Software Test Report

for

Prime Factor Calculator

(abridged and adapted for COS80010 assignment purposes)

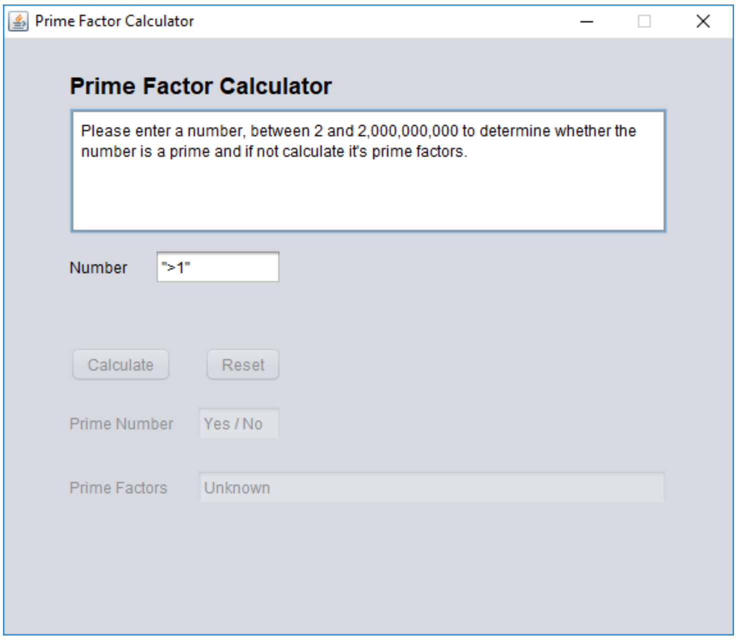
Table of Contents

Section 1 Introduction

A prime factor is a number that is a factor of a number and is also a prime. It is important to note that all real numbers greater than one, either have a set of prime factors or are themselves a prime number. For example, 12 has the prime factors {2, 2, 3} as they're all prime numbers and multiplied together gives 12. The prime factor calculator is a small program written in Java, that takes a given input number and calculates all the prime factors of the number

The program does this by first calculating all the possible prime numbers up to a given number which is the upper bounds for possible factors, this is done using the Sieve of Erastophenes algorithm for more efficient calculation. Once this is done, an algorithm is employed to determine which of the prime numbers are factors. There are two possible results as described above, either it is a prime or it is made up of prime factors.

The program is interactable through a graphical user interface (GUI), which is shown below.



Please enter a number, between 2 and 2,000,000,000 to determine whether the number is a prime and if not calculate it's prime factors.

Number ">1"

Prime Number Yes / No

Prime Factors Unknown

Figure 1 Prime Number Calculator GUI

The aim of this document is to explain the testing of the program. First the test plan is briefly introduced, then the methods used to automate the testing are described, then a summary of results are presented followed finally by a more detailed discussion of results.

Section 2 Test Plan

As this project is different to the one I submitted for Assignment 1, I have briefly outlined a test plan to clarify the following test case result sections.

Section 2.1 Requirements under test

This testing plan will test one functional and one non-functional requirement. These are tabulated below.

Requirement ID	Description
F1	Program must determine if a number is prime, or the number's prime factors through the use of a GUI interface.
N1	The program must calculate the prime factors of a large number relatively quickly. E.g. O(10^9) input should not take more than 20 seconds.

These requirements can be tested via the following test plans that have been identified using a black box testing approach. The following sections outline the different areas to be tested in this application.

The application is made up 3 major classes. The PrimeNumberFinder; which calculates a list of prime numbers to a given number, the PrimeFactorFinder; which takes a number and determines its prime factors and the Prime_GUI; which contains the logic for the user interface.

As the PrimeFactorFinder is linked intrinsically to the PrimeNumberFinder it will have to be unit tested via mocking the PrimeNumberFinder class within it.

The GUI implementation will only be considered during overall system testing.

