Learning Report – Applied System Development Life Cycle and Software Testing

L&T Technology Services

Global Engineering Academy

Genesis

Document History

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	To be approved By	Remarks/Revision Details
1		Name/PS No	Name/PS No	Module Owner Name	Comments
2	15/02/21	99003779			

Table of Contents

Table of Figures

Table of Tables

ACTIVITY 1
TO DESIGN A CALCULATOR USING C PROGRAMMING

State of Art :

In order to meet the requirement of today's world, man has to be very fast. To do so, it is very genuine to face difficulties to meet some basic essentials. So we did some brain storming to search for them. The Research has been divided on the basis of cost and features of different calculators. The price ranges from Rs 100-300 which can be used by students in school & in universities, scientist and scholars. This device includes a large range of features at lower cost. The features include basic arithmetic operations, BMI calculation, and conversion of Numeric into binary, octal, hexadecimal and vice versa. It can also perform basic trigonometry calculations. It includes nth root and power of n calculating features. Another category includes Body mass index (BMI) feature, calculation of trigonometric functions including exponential, logarithms and number conversion. The features can be enhanced further but at the cost will increase accordingly. At a range of Rs 600-650, complex calculations and imaginary numbers can be included. Calculators having medium costs are having medium set of features which includes matrix and calculus. Higher features at medium cost of price includes binary conversions and are foldable. Calculators of higher prices includes the functions of database management, higher accuracy, wider display for plots and graphs. It also includes smart touch, solar cell operations, battery charging and a waterproof.

Cost and Features:

The whole document has been divided on the basis of cost and features of different calculators. Following is a list of features based on different cost and prices of calculators.

1. Low cost and low featured Calculator: These types of calculator ranges from Rs. 50-200. It will be able to calculate basic arithmetic operations and are mobile.
2. Low cost and medium featured: Such calculator ranges from Rs. 100-300 which will be able to perform arithmetic calculations including fractions, nth root and power of n.
3. Low cost and high featured: It includes operation of trigonometry, logarithm, exponential, inversion and degrees. These calculators ranges from Rs. 250-500.
4. Medium Cost and Low featured: It ranges from Rs. 500-650 with features of solving complex calculations, imaginary number and is having a larger display.
5. Medium cost and medium featured: It ranges from Rs. 750-1500 including matrix operation, calculus and statistics.
6. Medium cost and high featured: These types of calculator ranges from Rs 1200-2500. These are able to perform number calculations and are foldable.
7. High cost and low featured: Such calculator ranges from Rs. 5000-10000. It includes features for respective business fields with high accuracy.
8. High cost and medium featured : It includes printing features with touch, solar charging, data security and internet at a price range of Rs 12000-25000

9. High cost and high featured: Its price ranges from 30000 to 1 lakhs and are named as programmable calculators.

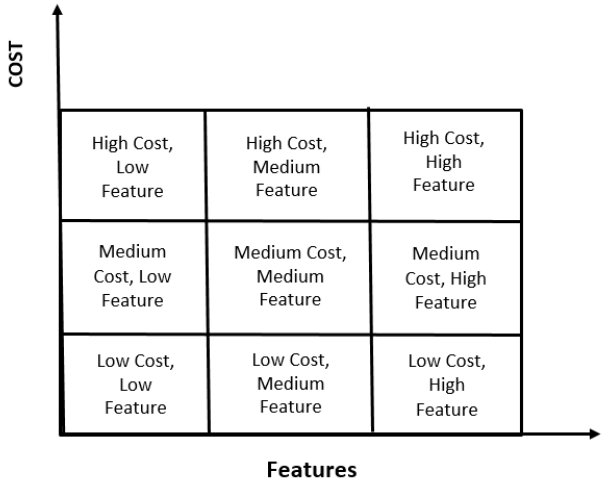

Figure 1: Cost vs features

Defining our system:

The designed product has all the necessary features required by the undergraduates and postgraduates students including scholars. The High level requirements include arithmetic operations, decimals, trigonometric functions, nth root, power of n, fractions, percentage, logarithms, exponentials, binary conversions etc. The low level requirements of the product includes addition, subtraction, multiplication, division under arithmetic operation, decimal upto 8 digits, trigonometric functions with square root and radians.

