# Team 4 4/30/2016

SER 216

James Austin

Nergal Givarkes

Enya Yan

Connor Premuda

Plotter Test Summary

# Objectives and Testing Goals

The goal of the team was to thoroughly test and examine the Plotter graphing application in order to find any defects, gaps in functionality, and any possible enhancements that could be made to the program. Ultimately, the team set out to optimize the user experience by logging, fixing, and re-testing any findings. To comprehensively test the program, the team analyzed and discussed different testing methodologies before deciding on a combination of automated tests (such as unit and functional tests) along with manual test cases. Although automated and programmatic testing as a big advantage in being reusable and valuable in regression testing for changes in the code, the team decided that manual testing was also very important given the mathematical nature of the application. Since there are a plethora of possibilities regarding what the user can enter the application, it would be an unwise use of resources to try and automate every single use case. Additionally, to ensure that the program can handle regular use by a user, performance and load testing have also been implemented; although this does not test functionality, it will ensure that the application will run with the expected performance in a production environment. The team decided to do this testing programmatically since load testing is very hard to reproduce manually.

# Tesing Results

## Unit Testing

## Integration Testing

## System Testing

## Performance Testing

## Regression Testing

# Summary of Bug Fixes

## Bug Fix 1 – Incorrect Integration Values

* Unit test of the three integration methods (Trapezium, Gauss and Simpson) unveiled that the values returned were incorrect.
* The algorithms of each method did not match the mathematical formulas.
* Trapezium and Simpson methods were modified and now produce the correct values.
* Gauss method required too much math and was avoided due to time constraints.

## Bug Fix 2

## Bug Fix 3

## Bug Fix 4

# Summary of Enhancements

## Enhancement 1

## Enhancement 2

## Enhancement 3

## Enhancement 4

# Significant Challenges

During the development and execution of the Plotter application test plan, we encountered many challenges. One of the biggest challenges was learning the source code for the application, and working with it in order to implement automated test cases. Many components of the source code were private in order to maintain good encapsulated coding standards. However, this made testing specific components, especially those of the user interface, difficult to test with fixed input. Furthermore, getting the specific output needed in order to verify the test is succeeding was difficult, or even impossible, at times. In order to use our testing resources efficiently, we decided that only certain aspects of the program would be tested programmatically, and other aspects would be tested manually. The team came to this conclusion after careful discussion around what the most resourceful way to test the program would be.

Since our team was composed of testing engineers from different parts of the country, our meeting times proved to be an obstacle that we had to work with each week. Face to face meeting were a crucial part of the effectiveness of the testing team. Through meetings, we are able to make sure all team members are aligned on the goals and deliverables for that week (as well as for the project as a whole), help out other team members who run into road blocks, and discuss/triage any defects or enhancements that were discovered during the testing process. The team was able to work through this challenge by agreeing on a set time each week the team could meet. Additionally, team members did their best to attend impromptu meetings that came up during the week in order to address for urgent issues. All team members were reliable enough to communicate via another channel if there were unable to join a face to face meeting.

Another large challenge the team faced was choosing and learning new testing frameworks to use for this project. It was decided to use other frameworks on this project, such as TestNG and JMeter, because the team felt that these more advanced frameworks contained functionality and capabilities that more traditional frameworks did not have, making the testing process more efficient and effective. However, this required each team member to set up their testing environment to use the new frameworks and learn how to use them. This resulted in a somewhat challenging learning curve that the team had to work through.

# What We Learned

The team was able to take away many learnings from the Plotter testing project. One realization that the team had is that in order to thoroughly test the Plotter application, may different types of tests had to be used. For example, it proved very difficult to try and test the program with just black box testing. We saw that many defects could have been easily missed without testing the specific inner workings of some of the application’s components. Similarly, if the team were to rely entirely on white box testing, it would take endless hours of coding resources to cover all of the possible use cases and input of a mathematical graphing program such as the Plotter. As a result, the team carried out the test execution using a mixture of unit and functional tests, as well as manual tests, to ensure as many defects were discovered as possible.

The team also learned that writing up a testing plan and specification made it easy to organize the testing and to prevent tests from overlapping each other. This also made it easy to visualize what was being tested, and what gaps there might have been in the test cases. Furthermore, the team realized that although having a test case outline was very helpful, approaching the testing a regressive stand point was also useful. For example, instead of waiting to find bugs in the program to test, tests were written ahead of time to check for expected output. This way, our automated tests did not just cover the defects that we corrected or enhancements that we implemented, but it also prevented new defects from being introduced as a result. This also added long-term value to the application since these tests can be re-run in the future when new features are added to the program.

One big learning that the team took away from this project was that following process and staying organized can be very beneficial. The team felt that they had a large advantage by following the correct process when it came to executing test cases, identifying/reporting a bug, fixing the bug, and re-testing the program. Sticking to this process made executing test cases easy and efficient. The team also found great value in keeping communication frequent and strong since this allowed each team member to understand the status of the testing, and be aware of what others are working on. Furthermore, it allowed team members to help each other out when road blocks were encountered (i.e., environmental issues, application running issues, etc.).

One of the deliverables the team produced were diagrams of different aspects of the Plotter program. Although creating these diagrams at the time seemed tedious, the team found great benefit in being able to reference back to the diagram when executing the test cases. If a team member had trouble understanding the expected behavior of the application, or the construct of certain components of the application, they could simply refer back to one of the diagrams for assistance (such as the state diagram, class diagram, use case diagram, etc.). Using the diagrams during testing made it easier to write tests as well as being able to pin point bugs.

# What We Would Improve

During the development/execution of the test case plan, the team came up with many ideas that would help refine the process if we were to have to do it again. One of the biggest things we would have done differently for next time would be to have a better understanding of the construct of the application before writing up the test plan and test specification. This is not to say we had no understanding of the application at that point in the process, but we feel we learned much more about the application during the test execution that would have been very useful when writing up the test cases. Additionally, we could have used this information to determine which specific sort of tests should be used for different components of the program. For example, if we would have known that a certain component would have been difficult to access in order to write an automated white box test for, we would have started out black box testing the aspect from the beginning (this would have saved time and made the testing process more efficient).

The team also felt that allowing for time for learning new technologies would have been very advantageous in the long run of the project. Although spending a lot of time on the new frameworks seemed time consuming at the time, being more knowledgeable would have helped expedite the test execution process. This was not limited to just test frameworks, it also took team members who were not familiar with GitHub as bit of time to learn and adjust to using this sort of repository. For team members who had never used GitHub, it was difficult for them at times to be able to confidently push up code changes or written document modifications.

The team also learned that the most involved part of carrying out the test plan was the test execution week. As a result, we this was also the week where the most team communication was necessary. If we were to carry out a project like this again, we would have attempted to set up more frequent team meetings that week. This would have helped us triage bugs faster, and also discuss any code changes that the team thought would be necessary to push to our code base. Without these meetings, the team members often pushed code changes without preliminary review (although, the team was able to review the commit later).