SE 3XA3: Test Report
Title of Project

Team #, Team Name
Student 1 name and macid
Student 2 name and macid
Student 3 name and macid

December 6, 2016

Contents

1 Functional Requirements Evaluation **1**

1.0.1 Area of Testing 1 1

1.0.2 Area of Testing 2 2

1.0.3 Area of Testing 3 3

1.0.4 Area of Testing 4 4

2 Nonfunctional Requirements Evaluation **5**

2.1 Usability 5

2.2 Performance 5

2.3 etc. 5

3 Comparison to Existing Implementation **5**

4 Unit Testing **5**

5 Changes Due to Testing **5**

6 Automated Testing **6**

6.0.1 Area of Testing 5 6

7 Trace to Requirements **7**

8 Trace to Modules **7**

9 Code Coverage Metrics **7**

List of Tables

1 **Revision History** 1

2 Trace Between Tests and Requirements 8

3 Trace Between Modules and Tests 8

List of Figures

This document ...

1 Functional Requirements Evaluation

1.0.1 Area of Testing 1

Requirement #1: The software shall read data given to it.
Requirements #4: The software will plot all the data points.

1. **Test ID #1.1**
   Type: Functional, Dynamic, Manual
   Initial State: Instantiate Graph(6) object with 6 markings on each quadrant.
   Input: The list object: [(1, 1), (2, 2), (3, 3), (4, 4)]
   Expected Output: A window depicting a graph with plotted points at (1, 1), (2, 2), (3, 3), and (4, 4).
   Output: A graph with the points (1,1), (2,2), (3,3) and (4,4) was created.
   Result: PASS

2. **Test ID #1.2**
   Type: Functional, Dynamic, Manual
   Initial State: None Object.
   Input: Instantiate Graph(6, data = [(1, 1), (2, 2), (3, 3), (4, 4)])
   Expected Output: A window depicting a graph with plotted points at (1, 1), (2, 2), (3, 3), and (4, 4).
   Output: A graph with the points (1,1), (2,2), (3,3) and (4,4) was created.
   Result: PASS

3. **Test ID #1.3**
   Type: Functional, Dynamic, Manual

Table 1: **Revision History**

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

Initial State: Instantiate Graph(6) object with 6 markings on each quadrant.
Input: The list object: [ ].
Expected Output: A window depicting a graph with no plotted points.
Output: An empty graph was outputted.
Result: PASS

4. **Test ID #1.4**
   Type: Functional, Dynamic, Manual
   Initial State: None Object.
   Input: Instantiate Graph(6, data = [ ])
   Expected Output: A window depicting a graph with no plotted points.
   Output: An empty graph was outputted.
   Result: PASS

1.0.2 Area of Testing 2

Requirement #2: The software will raise an exception if the data format cannot be plotted, and stop the program.

1. **Test ID #2.1**
   Type: Functional, Dynamic, Manual
   Initial State: A testing script that imports the method that validates data.
   Input: The list object: [(1, 1), (2, 2), (3, 3), (4, 4)]
   Expected Output: A safe end to the execution (i.e, no exception raised).
   Output: The program completed execution.
   Result: PASS

2. **Test ID #2.2**
   Type: Functional, Dynamic, Manual
   Initial State: A testing script that imports the method that validates data.
   Input: The list object: [(1, 1), (2, 2), (3, 3), (4)]
   Expected Output: The program raised an exception.

Output: An exception was raised with output in terminal as "Exception: Inconsistent data"
Result: PASS

3. **Test ID #2.3**
Type: Functional, Dynamic, Manual
Initial State: Instantiate Graph(6) object with 6 markings on each quadrant..
Input: Instantiate Graph(6, data = [(1, 1), (2, 2), (3, 3), (4)])
Expected Output: The program raised an exception.
Output: An exception was raised with output in terminal as "Exception: Inconsistent data"
Result: PASS

1.0.3 Area of Testing 3

Requirement #3: The software will construct a coordinate system that will fit all the data points.

1. **Test ID #3.1**
Type: Functional, Dynamic, Manual
Initial State: Instantiate Graph(6) object with 6 markings on each quadrant.
Input: The list object: [(1, 1), (2, 2), (3, 3), (17, 77)]
Expected Output: A window depicting a graph with max x axis value to be 17 and -17, and y axis to include 77 and -77.
Output: Max y axis was still 6 and -6
Result: FAIL

2. **Test ID #3.2**
Type: Functional, Dynamic, Manual
Initial State: Instantiate Graph(6) object with 6 markings on each quadrant.
Input: The list object: [(1, 1), (2, 2), (3, 3), (-17, -77)]
Expected Output: A window depicting a graph with max x axis value to include 17 and -17, and y axis to include 77 and -77.
Output: Max y axis was still 6 and -6
Result: FAIL

Two more test cases were added to validate the functionality the above test cases were trying to test.

