**Group 12**

**Iteration #4**

**Math Graphics v2.0.1**

**Software Test Plan**

Revision v1.00

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **REV** | **AUTHOR** | **DESCRIPTION** |
| 4/17/17 | 0.01 | ardecost | First Draft |
| 4/18/17 | 0.02 | nokerber | Updated test strategies. Updated sections 7-16. Added Appendix A for automated JUnit white-box test cases and Appendix B for system black-box test cases. |
| 4/22/17 | 0.03 | nokerber | Finished adding individual System black-box test cases to Appendix B. |
| 4/23/17 | 0.04 | nokerber | Added some JUnit white-box test cases related to the com.maths module to Appendix A. Also defined the naming convention for such test cases. |
| 4/23/17 | 0.05 | dpearson | Added JUnit white-box test cases related to the com module to Appendix A. |
| 4/23/17 | 1.00 | dpearson | Converted to PDF for submission. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1. Introduction 5](#_Toc480293619)

[1.1. Test Plan Objectives 5](#_Toc480293620)

[2. Scope 5](#_Toc480293621)

[2.1. User Entry 5](#_Toc480293622)

[2.2. Visualization 5](#_Toc480293623)

[2.3. Colors 5](#_Toc480293624)

[2.4. Save/Export 5](#_Toc480293625)

[3. Test Strategy 6](#_Toc480293626)

[3.1. System Test 6](#_Toc480293627)

[3.2. Performance Test 6](#_Toc480293628)

[3.3. Security Test 6](#_Toc480293629)

[3.4. Automated Test 6](#_Toc480293630)

[3.5. Stress and Volume Test 6](#_Toc480293631)

[3.6. Recovery Test 6](#_Toc480293632)

[3.7. Documentation Test 6](#_Toc480293633)

[3.8. Beta Test 6](#_Toc480293634)

[3.9. User Acceptance Test 7](#_Toc480293635)

[4. Environment Requirements 7](#_Toc480293636)

[4.1. Computer 7](#_Toc480293637)

[5. Test Schedule 7](#_Toc480293638)

[6. Control Procedures 7](#_Toc480293639)

[6.1 Reviews 7](#_Toc480293640)

[6.2 Bug Review meetings 7](#_Toc480293641)

[6.3 Change Request 7](#_Toc480293642)

[6.4 Defect Reporting 7](#_Toc480293643)

[7. Functions To Be Tested 8](#_Toc480293644)

[8. Resources and Responsibilities 8](#_Toc480293645)

[8.1. Resources 8](#_Toc480293646)

[8.2. Responsibilities 8](#_Toc480293647)

[9. Deliverables 9](#_Toc480293648)

[10. Suspension / Exit Criteria 10](#_Toc480293649)

[11. Resumption Criteria 10](#_Toc480293650)

[12. Dependencies 10](#_Toc480293651)

[12.1 Personnel Dependencies 10](#_Toc480293652)

[12.2 Software Dependencies 10](#_Toc480293653)

[12.3 Hardware Dependencies 10](#_Toc480293654)

[12.3 Test Data & Database 10](#_Toc480293655)

[13. Risks 10](#_Toc480293656)

[13.1. Schedule 10](#_Toc480293657)

[13.2. Technical 10](#_Toc480293658)

[13.3. Management 10](#_Toc480293659)

[13.4. Personnel 10](#_Toc480293660)

[13.5 Requirements 11](#_Toc480293661)

[14. Tools 11](#_Toc480293662)

[15. Documentation 11](#_Toc480293663)

[16. Approvals 11](#_Toc480293664)

[Appendix A – JUnit test cases 15](#_Toc480293665)

[Appendix B – Functional test cases 14](#_Toc480293666)

# 1. Introduction

The software testing team has been tasked with testing the Math Graphics 2.0.1 program which has been written in Java. This document serves as the software test plan for the program.

## Test Plan Objectives

This Test Plan for the Math Graphics program supports the following objectives:

* Setup and execute the software product, understand how the software works.
* Use the software as an end-user to gain operational familiarity.
* Communicate to the team the System Test strategy.
* Define deliverables and responsible parties.
* Communicate to all team members the various Dependencies and Risks

# 2. Scope

## 2.1. User Entry

The program allows the user to enter a function as input and have the graph displayed in the desired coordinate plane.

