McGill University

ECSE 321 – Project #4

Implementation and Automated Quality Assurance

Prepared by:

Brent Coutts (260617550)

November 18th, 2016

1 Testing

a) Briefly explain the differences between development, acceptance, and user testing.

1.1 Development Testing

Development testing is where the software developers test the components of the system that they are creating. The components can range from functions or object classes, to small groupings of these entities. Each of these components is tested separately in order to ensure that the component itself functions as it should. This is the lowest level testing process. These tests are run very frequently as the component is updated. Therefore, test automation tools like Junit are used to allow for quick re-runs of the tests upon updating the components.

1.2 Acceptance Testing

The final stage in the testing process, before releasing the system for operational use, is termed acceptance testing. In this stage of testing, user input data is supplied to the system as a whole, instead of simulated test data. This is an important testing phase because errors and oversights in the system requirements can be exposed. The system is exposed to real data that customers expect to function rather than test data that the developers anticipated system would be exposed to. Clearly if the system cannot handle user input, the system requirements need to be re-evaluated. Furthermore, issues with the requirements of the program can be exposed when the system's functionality does not meet the users needs or it is lacking in performance.

1.3 User Testing

User testing is the testing process where customers themselves provide their input and advice with regards to system testing. This approach is useful for software validation. It is essentially impossible for the programmer to replicate the exact stresses that the system will be exposed to in its working environment. This is therefore a very important testing stage, as it gives feedback on how the program will function when it is actually being used by the customer it was created for. There are three subcategories within user testing: alpha testing, beta testing, and acceptance testing.

In the alpha testing stage, the users are actively working with the software developers to test the system as it is being developed. The users can have a different perspective, allowing them to identify issues that would not be observed by the development team. They can provide practical information of the systems use that will allow for more realistic tests to be produced by the developers.

In the beta testing stage, an early system is released to the users/customers. The system can be specifically released to a specifically chosen group of customers that were early adopters of the system, or it can be released to any user that is interested in the software. This type of testing is ideal when the software product is going to be used in a large range of environments. Users can supply their feedback on issues they encountered when using the program to suit their needs.

b) When planning unit tests, why do we only test some of the potential input values rather than testing all of the input values? What are some of the strategies to choose which inputs to test?

1.4 Why test only some potential input values?

It is impossible to test all operating conditions for the program. Exhaustive testing is a method by which all possible input combinations are tested. This type of testing would involve testing every single possible input into the program. Furthermore, the java compiler would need to be tested by trying every possible java program. These facts make exhaustive testing essentially impossible. Even testing a large number of inputs can take a very large amount of time due to many different execution paths. A balance needs to be struck between the cost of testing and the cost of missing bugs.

1.5 Strategy to choose only some inputs to test?

A highly useful tool to choose a limited number of inputs to test is equivalence partitioning. This strategy groups inputs together based upon common characteristics. Then, the test data values are chosen by selecting cases bordering the partitions. In other words, one input value closest to the left of the partition’s edge and one input value closest to the right of the partition’s edge would be tested. One test case is also performed on a value in the middle of the partition. This method is extremely useful as it drastically limits the number of tests performed, while placing the tests in areas that are of the most interest. Values on the edges of the partitions are most likely to cause errors in the program, so testing that each side of the partition performs differently provides a high degree of confidence in the program.

2 Code Comments

a) Add a Javadoc comment to explain what the method is doing.
b) Add some good comments to the code.

Both of these steps were performed in eclipse. The eclipse archive file containing the Javadoc documentation, the commented method, and the JUnit tests is present in the assignment submission zip foler.

3 Equivalence Partitioning and JUnit Tests

For this project description a number is considered ‘lucky’ is the sum of its digits equals seven. For the purpose of clarity this type of lucky number is referred to as ‘first order lucky’. A number is also lucky if the sum of the sum of its digits is equal to seven. This type of lucky number is termed ‘second order lucky’. Any number that is not a lucky number must be considered ‘unlucky’.

There are an infinite number of lucky numbers, limited only by the range of integer values that the compiler can allocate. Thus, equivalence partitioning was used for a few test cases of interest in order to validate that the program is functioning properly.

The first equivalency is when a number is first order lucky. The lucky number itself (16) is tested, along with a number one integer larger (17) and a number one integer smaller (15) that are both unlucky numbers.

The second equivalency is for a second order lucky number. It is asserted that the number itself (88) is lucky. Then it is asserted that both a number one integer larger (89) and a number one integer smaller (87) would be considered unlucky.

The final equivalency is for a very large number with many zeros. Similar to in the last tests, the number itself is tested (700000000), followed by tests for a number one integer larger (700000001) and a number one integer smaller (699999999) than the lucky number.

If all of these test cases pass, one can be confident that the program functions as it is should. This confidence stems from the fact that the program is able to take a variety of different inputs/tests, while producing the proper output. The method is fully covered by the test cases as some numbers require checks for being a solely a first order lucky number and other require check for being a second order lucky number.

3.1 Assumptions

- The program does not need to handle negative numbers. If a negative number is input, either it will be forced to be positive or an error message would be thrown.
- The program only needs to function for integer values.

4 Important Note

When the JUnit tests are run, it is realized that the program does not sum the digits of the numbers properly. It is assumed this is by design allowing the programmer to see the benefits of testing for uncovering issues in the code method.