Section 2.2 Unit Testing

Section 2.2.1 Definition

To generate the test cases for the PrimeNumberFinder the concept of equivalence partitioning (EP) and boundary value analysis (BVA) has been used. The PrimeNumberFinder constructor will accept values from the range of [2 <= x <= 1,000,000,000] which is one equivalence class, with 4 non-equivalent classes being identified. [x < 0], [x = 0], [x = 1], [x > 1,000,000,000]. Hence there will be four different tests to account for these methods. Along with boundary values analysis to ensure that the boundaries are enforced properly.

To ensure that primes are being calculated correctly 3 different numbers will be tested in the method and compared with their expected results. These equate to the two boundary values, 2 and 1,000,000,000 and one other which covers the equivalence class of valid number. Due to inability to find a comprehensive list of prime numbers on the internet up to 1,000,000,000 only the last prime is considered to see if this test case works. And another test case of input number 7920 is used as well to verify a large number's results.

For the PrimeFactorFinder, similarly BVA and EP has been used to determine the test cases. For the constructor this means similar values to above except the increased max value to 2,000,000,000. The PrimeFactorFinder constructor will accept values from the range of [2 <= x <= 2,000,000,000] which is one equivalence class, with 4 non-equivalent classes being identified. [x < 0], [x = 0], [x = 1], [x > 2,000,000,000]. Hence there will be four different tests to account for these methods. Along with boundary values analysis to ensure that the boundaries are enforced properly.

There are then two valid equivalence classes for the CalculatePrimeFactors method. Either a number can be prime, or a number can have a list of prime factors. This is then separated into a general category of each, and a lower and high bounds case as well.

Section 2.2.2 Overview of Items to be Tested

Item to be tested	Test Description	Test Date
PrimeNumberFinder Constructor	Tests the constructor of PrimeNumberFinder	4/6/2019
PrimeNumberFinder CalculatePrimes	Test that all primes up to the given number are calculated correctly.	4/6/2019
PrimeFactorFinder Constructor	Tests the constructor of PrimeFactorFinder	5/6/2019
PrimeFactorFinder CalculatePrimeFactors	Test that the program is correctly calculating it's prime factors of it's a prime number.	5/6/2019

Section 2.2.3 Unit Test Plan for PrimeNumberFinder Constructor

Test id	Input /Parameter Values	Expected Output / Returned Value	Reason for the test	Actual Output / Returned Value	Correct (Yes / No)	Seriousness of the Error
1	InputNumber : 0	True when IllegalArgument exception caught.	To test that 0 is not accepted.			
2	InputNumber: 1	True when IllegalArgument exception caught.	To test that 1 is not accepted.			
3	InputNumber: -1	True when IllegalArgument exception caught.	To test that negative number is not accepted.			
4	InputNumber: - 5000	True when IllegalArgument exception caught.	To test that different negative is not accepted.			
5	InputNumber: 1,000,000,001	True when IllegalArgument exception caught.	To test that large number is not accepted.			
6	InputNumber: 2,000,000,000	True when IllegalArgument exception caught.	To test that different large number is not accepted.			

Section 2.2.4 Unit Test Plan for PrimeNumberFinder CalculatePrimes

Test id	**Input /Parameter Values**	**Expected Output / Returned Value**	**Reason for the test**	**Actual Output / Returned Value**	**Correct (Yes / No)**	**Seriousness of the Error**
1	InputNumber : 12	{2, 3, 5, 7, 11}	To test primes are being calculated correctly.			
2	InputNumber: 2	{2}	To test smallest value works.			
3	InputNumber: 7920	{2, 3, 5, …. , 7919}	To test that large number works.			
4	InputNumber: 1,000,000,000	999,999,937	To test that largest number works.			