## Visualization

The program allows the user to display graphs in the formats listed below:

* Cartesian 2D
* Polar 2D
* Cartesian 3D

The user has the ability to zoom in/out, pan up/down/left/right and choose the desired display range.

## Colors

The user has the ability to change and save the colors of the following elements:

* Background
* Panel
* Line 1
* Line 2
* Axis
* Line 3D

## 2.4. Save/Export

The user has the ability to save an image of the displayed graph or export the data to file.

# Test Strategy

The test strategy consists of a series of different tests that will fully exercise the graphics program. The primary purpose of these tests is to uncover the systems limitations and faults. A list of the various planned tests and a brief explanation follows below.

## 3.1. System Test

The System tests will focus on the behavior of the math graphics system. User scenarios will be executed against the system as well as screen mapping and error message testing. Overall, the system tests will test the integrated system and verify that it meets the requirements defined in the requirements document. For individual system test cases, see Appendix B.

## 3.2. Performance Test

Performance tests will be conducted to ensure that the graph’s drawing response times meet the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress of the calculator modules.

## 3.3. Security Test

Security tests are omitted and not needed as no user authentication is required to use the program.

## 3.4. Automated Test

A suite of automated tests will be developed to test the functionality of the math graphics program and perform regression testing on areas of the systems that previously had critical/major defects. Adequate coverage of the program’s code will be achieved and verified using the EclEmma tool. The suite will be created with JUnit and individual test cases can be found in Appendix A.

## 3.5. Stress and Volume Test

We will subject the math graphics system to large input conditions, or functions. This will put stress on the math calculator modules.

## 3.6. Recovery Test

Recovery tests are omitted and not needed because it’s expected that the displayed function will not save when the program is closed and if data is not exported.

## 3.7. Documentation Test

Tests will be conducted to check the accuracy of the user documentation. These tests will ensure that no features are missing, and the contents can be easily understood.

## 3.8. Beta Test

The Engineering team will beta tests the math graphics system and will report any defects they find. This will subject the system to tests that could not be performed in our test environment.

## 3.9. User Acceptance Test

Once the math graphics system is ready for public release, the engineering group will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

# 4. Environment Requirements

## 4.1. Computer

The minimum system requirements shown below were taken from Oracle for Java 8 and are also found at: java.com.

* Java compatible Windows, MAC or Linux computer.
* (Minimum) Pentium 2 266 MHz processor
* 128MB RAM
* 124 MB for JRE; 2 MB for Java Update
* Java JDK
* Eclipse IDE
* Math Graphics 2.0.1 source code
* Math Graphics 2.0.1 generated .jar file

# 5. Test Schedule

* Ramp up/System familiarization 4/21/2017 – 4/23/2017
* System Test 4/23/2017 – 4/29/2017
* Beta Test 4/29/2017 – 4/30/2017
* User Acceptance Test 4/29/2017 – 4/30/2017

# 6. Control Procedures

## 6.1 Reviews

The project team will perform reviews for each Phase. (i.e. Requirements Review, Design Review, Code Review, Test Plan Review, Test Case Review and Final Test Summary Review). Will be discussed in online meeting, using Flowdock. All team members will be emailed invites.

## 6.2 Bug Review meetings

Regular online meetings (Flowdock) will be held to discuss reported defects. The development department will provide status/updates on all defects reported and the test department will provide additional defect information if needed. All members of the project team will participate.

## 6.3 Change Request

Once testing begins, changes to the graphics program are discouraged. If functional changes are required, these proposed changes will be discussed with group. The group will determine the impact of the change and if/when it should be implemented.

## 6.4 Defect Reporting

When defects are found, the testers will complete a defect report on the defect tracking system, JTrac. The defect tracking system is accessible by all members of the team. When a defect has been fixed or more information is needed, we can change the status of the defect to indicate the current state. Once a defect is verified as fixed, the defect report can be closed.

