Jesse Fulcher

CSCI - 392

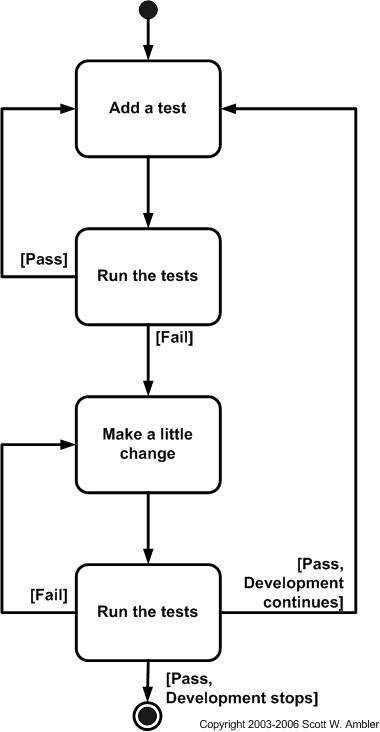
Independent Study

**Week 6 Research Paper**

Testing your software is one of, if not the very most important step of any development project. Weather it be a small assignment for class, or a large scale, fully functional business web application, testing ensures that you’ve got it right. With that, testing can take hours of repetitive use case scenarios, being done by a various number of users to ensure that the product is as promise for the client or end user. Wouldn’t it be nice if there was as way to run thousands of different automated tests overnight with nobody on a single computer? Well, this is what Microsoft can accomplish in their array of automated testing tools built into Microsoft Visual Studio 2013 Ultimate Edition and Test Manager 2013.

This semester I am taking on the task of discovering and learning how to utilize these tools that Microsoft has. So far I have not only been able to watch video’s and read about these tools, but I have already had the opportunity to put them to use in small test applications. In this paper, I will briefly go through the different tools available and give an overview of what I have discovered about them this far.

Before I dive into the different tools available to you in Visual Studio, I want to first describe the idea behind Test Driven Development. TDD is a technique used by developers to ensure the code they are writing is clean, precise and most importantly, correct. In Test Driven Development, the developer writes the methods, classes and code at the same time as their tests, instead of going back and testing their application after completion. With TDD, you are constantly going through cycles very similar to the one graphically drawn out below by Scott Ambler in his blogg, “Introduction to Test Driven Development”.



With Test Driven Development you are constantly going through this cycle of writing code, testing, modifying and repeating that process over and over, instead of writing it all upfront and testing it later. Now let’s take a look at some of the tools available to us from Microsoft that helps us perform Test Driven Development easier.

**Unit Tests**

The first tool that I started my research on was, Unit Testing. The idea behind Unit Testing is simple. Its all about breaking your production code down into individual pieces so that you can test small, individual chunks of your code against hardcoded test values. You break your code down into ‘units’, usually methods, to test repeatedly, ensuring that the small broken down methods of your code are executing correctly without you even having to be at your desk.

Building a Unit Test is even simpler than the concept of one. To begin, you need to create a new Project in Visual Studio and select Test under the Installed Templates. It is important to note that this tool is only available in the Ultimate Edition of Visual Studio, so without that you will not have access to Unit Testing abilities. Once you have created your new Unit Test Project you’ll need to reference your current project that you want to test. As soon as that is done you are ready to write your first Unit Test.

An instructional video I watched called, “Getting Started with Unit Testing”, made a recommendation to follow the AAA Rule. The AAA Rule is: Arrange, Act and Assert. Arrange means to set up your environment and gather the information you want to test in a method that will perform the way you need it to perform. For example, if you have a calculator application, and you want to make sure your variables are being calculated properly, you need to set up. Arrange that environment so that your tests will cover what you need it to cover. After you have Arranged, you need to Act, this is actually running of the test. Built into this testing tool is the ability to right click and run the test at any time and see the results of that test, which brings us to our final A, Assert. After the test has been run, its time to see if anything has failed and why. These test will hopefully help point you in the direction of where you need to make changes in your application.

The final recommendation of Unit Testing is to begin by writing the test to fail. Why would you do this? Because you want to make sure your test is working, and not just passing. Since this test will hopefully be used over and over, thousands of times to keep your application running smoothly, you need to ensure that you are getting the most out of your unit tests and ensure that they are working correctly. You really need to tests your tests to ensure your Unit Tests are working properly, otherwise there is no reason to write them.