Section 2.2.5 Unit Test Plan for PrimeFactorFinder Constructor

Test id	**Input /Parameter Values**	**Expected Output / Returned Value**	**Reason for the test**	**Actual Output / Returned Value**	**Correct (Yes / No)**	**Seriousness of the Error**
1	InputNumber : 0	True when IllegalArgument exception caught.	To test that 0 is not accepted.			
2	InputNumber: 1	True when IllegalArgument exception caught.	To test that 1 is not accepted.			
3	InputNumber: -1	True when IllegalArgument exception caught.	To test that negative number is not accepted.			
4	InputNumber: - 5000	True when IllegalArgument exception caught.	To test that different negative is not accepted.			
5	InputNumber: 2,000,000,001	True when IllegalArgument exception caught.	To test that large number is not accepted.			
6	InputNumber: 3,000,000,000	True when IllegalArgument exception caught.	To test that different large number is not accepted.			

Section 2.2.6 Unit Test Plan for PrimeFactorFinder CalculatePrimeFactors

Test id	Input /Parameter Values	Expected Output / Returned Value	Reason for the test	Actual Output / Returned Value	Correct (Yes / No)	Seriousness of the Error
1	InputNumber : 17	“17 is a prime number!”	To test primes are being calculated correctly.			
2	InputNumber: 2	“2 is a prime number!”	To test smallest prime works.			
3	InputNumber: 999999937	“999999937 is a prime number!”	To test that largest possible prime works.			
4	InputNumber: 17,483,781	‘3, 7, 19, 29, 1511’	To test that prime factors, work.			
5	InputNumber: 4	‘2, 2’	To test that smallest prime factors, work.			
6	InputNumber: 2,000,000,000	"2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5"	To test that largest prime factors, work.			

Section 2.3 Integration Testing

Section 2.3.1 Definition

As the only unit that is directly using another unit is the PrimeFactorFinder class, it is the only class to be tested here. The tests are to be repeated as above, but instead of mocking the PrimeNumberFinder class, they will now be calculated using the actual class instead. As the logic in the constructor is the same and it doesn’t depend on the PrimeNumberFinder class it can be omitted from these tests.

Section 2.3.2 Overview of Items to be Tested

Item to be tested	Test Description	Test Date
CalculatePrimeFactors	To test that the prime factors are being calculated properly.	4/6/2019

For each “integrated” component to be tested, you need to create a subsection to document the test plan for testing each individual component. The following is a suggestion.

Section 2.3.3 Integration Test Plan for <<Name of the integrated component>>

Test id	Input /Parameter Values	Expected Output / Returned Value	Reason for the test	Actual Output / Returned Value	Correct (Yes / No)	Seriousness of the Error
1	InputNumber : 17	“17 is a prime number!”	To test primes are being calculated correctly.			
2	InputNumber: 2	“2 is a prime number!”	To test smallest prime works.			
3	InputNumber: 999999937	“999999937 is a prime number!”	To test that largest possible prime works.			
4	InputNumber: 17,483,781	‘3, 7, 19, 29, 1511’	To test that prime factors, work.			
5	InputNumber: 4	‘2, 2’	To test that smallest prime factors, work.			
6	InputNumber: 2,000,000,000	"2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5"	To test that largest prime factors, work.			

Section 2.4 Functional (System) Testing

Section 2.4.1 Definition

The objective of the system testing is to verify that the GUI operates as intended and displays the correct results. As such similar numbers to the above integration test is to be used, with the addition of non-integer inputs such as a String and a double.

Section 2.4.2 Overview of Features to be Tested

Feature to be tested	Test Description	Test Date
Program must determine if a number is prime, or the number’s prime factor using the GUI.	Need to test that an input is entered, and the desired response is given for a range of values.	5/06/19

