From: *Jake Orben*

To: *Professor Pulimood*

Course Number / Course Title: CSC 415 / Software Engineering
Date of Submission: 9/20/2013
Instructor: Dr. Pulimood

Subject: Software Testing Variations

This is a memo about different types of testing, as well as the importance of testing.

1. Introduction

Software testing is a critical part of all software development. Because it provides information and statistics on the quality of a software solution. A tester does several things when testing software. Checking for bugs is among the most important things that a software tester deals with. Checking for errors that affect the functionality of the program in an unintended or negative way is important for software usability. The software is also being looked at to ensure that it is what the customer asked for; making sure that the software fulfills all of the functionality requested. That being said, testing can never cover every possible error in a large program due to the fact that there are so many variables, solving one problem can create another. Additionally there is often too much to check for, in a large program there can be thousands of components, each with a thousand different ways of entering data incorrectly to attempt to crash the program, that adds up to millions of possibilities [Xiangyun 2011].

The goal of this paper is to cover a basic understanding of testing and how it is executed, as well as a variety of ways that testing is performed and as several tools that can be used for testing a system. By the end of this paper the reader should have an understanding of the importance of testing, as well methods used to test software. This memo will also contain a recommendation for the company as a whole.

2.0 Types of Testing

When testing, “test cases” are often made. These test cases cover the functionality that is to be tested. Test scrips explicitly state what is being tested and how it is to be tested, if, for example, a tester is to try to log into a system, the script should provide both a working user ID and password, a non-working us ID and password as well as a user ID that does not exist. This ensures that all of the the functionality is covered. This ensures that the functionality being turned over to a customer is up to the quality that it should be before being distributed. One popular approach to testing is the box method, which has three variants, black-box, grey-box and white-box testing.

2.1.0 Unit Testing

Unit testing involves tests that verify the functionality of a specific piece of code at the class level of a program. Unit tests are often written by developers who worked on creating the code

so that they can be sure that the function that was written works as it is supposed to. Despite the fact that these tests are done, the software as a whole is not guaranteed to work. This would be similar to testing the lumber used to build a house, each piece might be solid, however, if the building is not put together properly, the entire thing will collapse. The reason that unit testing is important is that it can save a great deal of time and money. If a program is assembled without unit testing, and there is an error at the class level , each individual segment would have to be tested to find the error, which could cost a considerable amount of man hours. Additionally, fixing this error could cause errors in the rest of the program, meaning that a great deal of the design work would have to be done over. By contrast, if all of the segments were tested in advance, when assembled, there would not have been an issue and nothing would have to be re-designed. Almost all unit testing is done using white-box testing, which will be discussed later in this memo, due to the fact that there is not a way for a standard user to interact with the system to allow for black-box testing, which will also be covered later in this memo [IEEE Standard for Software Unit Testing 1986].

2.2.0 Integration Testing

Integration testing involves putting one or more units together to be sure that they work together as a cohesive part of a larger program. For instance, if one was to pre-assemble the walls of a house with the aforementioned lumber and test these walls to be sure that they are structurally sound, that would be the theory behind integration testing. Such testing is critical because it ensures that the basic blocks of a program work well together and do not cause any errors. This is a recursive process as two or more integrated pieces can be put together to be tested as a whole, thus creating a new integrated test. This is also important if one would like to save money, as every time a set of blocks is ready to be put together it can be tested, so that when the entire program is complete, there are fewer error that are found, making software easier and faster to deploy. A majority of integration testing is done using the white-box method, as there can often be no interface, making black-box testing impossible [Linnenkugel 1990].

2.3.0 System Testing

System testing is testing that is done to a completed software system. Such testing is perhaps the most critical, as it is what the customer sees. If software is not satisfactory, then the customer could cut funding or become irate with the business. There are several ways in which this testing can be done [Qing Zhou 2011]:

2.3.1 Black-Box Testing

In the black-box approach, the tester does not know how the software works internally, they only know what the software is supposed to do and only have access to a system from the front end of the software. This means that the user will not enter data in a format that they know works, as they do not know what is occurring on the back end of the software. Here are a couple of the more common forms of black-box testing that are done [Ning Li 2010].