3. **Test ID #3.3**
   Type: Functional, Dynamic, Manual
   Initial State: Instantiate Graph(6) object with 6 markings on each quadrant.
   Input: The list object: [(1, 1), (2, 2), (3, 3), (17, 77)]
   Expected Output: A window depicting a graph with max x axis value include 17 and -17, and y axis to include 77 and -77.
   Output: A graph with all the points plotted, and an x axis from -18 to 18, and y axis from -78 to 87
   Result: PASS

4. **Test ID #3.4**
   Type: Functional, Dynamic, Manual
   Initial State: None object
   Input: Instantiated Graph(6, [(1, 1), (2, 2), (3, 3), (-17, -77)]
   Expected Output: A window depicting a graph with max x axis value include 17 and -17, and y axis to include 77 and -77.
   Output: A graph with all the points plotted, and an x axis from -18 to 18, and y axis from -78 to 87
   Result: PASS

1.0.4 Area of Testing 4

Requirement #5: The software will connect a line that passes through all the data points if the data points are a function of x.

1. **Test ID #4.1**
   Type: Functional, Dynamic, Manual
   Initial State: An instantiated Graph object.
   Input: The default math.sin() function in python.
   Expected Output: A window depicting the sin() graph.
   Output: A graph with the sin wave plotted
   Result: PASS

2. **Test ID #4.2**
   Type: Functional, Dynamic, Manual

Initial State: An instantiated Graph object.
Input: The list object: [(1, 1), (2, 2), (3, 3), (4, 4)] on the method plot_points_with_line()
Expected Output: A window depicting a graph with the points plotted, and a line is connecting all points.
Output: A line passing through the points in the input
Result: PASS

3. **Test ID #4.3**
Type: Functional, Dynamic, Manual
Initial State: An instantiated Graph object.
Input: The list object: [(1, 1), (2, 2), (3, 3), (3, 4)] on the method plot_points_with_line().
Expected Output: The software will plot the points but not connect a line through them.
Output: The points were plotted, but the line was not drawn.
Result: PASS

2 Nonfunctional Requirements Evaluation

2.1 Usability

2.2 Performance

2.3 etc.

3 Comparison to Existing Implementation

This section will not be appropriate for every project.

4 Unit Testing

5 Changes Due to Testing

As seen in the Area of Testing 3.1 and 3.2 for Functional Requirements Evaluation, the test cases had failed to produce the expected output.

After manual code evaluation, it was realized by the developers that this was because an instantiated graph defaults to drawing the axes under the assumption of the data given to it. Because the test case was instantiating the Graph object with NoneType data, the graph would default to drawing the axes with a scale of 1 (therefore, the max values of magnitude 6, for the 6 markings). When calling the plot_...() methods, it would draw on a canvas already painted in the initializer.

To fix this, a few extra lines were added to the plot_...() methods where they would call the init() method to reinitialize the graph canvas and paint it with the appropriate axes. **The two test cases passed once these changes were complete.**

6 Automated Testing

Because of the timeframe of the project, the extent of the automated testing was unit testing, and testing of source code (seen in the test case below) using the online tool Pylint.

6.0.1 Area of Testing 5

Testing of source code.

1. **Test ID #5.1**
   Type: Functional, Static, Automated
   Initial State: Completed source code.
   Input: Source code.
   Expected Output: Review of source code.
   Output: log-file.
   Result: N/A

It would not have been feasible to do bitmap comparison automated testing for this project because to do so correctly and accurately would have been out of the scope of the project. The developers verified the outputs of the graphs manually to realize that the behavior was up to expectations, and the challenge in the nature of this sort of automated testing would not have been worth the value compared to the time dedicated to it, especially to realize an outcome that was already manually verified.

7 Trace to Requirements

8 Trace to Modules

9 Code Coverage Metrics

References

Test	**Requirement**
Functional Requirements Testing	
1	R1, R4
2	R2
3	R3
4	R5
Non-functional Requirements Testing	
Automated Testing	

Table 2: Trace Between Tests and Requirements

Test	**Module**
Functional Requirements Testing	
1	M1, M3.1, M3.3, M2.2, M1.1
2	M2.1
3	M3.3
4	M3.2, M1.1
Non-functional Requirements Testing	
Automated Testing	

Table 3: Trace Between Modules and Tests