Section 2.4.3 Functional Test Plan for GUI App

Test id	Input Values	Expected Output	Reason for the test	Actual Output	Correct (Yes / No)	Seriousness of the Error
1	InputNumber : 17	Prime field says "Yes"	To test primes are being calculated correctly.			
2	InputNumber: 2	Prime field says "Yes"	To test smallest prime works.			
3	InputNumber: 999999937	Prime field says "Yes"	To test that largest possible prime works.			
4	InputNumber: 87432920	Factors field contains "2, 2, 2, 5, 463, 4721"	To test that prime factors, work.			
5	InputNumber: 4	Factors field contains "2, 2"	To test that smallest prime factors, work.			
6	InputNumber: 2000000000	Factors field contains "2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5"	To test that largest prime factors, work.			
7	InputNumber: "This is a string"	Displays error message saying "*Please enter valid integer!"	To test that strings are handled.			
8	InputNumber: 10.5	Displays error message saying "*Please enter valid integer!"	To test that decimal numbers are handled.			
9	InputNumber: 0	Displays error message saying "*Please enter valid integer!"	To test that 0 is not accepted.			
10	InputNumber: 1	Displays error message saying "*Please enter valid integer!"	To test that 1 is not accepted.			
11	InputNumber: -1	Displays error message saying "*Please enter valid integer!"	To test that negative number is not accepted.			
12	InputNumber: - 5000	Displays error message saying	To test that different			

		"*Please enter valid integer!"	negative is not accepted.			
13	InputNumber: 2,000,000,001	Displays error message saying "*Please enter valid integer!"	To test that large number is not accepted.			
14	InputNumber: 3,000,000,000	Displays error message saying "*Please enter valid integer!"	To test that different large number is not accepted.			

Section 2.5 Performance Testing (Non-Functional System Testing)

Section 2.5.1 Definition

The purpose of this testing is to test the speed of the algorithm, to ensure that the prime number list is being calculated relatively fast.

Section 2.5.2 Overview of "Non-functional" Requirements to be Tested

Req. Id.	**Non-functional Requirements to be tested**	**Test Description**	**Test Date**
1.	Calculator can calculate on large number quickly.	Large number 1,987,654,321 should take less than 20 seconds to calculate factors of.	5/6/19

Section 2.5.3 Overall Performance Testing (Non-Functional System Testing)

Table 1 Overall Performance Test Plan of Calculator can calculate on large number quickly.

Test scenario	**Test environment and setting**	**Expected response time**	**Reason for the test**	**Actual response time**	**Meet expectation (Yes / No)**
1	On GUI, input number: 598765432.	20,000ms	To test speed of algorithm on large number.		

Section 3 Test Automation

As the program was written in Java, there was several testing libraries that I was able to employ to completely automate the testing of the application.

The Unit/Integration testing was completed using JUnit with some help from Mockito in the unit testing to mock the dependencies in the PrimeFactorFinder class.

To set up these tools, it's easy as downloading them from the internet, adding them as libraries into the NetBeans project and then importing the required functionality into the test script.

JUnit allows for classes to be created and then methods called and having the expected result and actual result compared to see if the intended outcome occurred. Mockito allows classes that are being used within other methods to be mocked so that the actual functionality of the dependent class is not used, enabling unit testing to be carried out.

To set up the tests was as simple as, writing the code instantiating the object and then performing the required methods on it and then using getters to get the value of the required variable and compare it to the hardcoded expected value. To customise the tests for each test case all that needed to be changed was the values being passed in and the expected outcome.

AssertJ has a swing component testing tool, for testing the performance of the Swing framework. To set up the tests it required setting each relevant JComponent a name and then used methods to check the value of these components to see if the contained the expected result, similarly to above. Likewise for customising all that needed to change was the input value and the expected outcome.

Section 4 Test Result Summary

The following section outlines a summary of the various test results conducted from the above test cases, organised by the type of testing done.

Section 4.1 Summary of Unit Testing Results

Please see the following graph and table for the summary of unit test results. A more in-depth discussion follows in Section 5.1.