SWOT Analysis:

Table 1: SWOT Analysis

Strength: • High Accuracy • Solar powered, more efficient • Touch screen, easy to accessible • foldable	**Weakness:** • Charging issues • Not accessible on cloudy days • Not resistive to water
Opportunities: • Easily accessible to by students • Complex calculations are easy to solve. • Affordable to all	**Threats:** • Security issues • Errors

4W's and 1'H :

Who: Basically used by students of UG and PG.
What: It is a highend, affordable calculator.
When: Easily accessible and performs complex calculations.
Where: To solve simple as well as complex calculations.
How: End user friendly and easily accessible.

UML Diagram:

High Level Design:

Structural Diagram

Class Diagram:

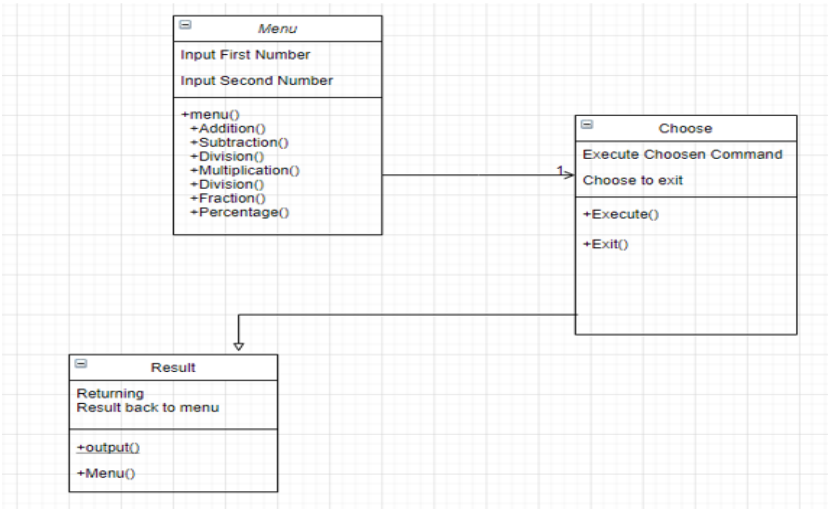

Figure 2: Class Diagram of arithmetic calculator

Behavioural Diagram:

Activity Diagram:

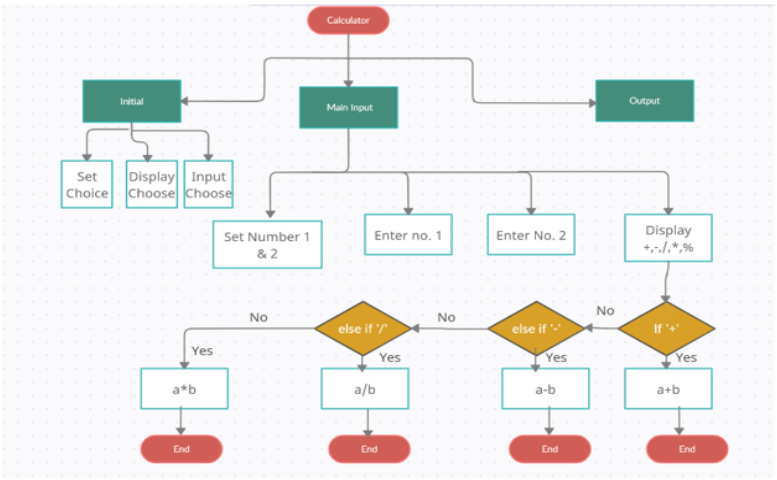

Figure 3: Activity Diagram of arithmetic calculator

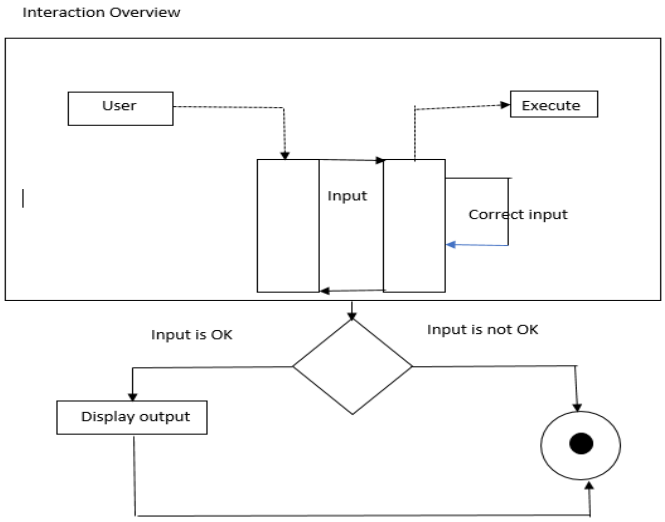

Figure 4: Interaction overview of Arithmetic calculator

Low Level Design:

