**Comp 3520, Comp 4530**

**Software Engineering / Advanced Software Engineering**

**Assignment #2: Software Testing**

**Kevin Zwick & Dirk Neumann**

**T00463994, T00626124**

**Due September 25th, 2018**

**Table of Contents**

**Introduction**

Every software development company has the goal to develop software which can cover the costs to develop it and then generate a profit. To achieve this the customers have to like the product and be willing to pay money for it. A customer only pays money for what he orders and so the requirements must be fulfilled. But how can you see if this is the case. The answer is to test it. Only stable systems which guarantee to give the customer whatever he expected without mistakes and unreasonable behaviors is acceptable. So testing is a fundamentally important part of software development, not only to find bugs, but also to prove that the system does what it is supposed to. In this paper we will describe how a new software start up without many resources and time is about to get well tested software to the customer.

**Software Testing**

Lets first describe where we are, namely that our mobile application game is at the point that its’ first release is ready, and we need to validate this to get it released. We are using an Incremental approach to development using Agile methods. We are also using integration for some of our code, that we may have modified to work with our project. In short, we have a working version of our game, even it is a little on the simple side of what it could develop into. At this point we can add some more into how we developed our application, namely Test-Driven Development (TDD). TTD is an approach to program development in which you interleave testing and code development (Beck 2002; Jeffries and Melnik 2007). “Test-driven development was introduced as part of the XP Agile development method. However, it has gained mainstream acceptance and may be used in both agile and plan-based processes” (Sommerville, 2016). This means that we have developed tests for code while we were developing our code. The test data is chosen to give a good representation of the data the user will be inputting. We need to have test data at any boundary conditions for the unit as well as some examples from within these boundaries. This data handles the valid data. There also needs to be test data for invalid inputs for the unit. This puts us well on the way to testing our game and gives excellent code coverage.

**Unit Testing**

The tests that we developed while we were coding using TTD will provide a core set of tests to validate our application using unit testing. We will be using automated testing using these tests thru an automation framework such as JUNIT. Another way that we will validate our code is like pair coding. Instead the two of us sitting at the same computer while coding, we will each develop tests and code for a certain task, then the other person will inspect the code before the automated tests are run by JUNIT. This will give us the benefits of pair coding and inspection and let us separately work on different parts of the application. Using these methods will give us excellent code coverage as our application develops.

**Component Testing**

The next stage in our testing is component testing is component testing. This can be object testing or testing several interacting objects together. This then moves to the testing of composite components. “Testing composite components should therefore focus on showing that the component interface behaves according to its specification. You can assume that unit tests on the individual objects within the component have been completed.” (Summerville, I., 2016). During these tests the test data will include test data that were run in the unit testing. This continues till all the components and their interfaces are tested. Components are added together to create composite components which are then tested again. Composite components are added together until the complete application for this iteration is complete. Here the component testing is done.

**System Testing**

We now have a complete system, so we can do system testing, also called integration testing. This starts as the running of our test data thru the completed application. It also consists of running the system thru its’ various state changes. Complete coverage of all possible combinations of the different possible choices is basically impossible for anything other than the most basic application that has very limited linear paths. An interesting game will have an essentially infinite number of possible paths. We need to test the basic paths which would be the equivalent of the use-cases of an application. For our game we need to test the various parts of the game and how they work together. After all of these tests have been completed we have a working game that is ready for release testing and validation.

**Release Testing**

As we now have a complete working version of our game we can finish off with release testing. Release testing should not be done by the development team. This could cause a problem for a very small company currently consisting of only 2 members. We can do our own in-house Alpha testing ourselves to see if it works at least. However, being recent graduates of the B.C.S. program from TRU we have contacts from TRU that can be our release testers. Let our friends do the release testing for us by letting them have pre-release copies of our game. This would be essentially our group of Beta Testers. We then get feedback from them anonymously thru submission thru the software or by setting up a web form or a forum where they can leave comments that we can review later. This will later develop into a list of people that have provided feedback on earlier iterations of the game that will be our beta testers for future iterations and new products developed in the future.

**Validation, Conclusion**

**References:**

Beck, K. (2002). *Test Driven Development: By Example.* Boston: Addison-Wesley.

Jeffries, R., and Melnik, G. (2007). *TDD: The Art of Fearless Programming*. IEEE Software 24: 24- 30. Doi:10.1109/MS.2007.75.

Sommerville, I. (2016). *Software engineering*. Harlow, England; New York: Addison-Wesley, 2016.