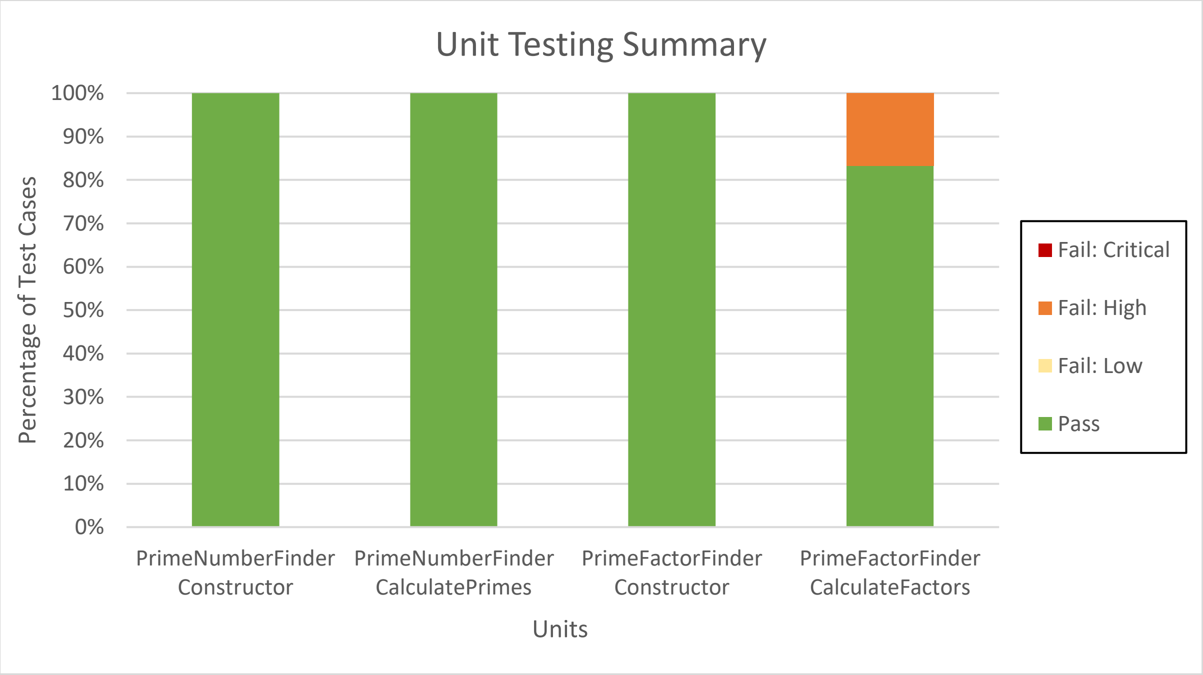


Figure 2. Unit Testing Results

Unit under test	Test Description	Test Date	Results Summary	Comments
PrimeNumberFinder Constructor	Tests the constructor of PrimeNumberFinder	4/6/2019	100% pass rate.	All input values successfully accounted for.
PrimeNumberFinder CalculatePrimes	Test that all primes up to the given number are calculated correctly.	4/6/2019	100% pass rate.	All primes calculated as expected.
PrimeFactorFinder Constructor	Tests the constructor of PrimeFactorFinder	5/6/2019	100% pass rate.	All input values successfully accounted for.
PrimeFactorFinder CalculatePrimeFactors	Test that the program is correctly calculating it's prime factors of it's a prime number.	5/6/2019	83.3% pass rate. 0% critical errors. 16.7% high errors.	Fails when lower bounds input is entered. All other tested inputs behave as expected.

Section 4.2 Summary of Integration Testing Results

Please see the following graph and table for the summary of integration test results. A more in-depth discussion follows in Section 5.2.

