Software Test Plan

Version 1.2

Team: Excelsior

April 3rd, 2020

Project Sponsor:

Dr. Toby Hocking

Mentor:

Dr. Eck Doerry

Team Members:

Austin Torrence, Matthew Rittenback, Brandon Thomas

Table of Contents

1. Introduction

Technology runs most of the modern world and it's the engineers of the world that design, build, or maintain technologies ranging from personal cell phones to the airliners that move millions of people across the world every day. Flaws in engineered products can often result in catastrophic failure sometimes resulting in the loss of human life. To ensure the quality of the engineering workforce, universities make sure their students are completing programs that are accredited by agencies who set the global standards for their major. These agencies provide guidelines to make sure that each and every engineer who graduates from an accredited program knows the necessary information to excel in their field and produce safe and high-quality engineered products.

ABET is a non-profit agency that accredits programs across all engineering and technology fields. ABET is focused on maintaining the quality of education for engineers enrolled in their accredited programs by (a) providing quality standards that programs at universities like Northern Arizona University need to adhere to to stay accredited and recognised; and (b) ensuring that programs monitor and enforce those standards. Graduating from a program accredited by ABET informs employers that this potential new hire has the ability to perform in their duties at a professional level.

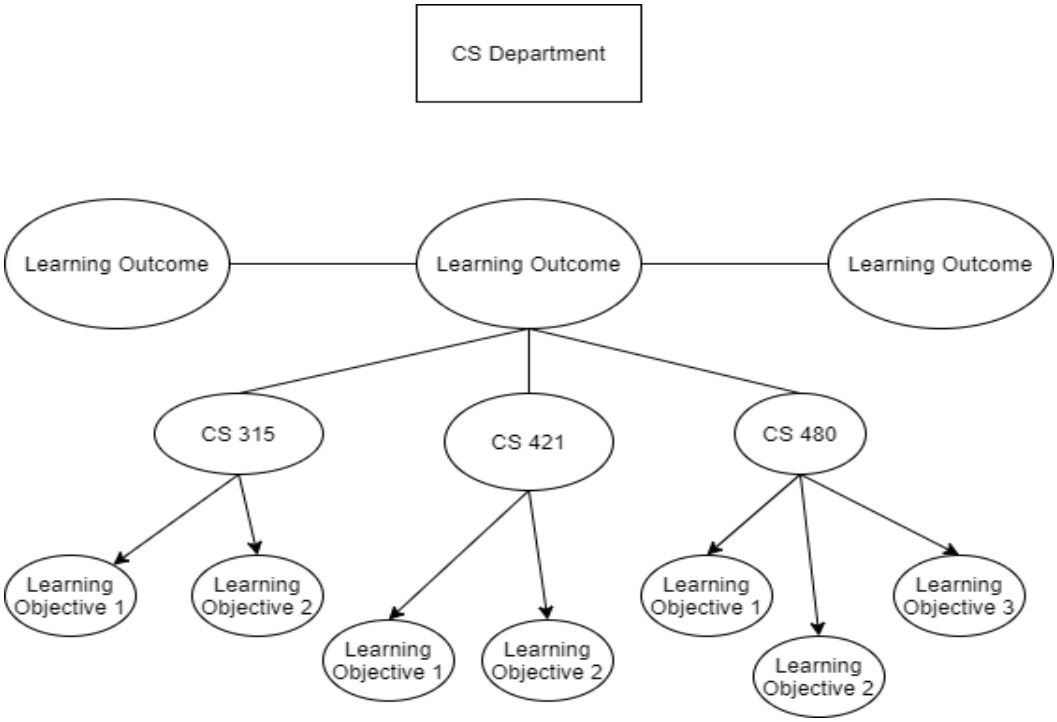

Figure 1: ABET Model showing Learning Outcomes supported by courses that all contribute to various Learning Objectives