2.2.1.1 Fuzz Testing

Fuzz testing is a sub category of black box testing. Using this method, invalid data is entered into a system in an attempt to cause it to crash. An example of how this could work would be if the system being tested relied heavily on SQL. In SQL, the ' is used to signify the beginning and end of a string, if one is used in a user ID and not properly dealt with, it could cause a crash or an error [Dingning Yang 2012].

2.3.1.2 Use Case Testing

Use case testing is often one of the first tests done. In use case testing, a user is generally tested against their role in the system. Such tests ensure that an administrator has all of the rights that they would need to run the system. Furthermore the role of a lower level user is tested as well, to makes sure that they do not have the rights to do something in the system that could potentially cause a problem. For instance, in Canvas a student should not be able to delete a professor's comments, or assignments, as such, there was most likely a test to make sure that this could not happen [Rajappa 2008].

2.3.1.3 State Transition Table

A state transition table is a Finite State Machine (a theoretical machine that exists and transitions through several states, depending on the input the machine is given) table describing the current state of the software, as well as all possible states that it can enter. If the machine (software) starts in state zero or S0 it can then enter S1 if it is given a certain value. This test maps all of the functionality of the software and ensures that all of the functions that the system can perform are able to be accessed successfully. For example, if Canvas had the ability to grade homework by itself, but there was no way to access this section of the software, nobody would be able to use the software properly[IEEE Standard Glossary of Software Engineering Terminology 1990].

2.3.2 White-Box Testing

White-box testing involves knowledge of the internal workings of a system and is often performed by programmers. White-box testing does not always use test cases, as it is up to the tester to decide the way in which each segment of the program should be tested. The tester takes each of the inputs that they want to check and verifies the output from each. Due to the fact that this testing is often done at the code level, there are several places that it can be used on. First is on unit testing, which consists of several modules and is the bare minimum for a function to work, this is considered the lowest level of the system. Testing can also be done on the integration level, which is several units or modules together as one cohesive unit. Finally testing can be done from the system level, the highest level, the system level is when all of the modules are put together into a full program. There are several common forms of white-box testing, where are two [Nita 2009].

2.3.3 Grey-Box Testing

2.3.3 Grey Box Testing

Grey-box testing is a mix of black-box testing as well as white-box testing. It involves knowing about the code used to design the system, like a white-box tester, but testing at the user or black-box level. This would mean that the tester knows the format that data is supposed to be entered into the system. The advantage of this being the tester could intentionally enter data in the wrong format to try and cause an error. For example, the User ID and Password to log into a system might not be secure if the system is using a SQL table, the tester could test by using SQL

injection. This would, if done correctly, allow a user to log in without authorization and is therefore a highly important test [Jianqiang Zhuo 1997].

3. Tools

There are many tools that can be used to help speed up testing. Some of these tools cost money, however, they are worth this money because they ensure a better product for the customer, improving the company's reputation, which could lead to bigger and better business.

3.1 Sauce Labs

One useful tool available for testing web browser based applications is Sauce Labs automated testing. This service relies on Selenium Builder (made by the same developers), which takes a set of prerecorded instructions and allows the instructions to be tested on a variety of different browsers and operating systems remotely. This is quite useful because it means that only one script has to be made. Additionally, once a script is made the execution is automated, the tester only needs to press start, the software takes a video as well as screen shots to attach to the test script, which many businesses require during testing to prove that the tests were done properly. Furthermore, once a script is made, small variables such as names and passwords can be changed either by hand or via a shell script, which will allow for Fuzz Testing to be easily executed [Sauce Labs Website].

3.2 Eclipse Built in Debugger

Another terrific tool is the built in debugger in Eclipse. This tool allow for a program to be looked at step by step, without missing any information. The debugger allows the user to put down stop points for a program to check the values of variables and edit them if the user desires. This allows for a programmer to check their unit of a program quite easily, additionally, this tool is free with eclipse, making it cost effective.

