Software Testing Plan

4/3/20
DigiTool Inc.

Project Sponsor: Xi Zhou
Team Faculty Mentor: Fabio Santos

Team Members

Traybourne North
Spencer Clark
Langqing Dai
Miguel Quinones

Version Number: 1.0

Table of Contents

1.0 Introduction

1.1 Application background

Our application is a web application that will allow students to practice Digital Logic concepts outside of the classroom any time they need to. The motivation for this project came from our client and was a result of him wanting a study toolkit that his students could use. To expand on this, students needed a free tool that they could use to practice the concepts of the course, and our application provides that. Our application consists of two modules, Karnaugh Maps and Numeric Conversions, and these modules are located within two modes in the application. Those two modes are “Module mode” and “Practice mode”. The main goals of our application is to provide students a tool to practice the concepts of their class, as well as a way for our client to track student progress within our application.

1.2 Software testing

Software testing allows developers to test the components of their software and ensure that everything is working correctly. This includes making sure that software is working according to the necessary specifications, as well as fixing any errors that might result from testing. The goal is that by the end of the testing, every component of the software is running properly and students won’t encounter any errors.

1.3 Our testing plan

Our plan is to conduct unit testing, integration testing, and usability testing on our application to ensure that it is working according to the specifications we designed. For the unit testing, we plan to conduct unit tests on each core aspect of our software, meaning the login functionality, the two modules in our “Module mode”, and the two modules in our “Practice mode”. For the integration testing, we plan on testing accessibility to our database as well as each method used in our web application. Finally, for usability testing, we plan on testing if a user is able to successfully work through the modules in our application.

Our testing plan will allow us to test the most important aspects of our application and fix possible errors that could occur through testing. Since our application is designed to be a tool for students to practice the concepts of Digital Logic, most testing will be done on our modules focusing on Karnaugh Maps and Numeric Conversions, ensuring they work completely. Our next focus is the login functionality and database communication, as this is also important to our application. The various tests we run will ensure that our goal is completed and we will begin by discussing our units tests.

2.0 Unit Testing

2.1 Purpose of unit testing

Unit testing is a method of software testing used to ensure that the modules and components of a piece of software are running correctly and that there are no issues within them. The main goal of unit testing is to make sure that the expected outcomes of your software meets what's needed and if there are any issues, you can correct them. For our application, we plan on using Javascript frameworks called MochaJS and Chai to perform our unit testing, as they streamline the process by providing testing functions we can use to test our code. These frameworks work by using assertions we pass to them to work out what the expected outcome of our code should be, and we can use that information to check if everything is working properly within our application.

2.2 Focus of our testing

We plan on testing the main modules of our application, Karnaugh Maps and Numeric Conversions, as well as the login functionality and database communication our application does. By breaking this down, we get this list of specific methods to test:

- Karnaugh Maps
  - Converting 3-variable truth tables
  - Selecting groupings for 3-variable mappings
  - Finding an equation for a 3-variable mappings
  - Converting 4-variable truth tables
  - Selecting groupings for 4-variable mappings
  - Finding an equation for 4-variable mappings
- Numeric Conversions
  - Binary, octal, and hexadecimal numbers to decimal
  - Decimal numbers to binary, octal, and hexadecimal
  - Binary numbers to octal and hexadecimal
  - Octal numbers to binary
  - Hexadecimal numbers to binary
  - Binary, octal, and hexadecimal fractions to decimal
  - Decimal fractions to binary, octal, and hexadecimal
  - Binary fractions to octal and hexadecimal
  - Octal fractions to binary
  - Hexadecimal fractions to binary
- Database communication/login functionality
  - Logging into the database
  - Ensuring the proper information is sent to the database

2.3 Karnaugh Map Testing

Since the Karnaugh Maps module consists of 3 sections (translating truth tables, grouping, and equation writing), we plan to conduct manual unit testing in each section. For translating truth tables, we plan to create 100 randomized truth tables with answers to match each truth table through a loop. From there, we will manually use assertions to compare that to the validated answer generated from our functions. We also need to test each function involved with translating truth tables in order to make sure each function works properly as well.

For grouping, we plan to create 100 randomized truth tables and Karnaugh Maps with answers to match each Karnaugh Map and each truth table through a loop. From there, we will manually use assertions to compare that to the validated answer generated from our functions. We also need to test each function involved with grouping problems in order to make sure each function works properly as well. The only thing we can’t test using a unit test is how the user forms rectangles in the canvas. How we test this will have to be done manually by checking to see if rectangles can properly and accurately be drawn in the places they need to be drawn.

For writing equations, we plan to create 100 randomized truth tables and Karnaugh Maps with auto generated groups. Each truth table and Karnaugh Map will also have answers and all of this will be generated through a loop. From there, we will manually use assertions to compare that to the validated answer generated from our functions. We also need to test each function involved with grouping problems in order to make sure each function works properly as well.