Structural Diagram:

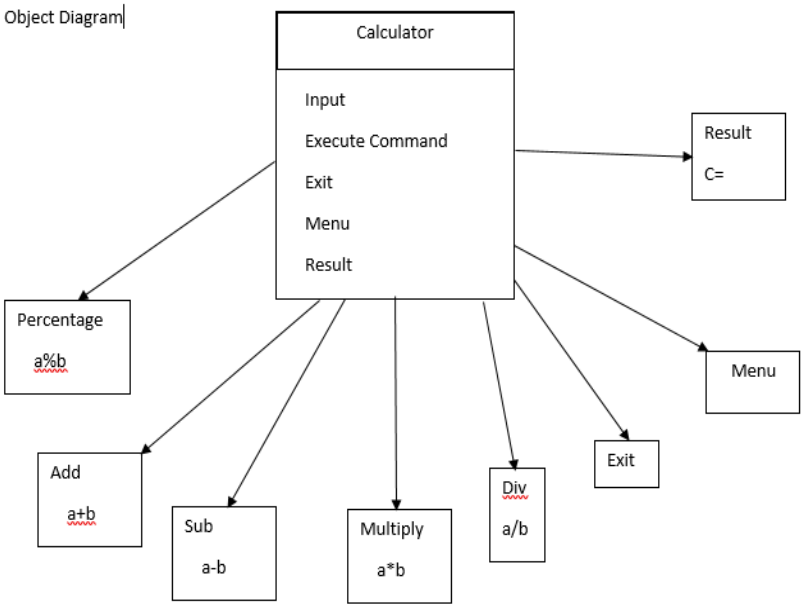

Figure 5: Object diagram of arithmetic table

Behavioural Diagram:

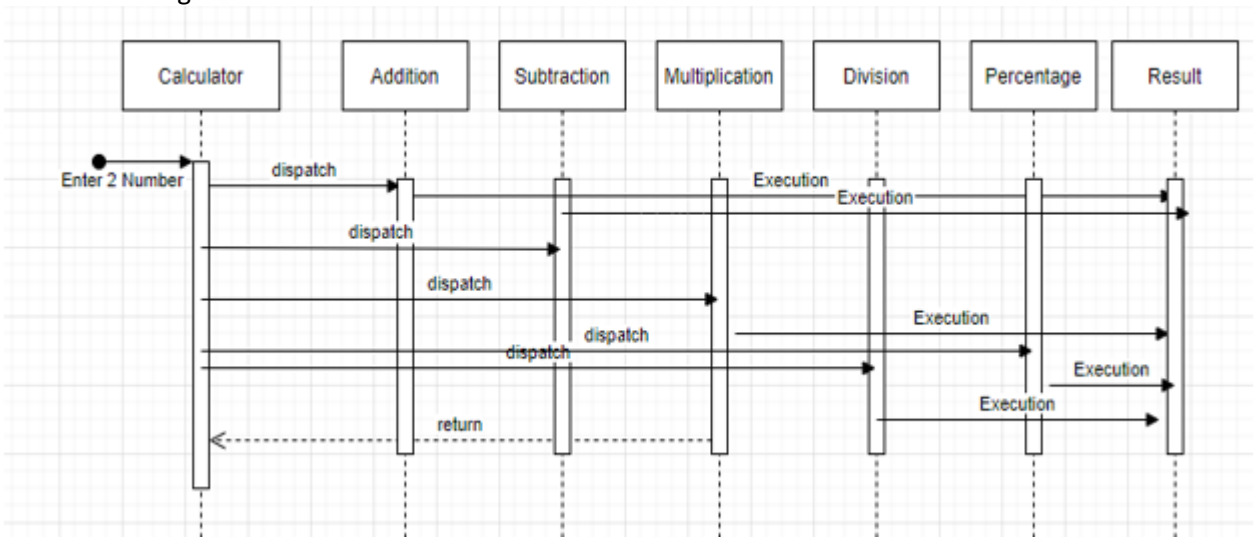

Figure 6: Sequence Diagram of Arithmetic Calculator

State diagram:

