Group Report
Unit Test Visualisation

Team: Group 4
Course: ELEN7046 - Software Technologies and Techniques
Date Submitted: 25 June 2012
Source & Documentation:
https://github.com/KeaganPhillips/Wit-Group-4-project

Abstract

The design and implementation of a software visualisation tool for unit testing is presented. The visualisation tool is to be used to graphically display, enhance and ... unit tests. The development of both the unit tests and its visualisation are discussed. The tool was implemented using TDD and uses C# Reflection to obtain the necessary information which was then sent to the visualiser to display. The implementation suggests that the tool can play an important role in software visualistaion education.

Contents

1 Introduction

This paper presents a report of an application that links software visualisation and software education. In the literature, it is stipulated that software visualization encompasses the development and evaluation of methods for graphically representing different aspects of software, including its structure, its execution, and its evolution.

The aspect of software that we present in this report is Unit Testing[1]. The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as expected. Each unit is tested separately before integrating them into modules to test the interfaces between modules. This testing mode is a component of Extreme Programming (XP)[2], a pragmatic method of software development that takes a precise approach to building a product by means of continual testing and revision.

Unit testing involves only those characteristics that are vital to the performance of the unit under test. This encourages developers to modify the source code without immediate concerns about how such changes might affect the functioning of other units or the program as a whole. Once all of the units in a program have been found to be working in the most efficient and error-free manner possible, larger components of the program can be evaluated by means of integration testing.

In this report, we primarily demonstrate an application that visualises all the unit tests in a given program. The application discussed in this report visualises not only the unit tests, but also the class diagrams together with their public methods and properties.

The remainder of this report is as follows. Section 2 discusses the licenses used in the development of the application. We provide the problem description in section 3, and discuss the approach we took to solve the problem in section 4. Installation process for the application is discussed in section 5. We provide a discussion on how the approach to the solution was implemented in section 6. In section 7, we discuss an analysis of our solution, and provide future works in section 8. We end the report with a conclusion

2 License Agreement

2.1 Overview

An analysis of software licenses was done considering the combination of both original work and third party components. The new BSD license also known as the 3-Clause BSD[3] license was chosen. The software and source code are made available under this license. Use, modification and redistribution is not restricted.

2.2 Components used and their license agreements

The following lists the components used within the system together with their respective license agreements.

- **Coffee Script**: MIT license[5]
- **Java script:** GNU GPL[6]
- **JQuery:** GNU GPL[6]
- **Kinetic.js:** GNU GPL[6]
- **PDF reporting tool:** CPOL[7]
- **ITextSharp:** GNU GPL[6]

For the complete reference, please refer to the document 'ELEN7046 - Group4 - Licence Agreement.pdf'[4]

3 Problem Description

3.1 Overview

As good software practitioners, we always aspire to adopt the current best practices in our industry. In recent years the so called 'Agile' methodologies has become very popular with TDD[8] (Test Driven Development) begin a very widely adopted agile practice. Today it is almost expected of developers to not only write program code, but also create automated unit tests and apply other TDD practices.

3.2 Problems with respect to Unit Tests

Writing automated unit tests comes with its own unique problems. Our project attempts to address some of these shortcomings by proposing a framework against which developers can write their automated tests. The end result should give developers a broad and high level view of all unit tests, making it easier for them to get a holistic view of all the unit tests in the system under test.

3.2.1 Large Complex projects

Tests can become difficult to read and understand especially as the suite of tests rapidly grows in number and complexity with each build deployed. It is very easy for a developer to get lost in the detail by reading the actual automated test code. This is because the developer can only view one automated unit test at a time, making it difficult to see the whole picture.

3.2.2 Comments in tests

Many developers don't write adequate comments in their tests leaving other team members in the dark with regard to what their initial intent was.

It's never good to have a test or group of tests that isn't well understood (even though the test(s) may pass). Tests are only useful when the context of the tests is clear and well understood.

3.2.3 The Learning Curve

The above described scenarios make it difficult for the development team to evolve a system over time. The result being that the learning curve is often very steep for new developers joining the development team as well as for junior developers on the team.

3.3 Proposed Solution

What is needed is a framework that provides a convention that developers can follow where tests are written in a uniformed way. The system must provide a mechanism to visually display all tests without the developer having to zoom into the code to look at tests one at a time.

This, we believe should add tremendous value with respect to the learning of an existing and evolving code base.