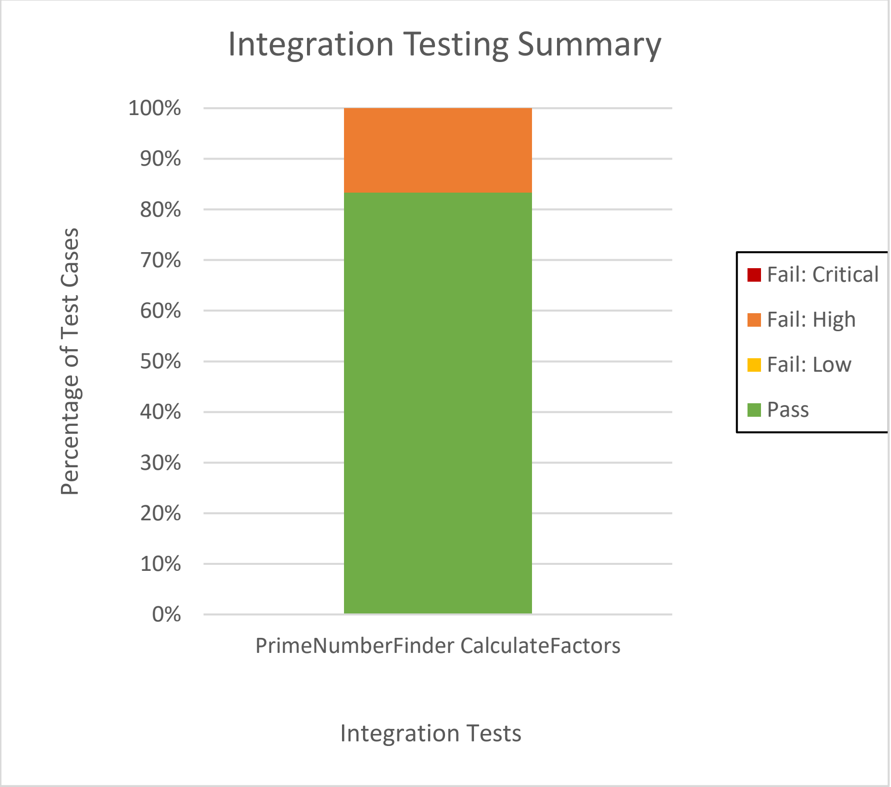


Figure 3. Integration Testing Results.

Integrated component under test	Test Description	Test Date	Results Summary	Comments
PrimeFactorFinder CalculatePrimeFactors	Test that the program is correctly calculating its prime factors of it's a prime number.	5/6/2019	83.3% pass rate. 0% critical errors. 16.7% high errors.	Fails when lower bounds input is entered. All other tested inputs behave as expected.

Section 4.3 Summary of Functional (System) Testing Results

Please see the following graph and table for the summary of unit test results. A more in-depth discussion follows in Section 5.3.