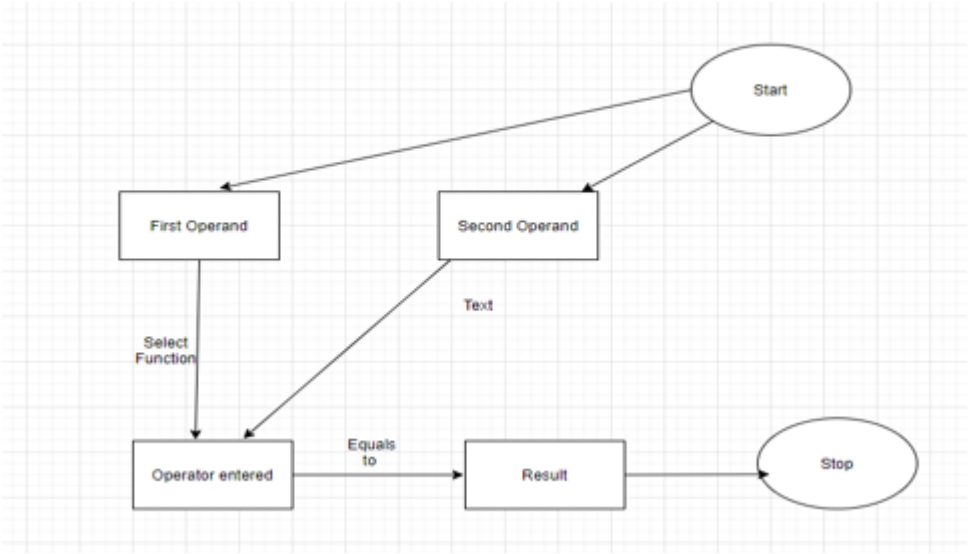

Figure 7: State Diagram of Arithmetic Calculator

Test Plan:-

HLR:

Table 2: Test Plan of HLR

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
H_01	➢ Perform operations of 2 positive numbers	➢ 2 + 4.05 ➢ 4/6+2/3 ➢ 4-3 ➢ 3*4 ➢ 8/4	➢ 6.05 ➢ 8/6=1.33 3333 ➢ 1.0 ➢ 12.0 ➢ 2.0	➢ 6.0 ➢ 1.33 ➢ 1 ➢ 12.0 ➢ 2.0	Requirement based
H_02	➢ Perform operation of positive number and zero	➢ 2+0 ➢ 2-0 ➢ 2*0	➢ 2 ➢ 2 ➢ 0	➢ 2.0 ➢ 2.0 ➢ 0	Scenario based
H_03	➢ Perform operation of large numbers	➢ 345+567 ➢ 893.03-876.23 ➢ 45*34 ➢ 345/28	➢ 912 ➢ 16.8 ➢ 1530 ➢ 12.32	➢ 912.0 ➢ 16.80 ➢ 1530.0 ➢ 12.32	Boundary based

LLR:

Table 3: Addition test plan

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Addition of positive integers and zero ➢	4+0	4	4.0	Requirement based
L_02	➢ Addition of negative integers ➢ Addition of positive integers and negative integers ➢ Addition of negative integer and zero	-4+-2 4+-3 -8+0	-6 1 -8	Error 1.0 error	Scenario based
L_03	➢ Addition of very large integers,	34+789	823	823.0	Boundary Based

Table 4: Subtraction Test Plan

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Subtraction of higher number from a lower number ➢ Subtraction of zero from a number	140-39 34-0	101 34	101.0 34.0	Requirement based
L_02	➢ Subtraction of lower number from higher number	45-35	10	10.0	Scenario based
L_03	➢ Subtraction of two non-numerals ➢ Subtraction of number from zero	@-! 0-37	Error -37	Error Error	Boundary based

Table 5: Multiplication Test Plan

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Multiply two number of different bits	140*39	5460	5460.0	Requirement based
L_02	➢ Multiplication by zero	45*0	0	0.0	Scenario based
L_03	➢ Multiplication of Large Numbers	140*657	91,980	91,980.0	Boundary based

Table 6: Division Test Plan

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Division of higher number by lower number and vice versa.	140/20 20/140	7 0.14	7.0 0.1428	Requirement based
L_02	➢ Division by zero	45/0	Error	Put a valid input	Scenario based
L_03	➢ Division of large numbers	11654/456	25.55	25.55701	Boundary based