# 7. Functions To Be Tested

The following is a list of functions that will be tested:

* Load the program GUI
* Enter a function to display.
* Draw the function, on the different planes.
* Colors features
* Export data features
* Control features (move up, down, left, right, zoom in/out)
* Chage/Display domain of the function (DF)
* Calculate Integral

A Requirements Validation Matrix will “map” the test cases back to the requirements. See Deliverables.

# 8. Resources and Responsibilities

The Test Lead and Project Manager will determine when system test will start and end. The Test lead will also be responsible for coordinating schedules, equipment, & tools for the testers as well as writing/updating the Test Plan, Weekly Test Status reports and Final Test Summary report. The testers will be responsible for writing the test cases and executing the tests. With the help of the Test Lead, the testers will be responsible for the Beta and User Acceptance tests.

## 8.1. Resources

The test team will consist of:

* A Project Manager
* A Test Lead
* 3 Testers

## Responsibilities

|  |  |
| --- | --- |
| Project Manager | Responsible for Project schedules and the overall success of the project. Participate on CCB. |
|  |  |
| Lead Developer | Serve as a primary contact/liaison between the development department and the project team.  Participate on CCB. |
|  |  |
|  |  |
| Test Lead | Ensures the overall success of the test cycles. He/she will coordinate weekly meetings and will communicate the testing status to the project team.  Participate on CCB. |
|  |  |
| Testers | Responsible for performing the actual system testing. |
|  |  |
|  |  |
|  |  |
|  |  |

# Deliverables

|  |  |  |
| --- | --- | --- |
| **Deliverable** | **Responsibility** | **Completion Date** |
|  |  |  |
| Develop Test cases | Testers | 4/20/2017 |
|  |  |  |
| Test Case Review | Test Lead, Dev. Lead, Testers | 4/23/2017 |
|  |  |  |
| Develop Automated test suites | Testers | 4/22/2017 |
|  |  |  |
| Requirements Validation Matrix | Test Lead | 4/23/2017 |
|  |  |  |
| Execute manual and automated tests | Testers & Test Lead | 4/29/2017 |
|  |  |  |
| Complete Defect Reports | Everyone testing the product | On-going |
|  |  |  |
| Document and communicate test status/coverage | Test Lead | Weekly |
|  |  |  |
| Execute Beta tests | Testers | 4/30/2017 |
|  |  |  |
| Document and communicate Beta test status/coverage | Test Lead | 4/30/2017 |
|  |  |  |
| Execute User Acceptance tests | Testers | 4/30/2017 |
|  |  |  |
| Document and communicate Acceptance test status/coverage | Test Lead | 4/30/2017 |
|  |  |  |
| Final Test Summary Report | Test Lead | 5/02/2017 |

# 10. Suspension / Exit Criteria

If any defects are found which seriously impact the test progress, the QA manager may choose to

Suspend testing. Criteria that will justify test suspension are:

* Hardware/software is not available at the times indicated in the project schedule.
* Source code contains one or more critical defects, which seriously prevents or limits testing progress.
* Assigned test resources are not available when needed by the test team.

# Resumption Criteria

If testing is suspended, resumption will only occur when the problem(s) that caused the suspension has been resolved. When a critical defect is the cause of the suspension, the “FIX” must be verified by the test department before testing is resumed.

# Dependencies

## Personnel Dependencies

The test team requires experience testers to develop, perform and validate tests. The test team will also need the following resources available: Application developers.

## Software Dependencies

The source code must be unit tested and provided within the scheduled time outlined in the Project Schedule.

## 12.3 Hardware Dependencies

The testers’ workstations (with specified hardware/software) need to be available during normal working hours. Any downtime will affect the test schedule.

## Test Data & Database

Test data, functions, should also be made available to the testers for use during testing.

# Risks

## 13.1. Schedule

The schedule for each phase is very aggressive and could affect testing. A slip in the schedule in one of the other phases could result in a subsequent slip in the test phase. Close project management is crucial to meeting the forecasted completion date.

## 13.2. Technical

No Technical risks at this time.

## 13.3. Management

Management support is required so when the project falls behind, the test schedule does not

get squeezed to make up for the delay. Management can reduce the risk of delays by supporting the test team throughout the testing phase and assigning people to this project with the required skills set.

