# 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 for any defects, gaps in functionality, and possible. Ultimately, the team set out to optimize the user experience by logging, fixing, and re-testing our 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 tests have an advantage in terms of reusability (i.e. regression tests), the team decided that manual testing was just as important. Given the mathematical nature of the application, a plethora of possibilities exist for what the user can input. It would be an unwise rationing of resources to try and automate every possible 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

Units tests were created when possible in order to make our automated test coverage as vast as possible. We decided that unit tests are valuable since they are reusable tests and can be run at a later time (for example ,when regression needs to take place). TestNG was selected as the main unit testing framework since it has the same advantages as JUnit, plus a lot of other additional functionality. Furthermore, TestNG can be used for other types of tests such as functional and integration tests.

## Integration Testing

The plotter software was already integrated and working when this evaluation process was conducted. When code was changed, automated tests were run to make sure none of the software was impacted in a negative way. Manual tests were also conducted to make sure the program still functioned as intended. This ongoing manual and automated testing insured integration was maintained while bugs were being fixed.

## System Testing

After all bugs fixes had been implemented system testing was conducted to make sure the complete and integrated software met the specifications outlined in the requirements document. Manual testing was used to check each bug fix and to test multiple check points to ensure the software maintained its original functionality.

## Performance Testing

The plotter software is currently run as a java application locally. The plotter software functioned at a high-performance level under all testing conditions. None of the testers detected any lag or glitches in the calculations and rendering of graphs. At some point if the application is used on a website additional performance testing would be required to evaluate how the software performs when under varying load conditions.

## Regression Testing

Both manual and automated test cases were used for regression testing. When code changes were made (for either bug fixes or enhancements), automated test cases were run first (this took very little time) followed by the team members executing manual test cases. Two packages were created for automated testing both using the software testNG. The package called “Testing” ran automated tests on a sample object to make sure the object remained stable after fixes were implemented to the found bugs. The second package is called “test” and it ran automated tests for the color panel and for initializing the visualizer. To make sure all test cases are covered manually, the team used the test specification document to keep track of which tests to run though.

# Summary of Bug Fixes

## Bug Fix 1 (B07)

The Trapezium Integration calculation was returning an incorrect answer. The algorithm used to calculate the integral was incorrect. The code was updated with the correct Trapezium algorithm so the result being returned is now correct.

## Bug Fix 2 (B08)

The Simpson Integration calculation was returning an incorrect answer. The algorithm used to calculate the integral was incorrect. The code was updated with the correct Simpson algorithm so the result being returned is now correct.

## Bug Fix 3 (B09)

The Gauss Integration calculation was returning an incorrect answer. The algorithm used to calculate the integral was incorrect. The code was updated with the correct Gauss algorithm so the result being returned is now correct.

## Bug Fix 4 (B01)

On the three visualization types the label to the left of the function input was cut off. This bug was fixed by increasing the label size to show the full text for the label.

## Bug Fix 5 (B02)

The display range label was cut off on the side of the visualizer panel. This resulted in the text being cut off. To fix this buf the code was updated to increase the label size and allow the full text to be displayed.

## Bug Fix 6 (B03)

The display range label located on the Integral Panel was cut off. This resulted in the displayed text being cut off. To fix this bug the code was updated to widen the label’s dimensions to show the full text.

## Bug Fix 7 (B12)

When changing from any visualization to another the displayed function value resets to the default value of “sin(x)”. It is desired that the user entered function is retained when changing to another visualization. Due to the structure of the code it was not time efficient to complete this bug fix. The program is usable without this change.

## Bug Fix 8 (B)

When a file is saved, it is being saved without a file extension. The bug fix was to update the code to prompt the user to select a file extension before saving. The program still can save files without this fix and due to time constraints, this bug will be scheduled for fix at a future date.

# 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 lessons from the Plotter testing project. One realization that the team had is that in order to thoroughly test the Plotter application, many 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).