Table 7: Remainder Test Plan

Remainder:-

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Remainder of two numbers	(45,5) (17,5)	0 2	0 2	Requirement based
L_02	➢ Remainder of one large and small number	(1,2)	1	1	Scenario based
L_03	➢ Remainder by zero	(1,0)	Error	NA	Boundary based

Table 8: Power Test Plan

Power:-

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Power Calculation of a number	(3^3)	27	27	Requirement based
L_02	➢ Power calculation of number one	(1^2)	1	1	Scenario based
L_03	➢ Power raised to zero	(2^0)	1	1	Boundary based

Table 9: Percentage Test Plan

Percentage:-

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Calculation of percentage of one number by another number	5/10 40/50	50 80	50.0 80.0	Requirement based
L_02	➢ Percentage calculation wrt zero	10/0	Error	Put a valid input	Scenario based
L_03	➢ Percentage of large numbers	456/984	46.34	46.34	Boundary based

Table 10: GCD Test Plan

GCD:-

Test ID	Description	Expected Input	Exp. Out.	Actual output	Type of Test
L_01	➢ Greatest Common divisor of two number	(1,2) (2,5)	1 1	1 1	Requirement based
L_02	➢ GCD of large Numbers	(35,64)	1	1	Scenario based
L_03	➢ GCD of Even numbers	(24,36)	12	12	Boundary based

Git :