## 13.4. Personnel

Due to the aggressive schedule, it is very important to have experienced testers on this project. Unexpected turnovers can impact the schedule. If attrition does happen, all efforts must be made to replace the experienced individual.

## 13.5 Requirements

The test plan and test schedule are based on the current Requirements Document. Any changes to the requirements could affect the test schedule and will need to be approved by the CCB.

# 14. Tools

The JUnit Automated test tool will be used to help test the math graphics program. All of the testers have been trained on and experience with the use of this test tool. The JTrac defect tracking tool will also be used to keep track of defects that arise during all phases of testing.

# Documentation

The following documentation will be available at the end of the test phase:

* Test Plan
* Test Cases
* Test Case review
* Requirements Validation Matrix
* Defect reports
* Final Test Summary Report

# 16. Approvals

|  |  |  |
| --- | --- | --- |
| **Name (Print)** | **Signature** | **Date** |
|  |  |  |
| **1.** |  |  |
|  |  |  |
| **2.** |  |  |
|  |  |  |
| **3.** |  |  |
|  |  |  |
| **4.** |  |  |
|  |  |  |
| **5.** |  |  |

# Appendix A – Automated JUnit test cases

This section describes the individual automated white-box JUnit test cases. These tests will be ran against the individual methods in the source code as part of an automated test suite. Please note that not all methods have a test case associated with the test suite because the methods chosen utilize the most objects and other methods as possible to achieve greater coverage. Also, coverage is maximized through the use of System black-box testing, especially for the graphics.

A test case ID naming system is used for these system tests. Here is the format of the test case ID’s: <X>\_n

<X> - This is used to show what module the test cases belong to. Here are the options:

C – The Com module

M – The Com.Main module

T – The Com.Math module

n – This is the test case number, starting at 001.

So an example test case ID might be: T\_003 which, using the legend above, indicates that it is the third test case in the com.math module.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project Name:** | **Plotter** | **Test Designed by:** | **nokerber** |  |  |  |
| **Module Name:** | **com.maths** | **Test Designed date:** | 4/23/2017 |  |  |  |
|  |  | **Test Executed by:** |  |  |  |  |
|  |  | **Test Execution date:** |  |  |  |  |
|  |  |  |  |  |  |  |
| **Preconditions:** | Required objects built, test suite created. | |  |  |  |  |
|  |  |  |  |  |  |  |
| **ID** | **Class** | **Method** | **Test Data** | **Expected Result** | **Actual Result** | **Status** |
| T\_001 | Calculator | getFunction3D | no inputs | no assertions |  |  |
| T\_002 | Calculator | calculateDerivativeFunction | no inputs | no assertions |  |  |
| T\_003 | Calculator | f | double, double | no assertions |  |  |
| T\_004 | Adv Calculator | SimpsonIntegral | Calculator | no assertions |  |  |
| T\_005 | Adv Calculator | trapezeumIntegral | Calculator | no assertions |  |  |
| T\_006 | Adv Calculator | gaussIntegral | Calculator | no assertions |  |  |
| T\_007 | MathTree | evaluate | Tnode, double, double | no assertions |  |  |
| T\_008 | MathTree | evaluateFunction | String | no assertions |  |  |
| T\_009 | ParseFunction | replaceFunction | int, int | no assertions |  |  |
| T\_010 | ParseFunction | calculateArgument | String | no assertions |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project Name:** | **Plotter** | **Test Designed by:** | **dpearson** |  |  |  |
| **Module Name:** | **com** | **Test Designed date:** | 4/23/2017 |  |  |  |
|  |  | **Test Executed by:** |  |  |  |  |
|  |  | **Test Execution date:** |  |  |  |  |
|  |  |  |  |  |  |  |
| **Preconditions:** | Required objects and test suite created. | |  |  |  |  |
|  |  |  |  |  |  |  |
| **ID** | **Class** | **Method** | **Test Data** | **Expected Result** | **Actual Result** | **Status** |
| C\_001 | LineData | buildPolygon(LineData, Vector) | LineData, Vector | no assertions |  |  |
| C\_002 | Polygon3D | divideIntoTriangle(Polygon3D) | Polygon3D | no assertions |  |  |
| C\_003 | Polygon3D | extractSubPolygon3D(Polygon3D, int, int) | Polygon3D, int, int | no assertions |  |  |
| C\_004 | Polygon3D | fromAreaToPolygond2D(Area) | Area | no assertions |  |  |
| C\_005 | Point3D | calculateCosin(Point3D, Point3D) | Point3D, Point3D | no assertions |  |  |
| C\_006 | Point3D | distance(double, double, double, double, double, double) | double, double, double, double, double, double | no assertions |  |  |
| C\_007 | Point3D | calculateCrossProduct(Point3D, Point3D) | Point3D, Point3D | no assertions |  |  |
| C\_008 | Point3D | foundPXIntersection(Point3D, Point3D, double) | Point3D, Point3D, double | no assertions |  |  |
| C\_009 | Zbuffer | fromColorToHex(Color) | Color | no assertions |  |  |
| C\_010 | Renderer3D | interpolatePhongNormal(Point3D, Point3D, double) | Point3D, Point3D, double | no assertions |  |  |
| C\_011 | Renderer3D | newLine(Point3D, Point3D) | Point3D, Point3D | no assertions |  |  |
| C\_012 | Renderer3D | getRotationMatrix(Point3D, double) | Point3D, double | no assertions |  |  |
| C\_013 | Renderer3D | calcAssX(Point3D) | Point3D | no assertions |  |  |