A department follows a specific guideline given by ABET which is shown in Figure 1. Departments will create ABET-approved learning outcomes that are to be supported by a variable number of classes. These classes will then have specific learning objectives related to the course that directly support the overall learning outcome. These learning objectives are measured in a variety of ways depending on the course, and examples of these measurements include questions on a final exam, a portion of a presentation, or any other academic benchmark. These learning objectives will then be manually recorded in an excel spreadsheet by the professor on a scale of 1 to 5 depending on the individual student's performance regarding that learning objective. These excel files then produce a short summary of the percentage of students who are performing to ABET standards. ABET requires that at least 70% of the students are at or above that 3 level.

Currently for the ABET coordinator, the process of monitoring of ABET data collection across multiple learning outcomes is inefficient and error-prone. There is no technologies in place at the moment for the coordinator to check which professors have completed uploading ABET data for a given course or semester. This can lead to professors forgetting to collect and

input data for a course from a past semester or even past year, which makes it difficult for the coordinator to collect the data. With 6 year intervals, the ABET coordinator is responsible for compiling all of the collected ABET data from that time period, to prove and show that the department is successfully supporting it's learning outcomes through the scores from the learning objectives that students achieve. With missing or lacking data from professors, it can be difficult for the coordinator to successfully pull together the necessary data at the end of these intervals to support the department in proving they are meeting ABET standards through the courses being taught.

As of now we have a product that is capable of taking in these excel files, exporting the desired data to a local config file, and displaying graphical information regarding specific learning outcomes created by the user. To ensure that our program can accurately import excel files, create a local config file, create course offering and learning objective objects, and graph data pertaining to the learning outcomes, we have devised a testing plan to make sure that each function, integration between portions of our product, and the usability are working correctly and as intended. Hopefully our testing design will help us or any future contributors debug any issues that arise and help us monitor the overall performance of our product.

2. Unit Testing

Unit testing is essential when it comes to programming. Unit testing is used to validate behavior at a smaller level and making sure that this one specific unit on its own is functioning as intended. If all units are tested properly, it can help identify issues or bugs in the program quicker and easier, not only for us as we fine tune our product, but for anyone else who wishes to contribute to this program after our capstone has finished.

We decided on using the framework Jasmine for our unit testing portion. Jasmine is a lightweight, behavior-driven testing framework for Javascript. It has no outside dependencies and has easy to write, well documented syntax making it easy to create simple unit tests. As our program isn't very large we will have unit tests for each and every function in our code. We want to build our tests in such a way that we can run the tests and be able to identify if any key individual components are not performing.

Another framework that we are using is Spectron, an Electron testing framework. It is a Node JS install for electron, it will be testing the desktop app part of this program. It is compatible with Jasmine since Jasmine is a node js package. Spectron will be checking if the window is visible to ensure it is working properly. It will be testing different features that our app will have for example the title of the app and when the app closes, it writes all errors into a log showing what test has failed.

Our specific functions and how they will be tested is outline below:

- app.start().then(funtion() {})
    - This function will check if the window of the program is open and/or created.

- App.start()then(function(){}).then(function(locatlStorageCheck){ return.equal(localStorageCheck, './config')
    - This function will check if the config folder is created.

- catch(function(error){console.error('Log', error.message)})
    - This function would be catching all error messages and putting them into a log file after the app is closed.

- function createWorkBook(excelSheet, sheetNum)
    - Even though createWorkbook() uses multiple functions it still takes an input and has an output meaning we can still use normal unit tests with it. Essentially when calling this function, the user's uploaded file is then parsed and a workbook is created. Once we have a workbook to be able to use, data from each course offering is taken and put into an array of courseOffering objects. We will be testing this function by making sure that when it is called, an object of the correct format is pushed to the courseOffering array. We will be making sure the object has every piece of data without any undefined or missing components.