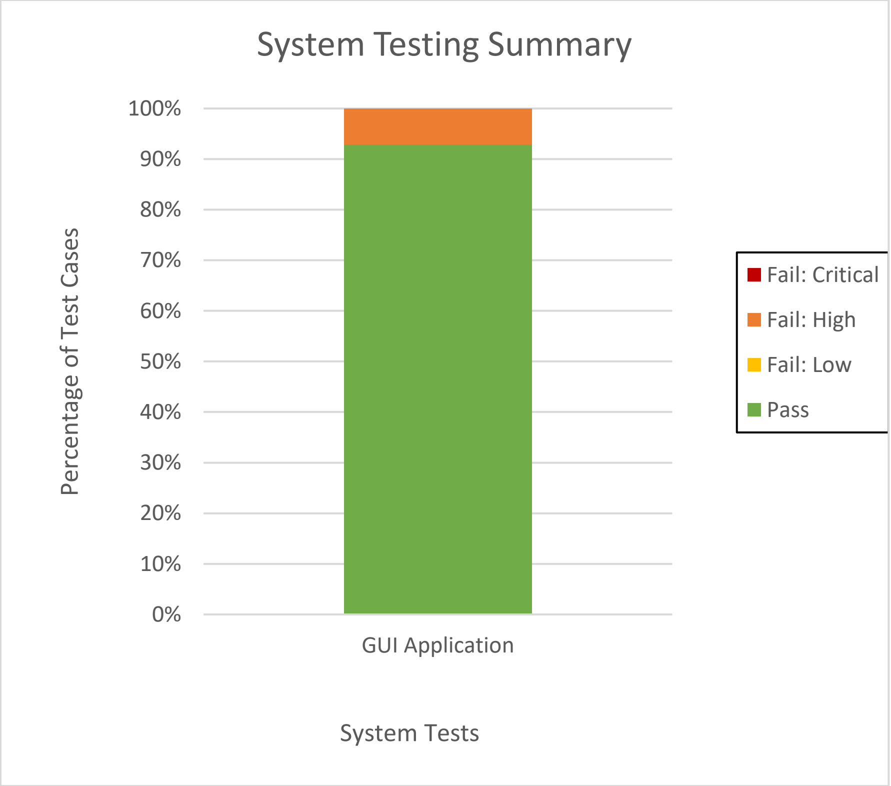


Figure 4. System Testing Results.

Functional requirement under test	Test Description	Test Date	Results Summary	Comments
Program must determine if a number is prime, or the number's prime factor using the GUI.	Need to test that an input is entered, and the desired response is given for a range of values.	5/06/19	92.9% pass rate. 0% critical errors. 7.1% high errors.	GUI behaved as expected. The one failure was same input that was failing before.

Section 4.4 Summary of Performance (Non-Functional System) Testing Results

Please see the following graph and table for the results of the performance testing.

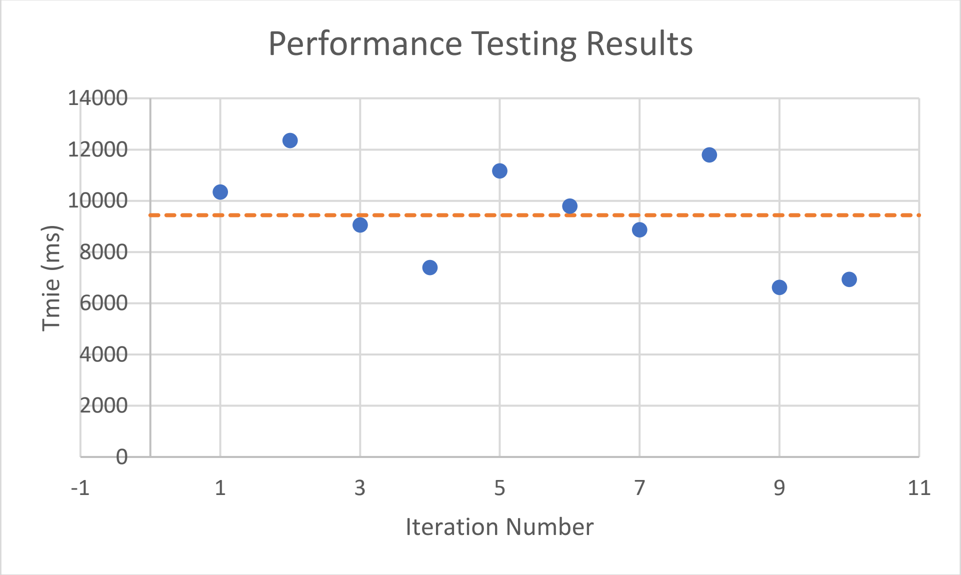


Figure 5. Performance Testing Results (with average line in orange).

Non-functional requirement under test	Test Description	Test Date	Results Summary	Comments
Calculator can calculate on large number quickly.	Large number 598765432 should take less than 20 seconds to calculate factors of.	5/6/19	100% pass rate.	Average completion time of 9473ms.

Section 5 Detailed Test Results

The following section outlines the detailed test results.

Section 5.1 Unit Testing related to PrimeNumberFinder

The following tables tabulate the unit testing results for the PrimeNumberFinder class based on the test cases outlined earlier in the document.

Table 2 Test results of PrimeNumberFinder Constructor based on Section 2.2.3 in this document.

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	4/6/2019	True	Pass	N/A	
2	4/6/2019	True	Pass	N/A	
3	4/6/2019	True	Pass	N/A	
4	4/6/2019	True	Pass	N/A	
5	4/6/2019	True	Pass	N/A	
6	4/6/2019	True	Pass	N/A	

Table 2 Test results of PrimeNumberFinder Constructor based on Section 2.2.4 in this document.