2.4 Numeric Conversion Testing

For our numeric conversions, the values that users input will only ever consist of one correct value, which is the one that is the correct conversion. With this being the case, we don't have many equivalence partitions or boundary values that we need to check. Still, our system will never ask users to input conversions below 0 or above 1,000, so we can use these values as baselines. However, there will be differences between the different conversions.

For binary, octal, and hexadecimal to decimal conversions, our equivalence partitions and boundary values will vary depending on the conversion. For binary to decimal, we will have boundary values of 0, 1, 256, and then 257, and our equivalence partitions in this case would be 0 and lower, 1 to 256, 257 and above, and we would only accept the second partition as being correct. The correct input would be based on the number given to convert, with incorrect inputs being any input that is not the correct conversion.

For octal to decimal, we'd have boundary values of 0, 1, 512, and then 513, and our equivalence partitions in this case would be 0 and lower, 1 to 512, and then 513 and above, with the second partition being the valid one. The correct input would be based on the number given to convert, with the incorrect input being any input that is not the correct conversion.

For hexadecimal to decimal, we'd have boundary values of 0, 1, 1,000, and then 1,001, and our equivalence partitions would be 0 and lower, 1 to 1,000, and 1,001 and above, with the second partition being the valid one. The correct input would be based on the number given to convert, and the incorrect inputs are the ones that are not the correct conversion.

For decimal to binary, we’d have boundary values of -1, 0, 1, and then 2, and our equivalence partitions would be -1 and lower, 0 and 1, and then 2 and above, with the second partition being the valid one. However, for this conversion, users would need to put in the binary representation of the number they’re given to convert, so they’d input 0 and 1 as often as necessary as opposed to only once. The correct input would be based on the number given to convert, and the incorrect inputs are the ones that are not the correct conversion.
For decimal to octal, we’d have boundary values of 0, 1, 512, and then 513, and our equivalence partitions would be 0 and lower, 1 to 512, and 513 and above, with the second partition being the valid one. The correct input would be based on the number given to convert, and the incorrect inputs are the ones that are not the correct conversion.

For decimal to hexadecimal, we’d have boundary values of 0, 1, 3E8, and then 3E9, and our equivalence partitions would be 0 and lower, 1 to 3E8, which includes the combination of numbers and hexadecimal numbers within that, and then 3E9 and above. The second partition would be the valid one, and the correct input would be based on the number being converted with incorrect inputs being anything else.

For binary to octal, we’d have boundary values of 0, 1, 400, and then 401, and our equivalence partitions would be 0 and lower, 1 to 400, and then 401 and above, with the second partition being the valid one. Correct inputs would be ones that are the correct conversion of the number given, with incorrect inputs being anything else.

For binary to hexadecimal, we’d have boundary values of 0, 1, 100, and then FB, and our equivalence partitions would be 0 and lower, 1 to 100 including hexadecimal numbers, and then FB and above, with the second partition being the valid one. The correct inputs would be the ones that are the correct conversions of the number given, with incorrect inputs being the ones that are not the correct conversions.

For octal to binary, we'd have boundary values of -1, 0, 1, and 2, and our equivalence partitions would be -1 and lower, 0 to 1, and then 2 and above. The second partition would be the valid one, however users could input 0 and 1 as often as necessary for this method. Correct inputs would be the ones that are the correct conversions, and incorrect inputs are the ones that are not.

For hexadecimal to binary, we'd have boundary values of -1, 0, 1, and 2, and our equivalence partitions would be -1 and lower, 0 and 1, and then 2 and above, with the second partition being the valid one. Like our other methods where we're converting to binary, users can input as many 0s and 1s as necessary. Correct inputs would be the ones that are the correct conversions, and the incorrect inputs are the ones that are not the correct conversions.

2.5 Database Testing

Within the tracker module, we must test that users can log in and the application knows which student is using it. Second, the tracker must submit the correct values to the correct places within our database table.

Since the database responds with a student ID that gets saved on the client side, we can ensure that the user can log into the database by asserting that given username and password returns a correct student ID.

Validating the submission values is a bit more difficult as the testing frameworks can't access what is logged to the database so we will have to observe each value in the correct place after a tracker call. Thankfully since this is all done with the same call, we do not have to manually validate multiple conditions.

3.0 Integration Testing

3.1 Reasons for Integration testing

With unit testing we can be sure that each individual component works properly, but we still don’t know if each component works with each other. This is where integration testing comes in. We need to check that each component can properly interact with the other components in the system, eg. passing data, performing functions, creating connections, etc. The interactions between systems is what turns each piece into a fully fledged and useful product.

Our systems are mostly self contained with one exception, the database. Each module needs to be able to post the users results to a table in the database. This creates a boundary between the tracking system and each module that must be verified.