# 

# Appendix B – System test cases

This section describes all of the individual system black-box test cases. These tests will be ran against the .jar file of the program, so the tester doesn’t know the exact code that’s being tested. Instead, the program is tested as a whole, as if an end-user was using the program.

A test case ID naming system is used for these system tests. Here is the format of the test case ID’s: S\_<X>\_n

S – Signifies that this is a System test case.

<X> - This is used to show what category the test case belongs to. Here are the options:

F – Indicates File I/O.

D – Display

M – Math

n – This is the test case number, starting at 001.

So an example test case ID might be: S\_D\_003 which, using the legend above, indicates that it is a System test, of the Display category, and is the 003rd test case in that category.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Scenario** | **Test Case** | **Pre-Conditions** | **Test Step** | **Test Data** | **Expected Result** | **Actual Results** | **Pass/Fail** |
| S\_F\_001 | File I/O | Open Program | Math Graphics .jar file on a supported system. | Launch the program, execute the .jar file. | The .jar file. | The program opens and displays the 2D Cartesian plane default function y = sin(x). Ensure there are no exceptions or errors during launch. |  |  |
| S\_F\_002 | File I/O | Close Program | Program should already be open. | Click on "Do", then select "Exit". | See Test Step. | The program exits cleanly, ensure there's no exceptions or errors. |  |  |
| S\_F\_003 | File I/O | Save Graph | Program should already be open on Cartesian 2D Plane. | 1) Put test data in "displayed function" box.  2) Click "Draw"  3) Click "Save", then "Save Image". | 2\*sin(x) | Ensure the program generates an image file with the displayed graph. |  |  |
| S\_F\_004 | File I/O | Export Graph Data | Program should already be open on Cartesian 2D Plane. | 1) Put test data in "displayed function" box.  2) Click "Draw"  3) Click "Save", then "Export Data".  4) Enter the filename to save, then click "Save". | 2\*x + 1 | Ensure the program generates a text file with coordinate information on each line separated by a comma. |  |  |
|  |  |  |  |  |  |  |  |  |
| S\_D\_001 | Display | Validate Arrow Keys | Program should already be open on 2D Cartesian Plane | Use test data on 2D Cartesian, 2D Polar, and then 3D Cartesian planes. | Left, Right, Up, Down arrow keys. | The view of the graph will move L, R, U, D, based on which arrow key is pressed. |  |  |
| S\_D\_002 | Display | Validate Mouse Wheel | Program should already be open on 2D Cartesian Plane | Use test data on 2D Cartesian, 2D Polar, and then 3D Cartesian planes. | Mouse Wheel up and down. | The view of the graph will move up or down based on which direction the mouse wheel is moved. |  |  |
| S\_D\_003 | Display | Validate Mouse Drag | Program should already be open on 2D Cartesian Plane | Use test data on 2D Cartesian, 2D Polar, and then 3D Cartesian planes. | Use the mouse to drag the graph in different directions. | The view of the graph will move based on which direction the mouse is used to drag the graph. |  |  |
| S\_D\_004 | Display | Graph Zoom | Program should already be open on 2D Cartesian Plane | Use test data on 2D Cartesian, 2D Polar, and then 3D Cartesian planes. | Use the "+" and "-" buttons at the bottom of the GUI. | The view of the graph will zoom in and zoom out, depending on which related button was pressed. |  |  |
| S\_D\_005 | Display | Colors - Background | Program should already be open on 2D Cartesian Plane | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save | Choose a different background color. | The view of the graph should be that background color now. |  |  |
| S\_D\_006 | Display | Colors - Panel | Program should already be open on 2D Cartesian Plane | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save | Choose a different panel color. | The view of the program panel should be that new color now. |  |  |
| S\_D\_007 | Display | Colors - Line | Program should already be open on 2D Cartesian Plane | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save | Choose a different line color. | The view of the graph function line should be that new color now. |  |  |