- function multiSheet(ws, sheetName)
    - This function is written inside createWorkBook and also uses multiple functions. We decided that since this function is a part of the createWorkBook() function

that it would be best to test these two functions together. Essentially this function depends on the entirety of createWorkBook() as a whole so we won't have a unit test for this function.

- function nextColVal(curCol)
    - This is a helper function to createWorkBook() and multiSheet(). Essentially this function takes in a string of the current column and returns the letter of the next column. We would test this function by assuming that if an 'A' was inputted, the expected output should be 'B', inputting 'C' gets 'D' and so on.

- function createlObjArr(summaryCellRow, summaryCellCol)
    - This function takes in the row and column of the cell where Summary exists. It then looks at the cells to the right of it and grabs the titles for each cell going to the right until it hits a blank cell. We would test this function by creating a workbook specifically for testing and make sure that when this function is called, it pushes the correct values to the lObjTitles array.

- function fillLe(qRow, qCol, le, x)
    - This function takes in the column and the row of the queue cell (Summary) and puts all the data for the 4 rows into an 'le' array. This data includes the number of students at a 1, 2, 3, or 4 or higher level. This is the data that is then used to create the bar charts pertaining to the learning outcomes. To make sure this data is getting imported accurately, we will use the aforementioned testing workbook and make sure these arrays are being filled with accurate data.

- Function createGraph()
    - This is the function that takes in any kind of data and puts it into a graph. In our project we will be using this function to graph data on learning outcomes for each semester that that learning outcome appears in. We will be testing this function by making sure that when a graph is created, the data being graphed is accurate. We will be using test data to put into this function to make sure that what is shown matches what is inputted.

3. Integration Testing

In the integrating testing section, it is to focus on our program components, the test will ensure each portion of the project is working together as intended. To do this, each test that is created will be verifying accurate communication between different portions of the product. The test will be ensuring that the program as a whole is functioning correctly.

The main parts of the program that will be tested on will be Electron, the config file, the main program, and the helper program. The main program communicates with the config file which then communicates with the helper program. Electron is what keeps the main program packaged up into an executable desktop application.

Our testing for Electron uses the Spectron framework to test the packaged program. Electron converts HTML, CSS and Javascript into a desktop app and creates a main window. Our test will determine whether or not an issue has arisen with Electron or our written program by identifying if a window has been created. The Spectron will be used to build multiple functions for testing Electron. The framework will collect all test results, and when the app is close, the results will be logged into a testing log file.

To test more of the HTML and Javascript side of the project, Jasmine will be able to run our program and verify communication with the log file. The config file is used by importing excel files from our main program, creating a config file on close (if one is not already made), writing data from the courseOfferings into JSON format in the config file. Once data is loaded into the config file, the program will be able to import that data back into the main program to be used in graphing and monitoring data. Our tests will verify data has been entered into the log file, that it is the correct data, and that that data will be able to be reloaded accurately back into the main program. This is tested by checking the path to file after data has been imported through the app, verifying the JSON objects in the config file are correct using test data, and verifying that the data that has been accurately reloaded by the program from the config file using the same test data.

In terms of monitoring information, a helper program will be implemented that will use data from the latest config file. This will essentially check for incomplete or missing data and notify the appropriate staff. Using predetermined test data with Jasmine, we will be checking to see if the helper program can accurately determine whether or not data is incomplete or missing based on the test data, and effectively send a notification to the appropriate professor. The notification testing will be done by making sure the email is populated with the correct information pertaining to the test data, and if it successfully sends the email to the appropriate test email. Between this test and the config file testing, contributors will be able to better pinpoint if data in the config file is inaccurate or if the helper program has a bug.

4. Usability Testing

In this final section discussing the testing plan, we will cover why and how the usability tests will be beneficial to the development of AMTV. The goal of usability testing is to get an understanding of how the end-user interacts and uses the functionality of the software. Gaining this knowledge of the user's experience with our software, gives us a vital understanding of the user's workflow within our software. If the user finds the software too difficult to use on a regular basis, such as updating ABET data across semesters, our software will lose the interest of potential end users. One of our key requirements for this project was to make sure the software was easy to update and maintain between semesters. Knowing this requirement, it is critical that we conduct thorough usability testing to ensure we meet the requirement. With proper usability testing, we will be able to receive feedback from the user on the overall design, UI and ease of use.