3.2 Focus of testing

The most important integration we have is the usage of tracker methods in each module. This functionality is a direct key requirement and there are several places where the integration can go wrong. Focusing our attention here will provide the most benefit for our time and guarantee the most functionality.

Another place of integration would be between our react component setup and the webpages. This was one of the first pieces to be implemented and has been functional since, it is also reasonably simple in design, therefore we will not be testing it.

3.3 Plan for testing

To ensure that each module uses the tracker properly we will need to observe 3 points.

1. The module creates and maintains necessary data such as:
    - Starting a timer on problem load
    - Stars given on final submission
    - Mistakes made updated for each failure
    - Whether the student completed the problem successfully
    - Which module is being used
    - Whether it is test or practice mode
    - Which student is using the application
2. The data passed to the tracker is correctly ordered and formatted.
3. The tracker is called only on completion of a problem and at every possible completion including failure.

We can test most of this with assertions of our tracked values via MochaJS and Chai, the rest can simply be observed as it is logged to the database.

4.0 Usability Testing

4.1 Meaning, goal, and how it works

Usability testing focuses on the interactions between our software system and the end users, which in this case will be the students enrolled in the Introduction to Electrical Engineering course taught at Northern Arizona University. The primary goal is to gather information on problems users face when using our software and we also want to gather data to determine how satisfied each user is with our product.[1] How usability testing works is we first come up with a test plan. What's within our test plan consists of a scope, purpose, schedule & location, scenario, the number of participants, etc. Once we have created a test plan, we would have to recruit participants. Lastly, after recruiting participants, we would have to analyze and report our findings. By taking these necessary steps, we can make changes (if needed) to ensure that our users will be satisfied with our product.

4.2 Plan for conducting usability testing

Before coming up with a testing regime, we have to understand specific characteristics of our project which include the background of end users, our project's novelty, and consequences that we could face if parts of our project were badly designed. We know that the background of end users will be students that attend Northern Arizona University and they'll either be freshmen, sophomores, juniors, or seniors. Our project's novelty is the fact that it encapsulates many different websites into one and offers many interaction activities to the user. This includes user input, grouping, and even custom question practice.

[1] https://www.usability.gov/how-to-and-tools/methods/usability-testing.html

Rather than entering many different websites to practice a specific thing, users can just use our website to practice and hone topics such as Karnaugh Maps (translating a truth table into a Karnaugh Map, grouping based on a Karnaugh Map, creating an equation based on a given Karnaugh Map) and Numeric Conversions (binary to octal, octal to decimal, decimal to hexadecimal, fraction conversion, etc.). As a group we understand that our design isn't 100% finished. The user interface presented to the user can be much better and the code presented with each web page can be a lot more modular than what it currently is. This can potentially increase response time between the user and the web page and it can decrease the time needed for the next question to be generated.

4.3 Testing regime and alternative methods

We had to take all of these factors and decide whether or not to record quantitative or qualitative data. We decided to base our testing regime based on qualitative data since we'd be able to observe the pathways participants took, analyze the problems participants could face, and understand comments / recommendations that they had once they finished testing our product. Our testing regime plans to involve our client giving the current students in his class the product to try out. From there, he would be able to gather data and share it with us to better improve our product. The original goal was to have our client give our product to students to test while he was teaching in China, but the corona virus prevented that from happening.

Our data will be stored in text documents and the data will be based on questions that we provide to the end users. Each report will include a background summary of what was tested, the methods held for each test, the results of the test, and findings & recommendations. Based on the commonalities found in each report, we can analyze them and form insights as to how to improve our product.

Depending on how the next couple of weeks play out, we would have to think of an alternative to how we conduct our usability tests if we are still unable to attend classes in person. If anything, each of us can ask five people that we personally know to test our product and after we do that, we can gather our results using that method. We would still be able to gather a lot of information, but it wouldn't be comparable to a whole class testing our product.

5.0 Conclusion

To conclude this paper, we believe that the unit tests, integration tests, and usability tests we conduct will result in an error-free, fully functional, and highly usable product. Running unit tests on our modules (Karnaugh Maps and Numeric Conversions), login functionality, and database communication will ensure that our product works the way that it should. It will also identify any errors within our product that needs to be fixed. Running integration tests on each component of our web application will make sure that each component can properly interact with the other components in the system. If each component can work correctly with other components in our system, it will allow us to conclude that our product is reliable and ready to use. It will also make sure that errors are identified early, which will allow us to fix those errors before the deadline. Lastly, by having end users test our product through usability tests, we can get an idea of our product's satisfaction rate. From there, we can make the necessary changes to better our product as the due date gets closer and closer.

By gathering and analyzing all test data, our product can be modified to have a high satisfaction rate and a high performance / functionality rate. Our product can also meet the needs of our end users and our client based on the data we gather from each test. We are completely confident that our product can reach its full potential before it is handed over to our client. Although there has been minor setbacks (COVID-19), this won't stop us from implementing tests to better our product.