4. Company Recommendations

For our company, due to the fact that it is so large and thus can afford the expense, I would recommend doing testing at all three levels, unit, integration and system testing. Furthermore, the software that we develop is quite large in scale, increasing complexity. Such increased complexity leave more room for errors. Owing to the fact that it is more time efficient, if we were to skip unit testing, an error could be seen in integration testing. This circumstance would cause the company to go back and do the unit testing, which would be inefficient. Additionally, Fuzz Testing, Use Case Testing, API Testing and Fault injection should be used to verify the integrity of the system. Fuzz Testing is good for ensuring that the system can handle any input, Use Case Testing is useful for making sure that a system is secure. API testing is a white-box test and is used to make sure that the software's inputs and outputs all communicate properly [YoungHan Choi 2008]. Fault Injection ensures that strange input can be dealt with at a low level, helping to prevent unwanted crashes [Fanping Zeng 2009]. These tests should be done to ensure that the customer receives a quality product and uses us in the future for their software endeavors. I would also highly recommend that Sauce Labs is used to check the HTML section of our product, due to its cost effectiveness and speed. Using the built in debugger from Eclipse is not necessary, however, it could speed up unit testing considerably if used.

5. Conclusion

In conclusion, it is critical that our product be tested in a variety of ways, if it is not, a customer could receive a sub-par product or development could go over budget, making the entire endeavor a waste of time, money and resources. Additionally, no section of testing should be

over looked, if unit testing is skipped, it could cause a huge time lag with integration testing, putting the company behind schedule. Furthermore, the company should not overlook the tools at our disposal. Despite the fact that some tools cost money, they can save a great deal of time and effort in the long run and help to assure a better product for the customer.

6. Bibliography

Dingning Yang; Yuqing Zhang; Qixu Liu, "BlendFuzz: A Model-Based Framework for Fuzz Testing Programs with Grammatical Inputs," Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on , vol., no., pp.1070,1076, 25-27 June 2012

Fanping Zeng; Juan Li; Ling Li; Xufa Wang, "Fault Injection Technology for Software Vulnerability Testing Based on Xen," Software Engineering, 2009. WCSE '09. WRI World Congress on , vol.4, no., pp.206,210, 19-21 May 2009

http://en.wikipedia.org/wiki/Software_testing

IEEE Standard for Software Unit Testing," ANSI/IEEE Std 1008-1987 , vol., no., pp.0_1,, 1986

IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries," IEEE Std 610 , vol., no., pp.1,, 1991

IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990* , vol., no., pp.1,84, Dec. 31 1990

Jianqiang Zhuo; Oman, P.; Pichai, R.; Sahni, S., "Using relative complexity to allocate resources in gray-box testing of object-oriented code," *Software Metrics Symposium, 1997. Proceedings., Fourth International* , vol., no., pp.74,81, 5-7 Nov 1997

Linnenkugel, U.; Mullerburg, M., "Test data selection criteria for (software) integration testing," Systems Integration, 1990. Systems Integration '90., Proceedings of the First International Conference on , vol., no., pp.709,717, 23-26 Apr 1990

Ning Li; Zhanhuai Li; Xiling Sun, "Classification of Software Defect Detected by Black-Box Testing: An Empirical Study," Software Engineering (WCSE), 2010 Second World Congress on , vol.2, no., pp.234,240, 19-20 Dec. 2010

Nita, M.; Notkin, D., "White-box approaches for improved testing and analysis of configurable software systems," Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on , vol., no., pp.307,310, 16-24 May 2009

Qing Zhou; Bin Liu; Zhengwei Yu; Xiaoqi Xing, "The process of requirement analysis about military software system testing," Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference on , vol., no., pp.645,650, 12-15 June 2011

Rajappa, V.; Biradar, A.; Panda, S., "Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory," Emerging Trends in Engineering and Technology, 2008. ICETET '08. First International Conference on , vol., no., pp.298,303, 16-18 July 2008

www.saucelabs.com

YoungHan Choi; Hyoungchun Kim; HyungGeun Oh; Dohoon Lee, "Call-Flow Aware API Fuzz Testing for Security of Windows Systems," Computational Sciences and Its Applications, 2008. ICCSA '08. International Conference on , vol., no., pp.19,25, June 30 2008-July 3 2008