| S\_D\_008 | Display | Colors - Line2 (DF) | Program should already be open on 2D Cartesian Plane | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save 6) Click on Show DF | Choose a different line2 color. | The view of the graph DF line should be that new color now. |  |  |
| S\_D\_009 | Display | Colors - Axis | Program should already be open on 2D Cartesian Plane | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save | Choose a different axis color. | The view of the graph axis should be that new color now. |  |  |
| S\_D\_010 | Display | Colors - Line 3D | Program should already be open on 3D Cartesian Plane with function sin(x+y) | 1) Colors 2) Change Colors 3) Select test data. 4) Click Ok 5) Click Save | Choose a different Line 3D color | The view of the 3D Cartesian graph 3D Line should be that new color now. |  |  |
|  |  |  |  |  |  |  |  |  |
| S\_M\_001 | Math | Draw - 2D Cartesian, null input | Program should already be open on 2D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | null input | There should be no errors, exceptions, and nothing displayed on the graph. |  |  |
| S\_M\_002 | Math | Draw - 2D Cartesian, error input | Program should already be open on 2D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | 3x + 1 | The correct format for test data is "3\*x + 1", so an error should be generated. |  |  |
| S\_M\_003 | Math | Draw - 2D Cartesian, valid input | Program should already be open on 2D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | 3\*x + 1 | The function should be displayed correctly on the graph. |  |  |
| S\_M\_004 | Math | Draw - 2D Polar, null input | Program should already be open on 2D Polar Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | null input | There should be no errors, exceptions, and nothing displayed on the graph. |  |  |
| S\_M\_005 | Math | Draw - 2D Polar, error input | Program should already be open on 2D Polar Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | 3sin(theta) | The correct format for test data is "3\*sin(theta)", so an error should be generated. |  |  |
| S\_M\_006 | Math | Draw - 2D Polar, valid input | Program should already be open on 2D Polar Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | 3\*sin(theta) | The function should be displayed correctly on the graph. |  |  |
| S\_M\_007 | Math | Draw - 3D Cartesian, null input | Program should already be open on 3D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | null input | There should be no errors, exceptions, and nothing displayed on the graph. |  |  |
| S\_M\_008 | Math | Draw - 3D Cartesian, error input | Program should already be open on 3D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | sin(x)sin(y) | The correct format for test data is "sin(x)\*sin(y)", so an error should be generated. |  |  |
| S\_M\_009 | Math | Draw - 3D Cartesian, valid input | Program should already be open on 3D Cartesian Plane | 1) Enter test data in "Displayed function y=" text box. 2) Click "Draw". | sin(x)\*sin(y) | The function should be displayed correctly on the graph. |  |  |
| S\_M\_010 | Math | Range - 2D Cartesian, null input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | null input | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_011 | Math | Range - 2D Cartesian, invalid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | a) -10 b) Z | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_012 | Math | Range - 2D Cartesian, reversed input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | a) 10 b) -10 | Ideally, the program should autocorrect the reversed inputs and still display the graph from -10 to 10. |  |  |
| S\_M\_013 | Math | Range - 2D Cartesian, valid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | a) -10 b) 10 | The program should update and show the graph with the new range. |  |  |