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	4/6/2019	{2, 3, 5, 7, 11}	Pass	N/A	
2	4/6/2019	{2}	Pass	N/A	
3	4/6/2019	{2, 3, 5, …. , 7919}	Pass	N/A	
4	4/6/2019	999999937	Pass	N/A	Only testing for largest possible prime due to inability to find table of all prime to 1,000,000,000.

Section 5.2 Unit Testing related to PrimeFactorFinder

The following tables tabulate the unit testing results for the PrimeFactorFinder class based on the test cases outlined earlier in the document.

Table 3 Test results of PrimeNumberFinder Constructor based on Section 2.2.5 in this document.

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	4/6/2019	True	Pass	N/A	
2	4/6/2019	True	Pass	N/A	
3	4/6/2019	True	Pass	N/A	
4	4/6/2019	True	Pass	N/A	
5	4/6/2019	True	Pass	N/A	
6	4/6/2019	True	Pass	N/A	

Table 2 Test results of PrimeNumberFinder CalculatePrimeFactors based on Section 2.2.6 in this document.

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	5/6/2019	“17 is a prime number!”	Pass	N/A	
2	5/6/2019	Exception thrown “ERROR: Number cannot be one”	Fail	High	Prime number list not generated as 1 is passed in as the value which is not accept.
3	5/6/2019	“999999937 is a prime number!”	Pass	N/A	Largest prime calculated.
4	5/6/2019	‘3, 7, 19, 29, 1511’	Pass	N/A	
5	5/6/2019	‘2, 2’	Pass	N/A	
6	5/6/2019	"2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5"	Pass	N/A	

Section 5.2 Integration Testing related to PrimeFactorFinder

The following tables tabulate the integration testing results for the PrimeFactorFinder class based on the test cases outlined earlier in the document.

Table 4 Test results of PrimeFactoerFinder based on Section 2.3.3 in this document.

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	5/6/2019	"17 is a prime number!"	Pass	N/A	
2	5/6/2019	Exception thrown "ERROR: Number cannot be one"	Fail	High	Still failing, due to same error as above.
3	5/6/2019	"999999937 is a prime number!"	Pass	N/A	Largest prime calculated.
4	5/6/2019	'3, 7, 19, 29, 1511'	Pass	N/A	
5	5/6/2019	'2, 2'	Pass	N/A	
6	5/6/2019	"2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5"	Pass	N/A	

Section 5.3 Functional (System) Testing

The table below tabulates the results of the functional testing of the GUI application.

Table 5 Test results of GUI Application based on 2.4.3 in this document

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	5/6/2019	Prime field says "Yes"	Pass	N/A	
2	5/6/2019	Prime field says "Yes"	Fail	High	Two still fails here, to no surprise, error is propagating through the program.
3	5/6/2019	Prime field says "Yes"	Pass	N/A	
4	5/6/2019	Factors field contains "2, 2, 2, 5, 463, 4721"	Pass	N/A	
5	5/6/2019	Factors field contains "2, 2"	Pass	N/A	
6	5/6/2019	Factors field contains "2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5"	Pass	N/A	

Test Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
7	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
8	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
9	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
10	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
11	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
12	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
13	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	
14	5/6/2019	Displays error message saying "*Please enter valid integer!"	Pass	N/A	

Section 5.4 Performance (Non-Functional System) Testing

This subsection discusses the detailed results of various performance testing activities based on the software test plan document.

Section 5.4.1 Overall Performance Testing of PrimeFactorFinder Calculate Primes

Table 6 Results of Performance Test of PrimeFactorCalculator based on Section 2.5.3 in this document.

Test Scenario Id	Date tested	Actual output / return value	Pass / Fail	Seriousness of error	Summary / Comments
1	5/6/2019	9,473 ms	Pass	N/A	Time is well under limit of 20,000ms