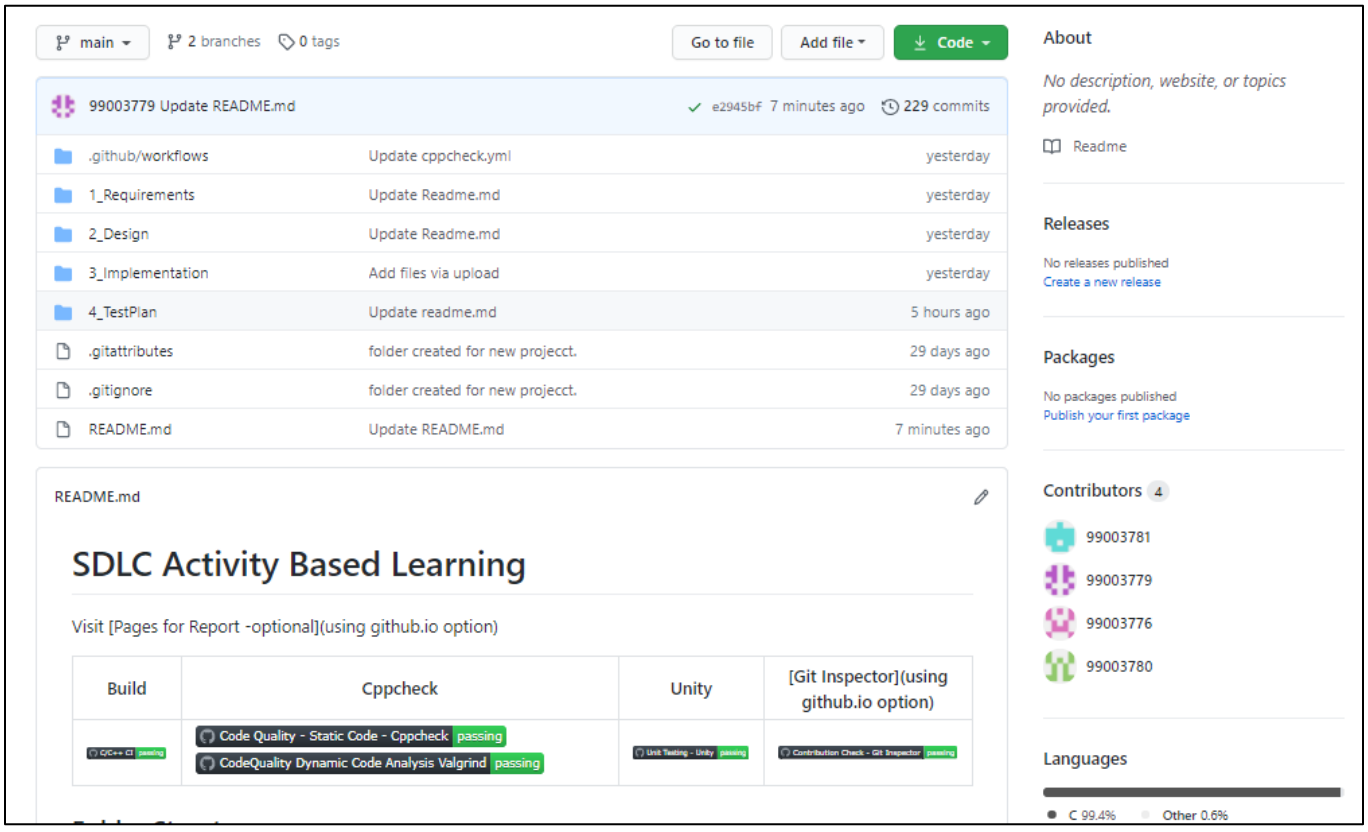

Figure 8: https://github.com/99003781/N8-Calculator.git

Git Issues:

