Kyle Theriault
Software Testing
Professor Fluet

Independent Study Summary

Over the course of this quarter, I have learned a great deal about software testing and computer science as a whole. The goal of the Software Testing Independent Study was to focus in on different testing topics and learn to implement these ideas. These topics started with unit testing, then risk/requirements testing, regression testing, load/stress testing and finally database testing. Each of these topics were chosen because they seemingly are very different topics in the software testing domain. Through the course of the quarter I actually found that instead of being completely different most of these testing topics were very similar and in a lot of cases the topics overlapped with each other. For each topic, I spent one week doing research and writing a paper and then the next week creating an implementation of a major concept from the topic. Both the papers and implementations are attached to this summary.

The first topic I covered was Unit Testing. I started with this topic because it was the seemingly the most simple testing concept. After doing research on the topic I found this to be true but also found that Unit Testing is actually the most time consuming and tedious testing process of all. Unit Testing requires the tester to check every small segment of code for defects. The goal is to do this testing on these small segments of code, or units, and find defects before the code has reached its later stages and it is harder to find the defect. By focusing on small areas of code to test we are reducing the risk of defects. All the research that I found on Unit Testing pointed out that because Unit Testing focuses on smaller regions, it is the most efficient at finding defects. The downside of Unit Testing is that it is a very long and tedious process where many of the tests repeat or are very similar. This property causes testers or coders to get lazy when they are coding and forget or ignore unit testing all together. Since Unit Testing is not very time efficient it also makes it harder on smaller businesses to fully unit test their code as they do not have the resources and money to give to testers and developers to unit test the code as they go. Overall Unit Testing is very efficient at finding defects but often is overlooked or ignored because of programmer/tester laziness or lack of resources. For my implementation for this topic I wrote a very basic calculator program that had a series of tests that focuses on testing specific functions of the application. The tests were ran in Java and in a Junit test suite which made it easy to run and see the results. The implementation lacked in the area of complexity and needed to show a little more how a tester could focus on one specific “unit” for finding defects. For more information, see my Unit Testing Paper and implementation attached.

The second topic was Risk vs Requirements Based testing. When I initially chose this topic for my curriculum I assumed that Requirements Based Testing was the more practical of the two types of types as I had never used Risk Based Testing. It turned out that actually Risk Based Testing is the most commonly used of the two and the differences between the the two types of testing is more based on resource management then anything else. Requirements Based Testing is very much the academic standard and most practical type of testing. A tester is given a set of requirements that was designed by the developer or the client. The tester then must make sure that each of the requirements listed was completed and passes verification (usually a test that returns a boolean true or false). The difference between Requirements and Risk based testing comes in when you factor in limited resources. If a tester has to check 100 different requirements but is only given 4 hours to create his tests (based on costs of labor, etc.) then obviously a tester can not complete tests on all of the requirements. Instead testers can use Risk Based Testing which focuses on the prioritization of requirements. This prioritization is created by either the client, the developer or the tester themselves. The difference between how each of these three parties prioritizes their requirements or testing segments is vastly different. The client or

developer is more likely to factor in cost then the tester for instance. The overall point of Risk Based testing is that based on this prioritization, the requirements/tests that have the highest priority are executed first, in order to maximize the time used on testing. If you can complete tests on the more important functions then your coverage of an application is greater then if you completed as many tests as possible in a random order. For my implementation on this topic I created two test suites on two different applications. The first test suite was on another Calculator program and when creating this test suite I used the Requirements based method of testing. I created a list of requirements that needed to be achieved in the Calculator program and then I created a test suite that contained tests that checked each of the requirements. The next test suite was created for a Mine Sweeper game and this test suite used the Risk Based method of testing. Prior to creating the test suite, I sat down and prioritized each function of the game. Each function was given a ranking based on cost to fix and importance to the game. Based on this ranking I created tests for the top five most important functions. This test suite showed how I would have created a test suite if I was given a limited amount of resources. To see more on my research and my implementation, see my attached paper and code.