AMTV is a tool that is built for use by ABET coordinators to easily compile and track data for numerous learning objectives supported by multiple courses inside a department. The tool needs to be intuitive in how the coordinator can upload, update and view his/her collected ABET data and provide clear indicators of missing data or poor course performance. If the tool fails to deliver on assisting the coordinator with managing ABET data, then it might end up costing the coordinator more time in the long run with trying to manually compile his/her ABET data. To ensure AMTV can assist coordinators in tracking and monitoring ABET data for their department, usability tests will be performed to provide insight on the software's usability.

In order to gain an understanding of the overall usability of AMTV, we will be conducting a number of tests that target specifically the user's workflow and experience while using AMTV. Given the fast-tracked nature of the project, we will be working with a few select end users to conduct our testing. However, these few key users were chosen for their extensive experience with collecting ABET data, and have possibly been ABET coordinators themselves at a point in their professional career. Due to their experience, the input and feedback these user's provide will be valuable as they can inform us on the ideal workflow they would like to achieve while using our software.

To ensure the usability of AMTV, two separate series of usability testing will be conducted. The first series of testing will include showing both Dr. Doerry and Dr. Palmer working prototypes of our software that demonstrate the basic features of AMTV. These professors then will be able to use our software with mock ABET data they will produce in order to provide feedback on their experience with collecting and monitoring the mock data. These tests are intended to be done a few times with new iterations of our software to address the feedback and hopefully achieve the ideal workflow for the professors. Some key questions to provide the understanding we are searching for, may include:

- Was installation clear and easy to complete? Why or why not?
- Was initial data uploading simple or difficult to complete?
- Is it easy to navigate through the different LO's and courses?
- On a scale of 1-10, how was your overall experience using AMTV? Please explain your rating.

The second series of testing will be conducted alongside the first series with the goal of working with Dr. Hocking to ensure a solid overall user experience with emphasis on the data visualization. The feedback received from Dr. Hocking will be useful in developing the overall user flow, but it will be especially useful in developing the versatility of our data visualizations. Some key questions for Dr. Hocking may include:

- Is it easy to navigate through the LO's and courses to display a desired graph?
- Do the graphs provide enough functionality to display the desired information?
- On a scale of 1-10, how was your overall experience using AMTV? Please explain your rating.

In the two weeks leading up to UGRADS, we will be sending our target testing audience a prototype of AMTV with our questionnaires, with the request for an online meeting to discuss the results and feedback they may have. Ideally the group would like to receive feedback from all three professors during the first week to allow for improvements and a new version of AMTV to be developed and to be tested in the second week. After meeting a number of times with our end-users who have completed our usability tests, we will have received important feedback to help revise and improve the overall usability of AMTV.

5. Conclusion

In today's world, technology is developing at a rapid speed, and the demand for engineers is increasing at an equally rapid speed. However, some of the technologies that engineers can build can pose a serious danger if built incorrectly. Universities that educate these engineers use ABET accreditation to prove that they provide a certain level of quality and that engineers know the necessary tools/skills to build safe functional products. While effective, the data collection portion of ABET is arduous involving potential for missing or incomplete data. This process makes it hard for professors to remember to collect data, to know if data is complete, and to understand and analyze the data to see outcome achievement progress or failure.

In this document, we hope to provide a solid testing design that will be useful to any contributors to our product as of now or in the future. These series of tests will ensure that the program will be accurately importing and storing ABET data, graphing them accordingly, and monitoring the data providing notifications to appropriate staff when necessary. We hope that our testing phase will help us better monitor the overall performance of our program and to adjust accordingly.