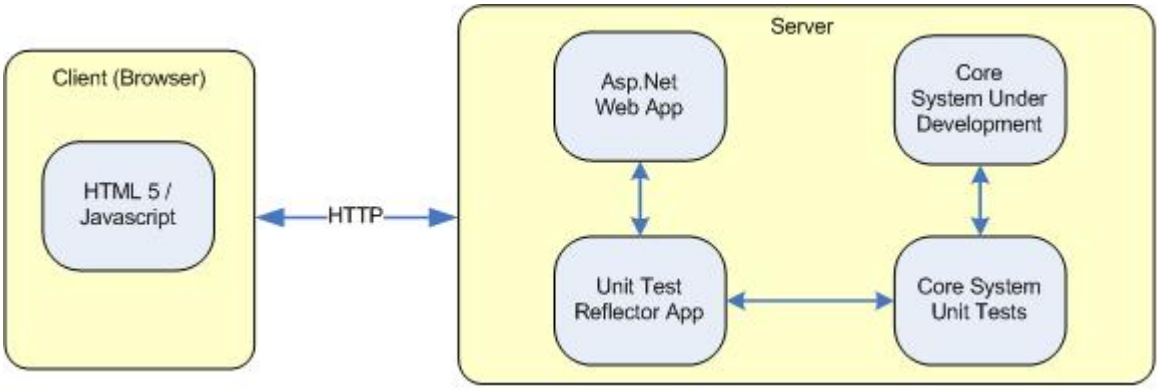

Figure 1:

4 Approach

Given the above stated requirement, the team proposed the following solution. A web application will be developed that provides a visual representation of the unit tests of the system under development

4.1 High level design overview

Figure 1 depicts a high level overview of the proposed solution. This section will briefly discuss the key components of the system from right to left.

4.1.1 Core System Under Development

This component represents the actual core application. This is the system that will actually run in the production environment and the application against which the unit tests will run. This system is completely oblivious of the other components depicted. Obviously developers must use an "interface driven" approach upon developing this component to facilitate modularity and enable for easy unit testing. The actual functional requirements for this application is beyond the scope of the project.

Please see section 3.1 of Keagan Phillips's individual report[9] for more a more in depth discussion regarding this component.

4.1.2 Core System Unit Tests

This component represents the actual unit test project and contains automated unit tests. These tests target the "Core System Under Development" component.

Tests must be written following a predefined convention[10]. The development team used the TDD[8] (Test Driven Development) technique and "Given / When / Then"[11] convention for specifying tests scenarios.

Please refer to section 3.2 of Keagan Phillips's individual report[9] for more detail on this component.

Please see the "Unit Test coding convention" document for an in depth discussion on how to wire automated tests using the provided framework.[10]

4.1.3 Unit Test Reflector

This component's responsibility is to extract the unit test metadata embedded in the "Core System Unit Tests" component and transform that data into a C# data structure. The team made use of the Microsoft Dot Net Framework's Reflection[12] feature to achieve this goal. This data can then be collated and presented to the user in almost any format including document format (i.e. HTML page or PDF) or in our case visually and in pdf format as shown later on.

Please refer to section 3.3 of Keagan Phillips's individual report[9] for more detail on this component.

4.1.4 Asp.Net Web Application

This component acts as a shell to host our solution in a web based environment. It calls the component 'Unit Test Reflector Application' upon receiving a web request. It then retrieves a data structure with the relevant test metadata. The data is then converted from a dot Net C# object, to a JSON[13] data structure to be transmitted back to the browser as an HTTP response.

See Borellis report for more information.[14]

5 Installation Procedure

blah.. blah...

References

[1] http://www.extremeprogramming.org/rules/unittests.html

[2] http://www.extremeprogramming.org/

[3] http://www.opensource.org/licenses/BSD-3-Clause

[4] For the Group Project License Agreement see:

https://github.com/KeaganPhillips/Wit-Group-4-project/tree/master/Documentation/

[5] http://www.opensource.org/licenses/mit-license.html

[6] http://www.gnu.org/copyleft/gpl.html

[7] http://www.codeproject.com/info/cpol10.aspx

[8] http://www.agiledata.org/essays/tdd.html

[9] For Keagan Phillips's Individual Report see:

https://github.com/KeaganPhillips/Wit-Group-4-project/blob/8fe27d49f80fb3be2ed421e3bf7ce8daff002794/Documentation/Individual%20Reports/Individual%20Report%20-%20Keagan%20Phillips.pdf?raw=true

[10] For the Unit Test Coding Convention see:

https://github.com/KeaganPhillips/Wit-Group-4-project/tree/master/Documentation/

[11] Structuring your test using Given/When/Then

http://testedobjects.sourceforge.net/m2-site/main/documentation/docbkx/html/user-guide/ch03s04.html

[12] http://msdn.microsoft.com/en-us/library/ms173183%28v=vs.80%29.aspx

[13] JSON (JavaScript Object Notation)

http://www.json.org/

[14] For Boreli's Individual Report see:

https://github.com/KeaganPhillips/Wit-Group-4-project/tree/master/Documentation/