The third topic I covered was Regression Testing. In my research I found that Regression Testing, which is the concept of creating a test suite that can be run as a program evolves, has many different methodologies. Regression Testing, like Risk Based Testing, has to account for limited resources in its testing. Regression Testing is widely used in the real world companies and programmers want to keep rerunning tests on their applications as the software evolves or a new release/update has occurred. In my research I found that there are four major Regression Testing methodologies that are predominantly used. These are the “retest-all” methodology which is the practice of adding all new tests and running all previous tests. There is the “Regression Test Selection” methodology which is generally the idea of only adding and running needed test cases from the test suite. There are many different techniques for selecting which test cases to select, those can be seen in depth in my paper which is attached. The third methodology is “test suite prioritization” which is in essence Risk Based Testing in that it orders the test cases from most important to least important and runs the most important tests first. This is a prime example of the overlap between these testing topics. The final methodology is “test suite reduction” which the attempt to remove all non-needed or no longer relevant test cases from the test suite in order to save space and to save time. For my implementation for this topic I used new automation software called Ranorex to create a detailed test suite for the MSPaint application. The test suite was designed to be able to use any of these methodologies during regression testing and in my implementation explanation I show how a tester could take this test suite and use the RTS methodology in the future. For more information, please see attached paper and implementation.

The fourth topic that I covered this quarter proved to be the most difficult for me. This topic was Stress and Load testing. The concept of Stress and Load testing is very simple. Many new web applications or database applications need to know what their limits are in terms of concurrent users or network limits. Load and Stress tests are designed to show the tester what the limits of the applications are. There are a variety of ways to reach this goal, as a tester could increase the load the application is handling slowly of the course of time to see at which load the program breaks. The tester could also not attempt to break the application but instead at a higher and higher load in order to see how a programs response time or throughout is effected by the change in load. Load and Stress tests are more of a special type of testing because a majority of companies do not appropriate time or money for these type of tests as it is not relevant their software. In my research though it shows that over the last 10 years companies who have run web applications and have not stress/load tested their algorithms have seen disastrous results when their web applications get overloaded by concurrent users all attempting to use the software at the same time. The implementation for the stress and load testing topic was a very difficult one for me. I started out my creating a Client and Server application that would send a message from the Client to the Server. I then ran a script that would create thousands of instances of the

Client to see if it would break the Server. It turned out that I needed to change my code to first, do more work and second, include threading. I was successfully able to update my code so that instead of a simple message, the Client receives a file from the Server every time a call to the Server is made. This increased the work of the Server application but I was not able to include threading. As Professor Fluet explained, without threading there is no way for my test to truly break the Server as currently all calls are simply being queued and will eventually be executed. For more information of this topic, see my paper and partial implementation.

My final topic of the quarter was Database Testing. This topic was very interesting but at the same time also very mathematical heavy when it came to research. Database Testing I found is predominantly done very simply by testers with a simple SQL query that returns a specific value. If that value matches the expected value, then the test passes. In the research that I found on this topic I found that these simple SQL statements, though getting part of the job done, do not off complete coverage of a database. The goal of testing is to find errors and/or defects and I found three different papers that highlighted different methods of maximizing coverage. Each of these three methods can be seen in my paper in depth, which is attached. The Database Testing concept is very simply at a high level but a lot more difficult when you dig into the topic. A database test is attempting to verify that the values in the database are correct or logical. In a lot of cases the data is correct but in some cases there are exceptions where the data has been entered incorrectly or illogically. If a table named Persons exists and all people have a userId which is equal to the date(DD:MM:YYYY:X) where X is an integer, then there needs to be a test that userID that is equal to a date plus an actual integer. In my implementation I created two different test suites that have the same general tests only each test suite uses these test cases on different databases. In order to fully see the effect that these test suites have on coverage I must take my small scale testing suite and use it on a larger suite to see if I can find a defect. Overall you can see more information in my attached paper and code.

In the end I have learned a lot of amount various topics in Computer Science but most importantly learned how much testing interconnects to other testing concepts. This quarter alone I have had the ability to not only learn a variety of new testing techniques but to also familiarize myself with new software and unfamiliar concepts. I had only used a Java SocketServer object once previous to this work (Freshman year) but was able to brush up on this area during the Load Testing implementation. I also learned the lesson of staying organized and focused. Since I personally created my due dates and schedules, I had to make sure that I stayed on top of my work and was getting work to Professor Fluet at the time specified. This ended up being more difficult at times based on my busy schedule, so organizing my time became key. Overall I have enjoyed my independent study and have learned a lot of new things about software testing over the last few months.