| S\_M\_014 | Math | Range - 2D Polar, null input | Program should already be open on 2D Polar Plane and r(theta) = 5\*cos(theta) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | null input | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_015 | Math | Range - 2D Polar, invalid input | Program should already be open on 2D Polar Plane and r(theta) = 5\*cos(theta) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | theta1) -10 theta2) Z | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_016 | Math | Range - 2D Polar, reversed input | Program should already be open on 2D Polar Plane and r(theta) = 5\*cos(theta) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | theta1) 10 theta2) -10 | Ideally, the program should autocorrect the reversed inputs and still display the graph from -10 to 10. |  |  |
| S\_M\_017 | Math | Range - 2D Polar, valid input | Program should already be open on 2D Polar Plane and r(theta) = 5\*cos(theta) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | theta1) -10 theta2) 10 | The program should update and show the graph with the new range. |  |  |
| S\_M\_018 | Math | Range - 3D Cartesian, null input | Program should already be open on 3D Cartesian Plane and z(x,y) = sin(x+y) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | null input | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_019 | Math | Range - 3D Cartesian, invalid input | Program should already be open on 3D Cartesian Plane and z(x,y) = sin(x+y) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | ax) 10 bx) Z ay) 10 by) X | The displayed function shouldn't change since an invalid range input was used. |  |  |
| S\_M\_020 | Math | Range - 3D Cartesian, reversed input | Program should already be open on 3D Cartesian Plane and z(x,y) = sin(x+y) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | ax) 10 bx) -10 ay) 10 by) -10 | Ideally, the program should autocorrect the reversed inputs and still display the graph from -10 to 10. |  |  |
| S\_M\_021 | Math | Range - 3D Cartesian, valid input | Program should already be open on 3D Cartesian Plane and z(x,y) = sin(x+y) graphed. | 1) Enter test data in "Displayed Range" for a and b.  2) Click "Draw". | ax) -10 bx) 10 ay) -10 by) 10 | The program should update and show the graph with the new range. |  |  |
| S\_M\_022 | Math | DF line - 2D Cartesian | Program should already be open on 2D Cartesian Plane and y=0.25 \* x^2 graphed with range -10,10. | 1) Click the DF Button | DF Button | A new line should be displayed that shows the DF. |  |  |
| S\_M\_023 | Math | Integral for 2D Cartesian - Trapezium null input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: leave null x2: leave null select: "Trapezium" | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_024 | Math | Integral for 2D Cartesian - Trapezium invalid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: A select: "Trapezium" | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_025 | Math | Integral for 2D Cartesian - Trapezium valid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: 5 select: "Trapezium" | Ensure the Integration calculation is correct in the uneditable "Res:" text field box. |  |  |
| S\_M\_026 | Math | Integral for 2D Cartesian - Simpson null input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: leave null x2: leave null select: Simpson | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_027 | Math | Integral for 2D Cartesian - Simpson invalid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: A select: Simpson | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_028 | Math | Integral for 2D Cartesian - Simpson valid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: 5 select: Simpson | Ensure the Integration calculation is correct in the uneditable "Res:" text field box. |  |  |
| S\_M\_029 | Math | Integral for 2D Cartesian - Gauss null input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: leave null x2: leave null select: Gauss | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_030 | Math | Integral for 2D Cartesian - Gauss invalid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: A select: Gauss | The calculated value in the "Res:" text field box shouldn't change, no calculation should be made. |  |  |
| S\_M\_031 | Math | Integral for 2D Cartesian - Gauss valid input | Program should already be open on 2D Cartesian Plane and y=sin(x) graphed with range -10,10. | 1) Click the Integral button 2) Enter test data 3) Click "recalculate" | x1: -5 x2: 5 select: Gauss | Ensure the Integration calculation is correct in the uneditable "Res:" text field box. |  |  |