**Coded UI Test**

Having first hand experience with the manual testing process, I feel like Coded UI tests could be super valuable to any company that needs to perform pre release testing. It has been my experience in web development that before you publish any changes to your online application you first need extensive UI Testing. You want that experience to be the same or better for the end user, not different and confusing. In order to ensure controls are working properly you typically have a team of developers and/or testers trying to break your system. They will come up with as many odd scenarios and use cases as possible in hopes of tricking the system into allow them to do something or see something they aren’t supposed to. The process can be tedious and is very prone to user error. In most applications there are thousands of different cases that could all be tested and depending on a team of testers to hit every single scenario every single time is almost too much to ask for.

With automated UI Testing in Visual Studio, you can create Test Methods by recording your actions and events. For example, if you have an online web store application, you could use UI Testing to record the click events it required to take you to that online store. Then you could try typing in values you know shouldn’t be accepted (a or -1), followed by click events that should not be allowed or drop downs that should open a certain direction. Anything that you would normally test, you can do and record the actions. When you're done, you click save, and Visual Studio will build a method of click events and other various control actions that can be run over and over versus having to manually click those events each and every time.

A large part of testing is side effects. Side effects occur when developers put out what they believe to be a fix to one problem, but it causes a variety of other problems, sometimes even unrelated to original problem, possibly in an entirely different part of the application. Because of these side effects it is crucial that whenever an update or fix is made, the testing team goes in and retests ALL scenarios, in case of side effects. With automated UI Testing, you could have a set of recorded tests done that you know you need to run every time. Once a developer has put out a fix and they are ready to test again, you run those UI tests again, knocking out that many more cases and helping eliminate errors.

**Web Performance and Load Test**

Web performance and load testing is exactly what it sounds like. Visual studio has given us the tools to be able to simulate what it would be like if we had hundreds or even thousands of users on our web application at one time. Very similar to the UI Tests, the IDE gives you the ability to record an action such as logging in, or performing a particular act on your web app. After you have this action recorded you can then simulate this action over and over while not being anywhere close to your development computer.

In the video I watched, “Load Testing”, it was reccomended by Charles Sterling to run a more realistic scenario. Instead of simulating a user with the same name, Joe Smith for example, you should run the test with all the names in the Phone Book. This simulates different users attempting to log in, giving you better results. The whole point of the Load Testing tool is to simulate a real world scenario and see how your site holds up under the pressure of a massive amount of users.

Overall in this first couple of week of this semester I have already learned a tremendous amount about Automated testing. I look forward to putting my newly acquired knowledge to work when I build my very own ASP.NET Online Web applicaiton. I believe that I now have the knowledge and skill set needed to begin building an application with the Test Driven Development model in mind by designing my application with Unit Tests and Automated UI Testing in my toolbelt.

**References**

“Automated UI testing with Visual Studio 2012”, Channel 9, Brian Keller, May 17 2013 <http://channel9.msdn.com/events/Visual-Studio/Visual-Studio-2012-Virtual-Launch/Automated-UI-testing-with-Visual-Studio-2012>

“Getting Started with Unit Testing Part 1”, Channel 9, Robert Green, Jun 04 2014

<http://channel9.msdn.com/Shows/Visual-Studio-Toolbox/Getting-Started-with-Unit-Testing-Part-1>

“Getting Started with Unit Testing Part 2”, Channel 9, Robert Green, Jun 11 2014

<http://channel9.msdn.com/Shows/Visual-Studio-Toolbox/Getting-Started-with-Unit-Testing-Part-2>

“Getting Started with Unit Testing Part 3”, Channel 9, Robert Green, Jun 18 2014

<http://channel9.msdn.com/Shows/Visual-Studio-Toolbox/Getting-Started-with-Unit-Testing-Part-3>

“Guidelines for Test-Driven Development”, Microsoft Development Network, Jeffrey Palermo, [http://msdn.microsoft.com/en-us/library/aa730844%28v=vs.80%29.aspx#feedbac](http://msdn.microsoft.com/en-us/library/aa730844%28v=vs.80%29.aspx#feedback)

“Introduction to Test-Drive Development”, Agile Data, Scott Ambler, <http://www.agiledata.org/essays/tdd.html>

“Load Testing, Channel 9, Charles Sterling and Robert Green, Jul 23 2014

<http://channel9.msdn.com/Shows/Visual-Studio-Toolbox/Load-Testing>