IDE is a .trx file. If we need an .html report from the .trx file, we need to generate it using the command mstestexecutionreport as shown below in Figure 12.0. This can obviously be automated into build steps after a set of tests have been run.



```
C:\>cd C:\Program Files (x86)\TechTalk\SpecFlow

C:\Program Files (x86)\TechTalk\SpecFlow>Specflow help
usage: specflow <subcommand> [args]
Type 'specflow help <subcommand>' for help on a specific subcommand.

Available subcommands:
generateall - Generate tests from all feature files in a project
stepdefinitionreport - Generates a report about usage and binding of steps
nunitexecutionreport - Formats an NUnit execution report to SpecFlow style
mstestexecutionreport - Formats an MsTest execution report to SpecFlow style

C:\Program Files (x86)\TechTalk\SpecFlow>specflow help mstestexecutionreport
Formats an MsTest execution report to SpecFlow style
usage: specflow mstestexecutionreport projectFile [/testResult:value] [/xsltFile
:value] [/out:value]
    projectFile          Visual Studio Project File containing specs
    [/testResult:value]  Test Result file generated by MsTest. Defaults to TestR
esult.trx
    [/xsltFile:value]    Xslt file to use, defaults to built-in stylesheet if no
t provided
    [/out:value]         Generated Output File. Defaults to TestResult.html

C:\Program Files (x86)\TechTalk\SpecFlow>
```

Figure 12.0 – Specflow commands to generate Report

The test report generated in .html format would look like the below screen shot as shown in Figure 13.0.



Generated by SpecFlow at 07/31/2012 20:11 (see http://www.specflow.org/).

Summary

Features	Success rate		Scenarios	Success	Failed	Pending	Ignored
1 features	100%	🟩	1	1	0	0	0

Feature Summary

Feature	Success rate		Scenarios	Success	Failed	Pending	Ignored
VC# Templates in Visual Studio 2010 support WPF Applications	100%	🟩	1	1	0	0	0

Feature Execution Details

Feature: VC# Templates in Visual Studio 2010 support WPF Applications

Scenario	Status	Time (s)
Test ID:VB2001 Test VC# Templates in Visual Studio 2010 [hide]	success	71.407

```
Given that the VisualStudio application is started
  -> done: StepDefinition1.GivenThatTheVisualStudioApplicationIsStarted() (20.8s)
When I select the Menu File on MainWindow
Using RecheckDurationInMilliseconds=100 for Bricks/Bricks
```

Computer | Protected Mode: Off 100%

Figure 13.0 – Specflow Report

5 Project Example - Time Saved in Automation and Return on Investment (ROI)

5.1 Manual Testing

Before we suggested using SpecFlow and White, only manual testing was planned for the application under test, which in this case is a Visual Studio plugin. The same testing steps were repeated again and again. The estimate for one testing cycle was one week. There were certain areas which could not be tested manually for all the conditions which involved comparing data result values to the accuracy of nine digits! For every release, the team was spending one week's time in testing activities.

Since one week of testing was required to complete one cycle of testing, testing all the functionalities on the daily builds of the software was impossible. The team decided to do just two rounds of testing per month. Due to this, defects could not be caught on daily builds of the application under test.

5.2 Automated Testing

After exploring ways to automate the VS2010 plugin, all the functionalities of the application could be tested in a day. Testing tasks which were taking days to complete got completed in hours' time. Hence, the automation scripts were integrated with the daily builds. The result details were sent as an e-mail attachment to all the stakeholders on daily basis. The entire team was getting updated on the testing details.

5.3 Return on Investment (ROI)

The table below gives the data as to how much time and cost we save after automating the tests using SpecFlow and White.

In all companies, management needs high output with less cost for every project done. By calculating the ROI for automation and projecting the details as given in this table, we are able to demonstrate real benefits we reaped in carrying out automation using open source tools.

5.3.1 Time Saved in Automation

Functionality	Manual	Automation
VS2010 template testing and verification on application under test	**360 min**	**25 min**
Backward compatibility	**240 min**	**25 min**
Multiple template loading and verification on application under test	**480 min**	**25 min**
Data verification testing	**360 min**	**60 min**
Functionality testing	**360 min**	**25 min**
Total Time	**1800 min(30hrs)**	**95 min**

Figure 14.0 – Time saved in automation

Cost of manual testing per annum		
# test cycles per annum	**20**	
# hardware configurations on which test cycles to be executed	**5**	
Manual testing effort per cycle per hardware configuration (person-days)	**5**	
Total manual test effort (person days)	**500**	
Cost of automated testing per annum	**First year**	**Subsequent years**
Tools training / learning (person-days)	**20**	**0**
Test framework & script preparation (person days)	**30**	**0**
Test framework & script maintenance (person days)	**10**	**10**
Execution & analysis of results for 20 cycles & 5 h/w platforms	**100**	**100**
Total automated test effort (person days)	**160**	**110**
Cost savings (considering both automation & open source tool usage)		
Due to effort at $40 / hours ($) & considering 8 working hours per day	**108,800**	**124,800**
Due to open source tool usage in lieu of licensed tool ($)	**10,000**	**10,000**
Total savings ($)	**118,800**	**134,800**

Figure 15.0 – ROI calculation

5.3.2 Some Additional Benefits of Test Automation:

1. Faster execution of the tests and elimination of human errors.
2. Systematic testing process.
3. Maximized test coverage which will assure the quality of the deliverables consistently.
4. Repeatability of test execution.
5. More focus on testing new features.
6. Tests can be re-used on multiple hardware platforms and configuration.
7. Enhanced product quality.
8. Test scripts integration with daily builds as part of continuous integration activity.
9. Intangible benefits, which include developer and tester satisfaction.

6 Conclusion

Selecting the right tool for Agile Testing Methodologies is very important. Every company wants to optimize cost at the same time deliver a quality product. Hence, test engineers need to be very dynamic and explore several options to reduce test cycle time for repeated procedures. Industry standard tools are very expensive and add to cost year on year basis. To optimize cost, we need to use open source tools and show the ROI to the management in test automation.

Acknowledgements

The authors would like to thank Ian Dees who introduced us to BDD and White and to continuously provide support and encouragement to us in all the projects that we have worked. We are also thankful for his comprehensive review of this paper. We would like to thank Richard Vireday for his valuable review comments of this paper.
And to acknowledge the great work of the Open Source teams of SpecFlow and White, especially Jonas Bandi and Gasper Nagy who answered a lot of our questions, which enabled us to set up our test framework successfully.

References

http://www.specflow.org/

http://cukes.info/

http://cukes.info/http://white.codeplex.com/

http://behaviour-driven.org/

http://dannorth.net/introducing-bdd/

https://github.com/techtalk/SpecFlow

http://github.com/techtalk/SpecFlow-Examples

http://github.com/aslakhellesoy/cucumber/wiki

http://www.marcusoft.net/search/label/SpecFlow

www.cognizant.com/insightswhitepapers/open source testing tools: the paradigm shift