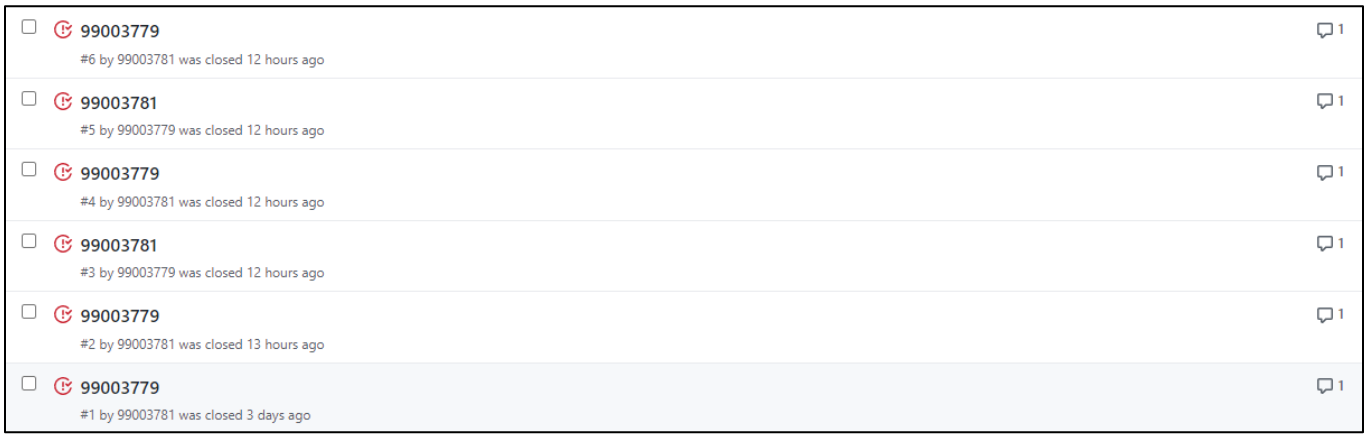

Figure 9: Screenshots of Git issues raised and closed

Main Project:

```
#include "unity.h"
#include "main.h"
#include "calculator_operations.h"

/* Prototypes for all the test functions */

void test_add(void);
void test_add_testcase2(void);
void test_add_testcase3(void);
void test_add_testcase4(void);
void test_add_testcase5(void);

void test_subtract(void);
void test_subtract_testcase2(void);
void test_subtract_testcase3(void);
void test_subtract_testcase4(void);
void test_subtract_testcase5(void);

void test_multiply(void);
void test_multiply_testcase2(void);
void test_multiply_testcase3(void);
```

```
void test_multiply_testcase4(void);
void test_multiply_testcase5(void);

void test_divide(void);
void test_divide_testcase2(void);
void test_divide_testcase3(void);
void test_divide_testcase4(void);
void test_divide_testcase5(void);

void test_percentage(void);
void test_percentage_testcase2(void);
void test_percentage_testcase3(void);
void test_percentage_testcase4(void);
void test_percentage_testcase5(void);

void test_gcd(void);
void test_gcd_testcase2(void);
void test_gcd_testcase3(void);
void test_gcd_testcase4(void);
void test_gcd_testcase5(void);

void test_rem(void);
void test_rem_testcase2(void);
void test_rem_testcase3(void);
void test_rem_testcase4(void);
void test_rem_testcase5(void);

void test_power(void);
void test_power_testcase2(void);
void test_power_testcase3(void);
void test_power_testcase4(void);
void test_power_testcase5(void);

void setUp(){}

void tearDown(){}

int main(void)
{
 UNITY_BEGIN();

   RUN_TEST(test_add);
```

```
  RUN_TEST(test_add_testcase2);
  RUN_TEST(test_add_testcase3);
  RUN_TEST(test_add_testcase4);
  RUN_TEST(test_add_testcase5);

/*SUBTRACTION*/
RUN_TEST(test_subtract);
RUN_TEST(test_subtract_testcase2);
RUN_TEST(test_subtract_testcase3);
RUN_TEST(test_subtract_testcase4);
RUN_TEST(test_subtract_testcase5);

/*MULTIPLICATION*/
RUN_TEST(test_multiply);
RUN_TEST(test_multiply_testcase2);
RUN_TEST(test_multiply_testcase3);
RUN_TEST(test_multiply_testcase4);
RUN_TEST(test_multiply_testcase5);

/*DIVISION*/
RUN_TEST(test_divide);
RUN_TEST(test_divide_testcase2);
RUN_TEST(test_divide_testcase3);
RUN_TEST(test_divide_testcase4);
RUN_TEST(test_divide_testcase5);

/*PERCENTAGE*/
RUN_TEST(test_percentage);
RUN_TEST(test_percentage_testcase2);
RUN_TEST(test_percentage_testcase3);
RUN_TEST(test_percentage_testcase4);
RUN_TEST(test_percentage_testcase5);

/*GCD*/
RUN_TEST(test_gcd);
RUN_TEST(test_gcd_testcase2);
RUN_TEST(test_gcd_testcase3);
RUN_TEST(test_gcd_testcase4);
RUN_TEST(test_gcd_testcase5);

/*REMAINDER*/
RUN_TEST(test_rem);
```

```
 RUN_TEST(test_rem_testcase2);
 RUN_TEST(test_rem_testcase3);
 RUN_TEST(test_rem_testcase4);
 RUN_TEST(test_rem_testcase5);

 /*POWER*/
 RUN_TEST(test_power);
 RUN_TEST(test_power_testcase2);
 RUN_TEST(test_power_testcase3);
 RUN_TEST(test_power_testcase4);
 RUN_TEST(test_power_testcase5);

return UNITY_END();
}

/*ADDITION*/
void test_add(void)
{
 TEST_ASSERT_EQUAL(30, add(10, 20));
}

void test_add_testcase2(void)
{
 TEST_ASSERT_EQUAL(-10, add(10, -20));
}

void test_add_testcase3(void)
{
 TEST_ASSERT_EQUAL(-20.5, add(-40.5, 20.0));
}

void test_add_testcase4(void)
{
 TEST_ASSERT_EQUAL(41.0, add(15.5, 25.5));
}
void test_add_testcase5(void)
{
 TEST_ASSERT_EQUAL(15000, add(7500, 7500));
}

/*SUBTRACTION*/
void test_subtract(void)
```

```
{
 TEST_ASSERT_EQUAL(-3, subtract(0, 3));
}

void test_subtract_testcase2(void)
{
 TEST_ASSERT_EQUAL(2, subtract(5, 3));
}

void test_subtract_testcase3(void)
{
 TEST_ASSERT_EQUAL(7, subtract(10, 3));
}

void test_subtract_testcase4(void)
{
 TEST_ASSERT_EQUAL(-4, subtract(3, 7));
}

void test_subtract_testcase5(void)
{
 TEST_ASSERT_EQUAL(100, subtract(1000, 900));
}

/*MULTIPLICATION*/
void test_multiply(void)
{
 TEST_ASSERT_EQUAL(0, multiply(1, 0));
}

void test_multiply_testcase2(void)
{
 TEST_ASSERT_EQUAL(15, multiply(5, 3));
}

void test_multiply_testcase3(void)
{
 TEST_ASSERT_EQUAL(0, multiply(10, 0));
}

void test_multiply_testcase4(void)
{
```

```
 TEST_ASSERT_EQUAL(-30, multiply(6, -5));
}
void test_multiply_testcase5(void)
{
 TEST_ASSERT_EQUAL(10, multiply(2, 5));
}

/*DIVISION*/
void test_divide(void)
{
 TEST_ASSERT_EQUAL(0, divide(1, 0));
}
void test_divide_testcase2(void)
{
 TEST_ASSERT_EQUAL(5, divide(10, 2));
}

void test_divide_testcase3(void)
{
 TEST_ASSERT_EQUAL(5, divide(5, 1));
}
void test_divide_testcase4(void)
{
 TEST_ASSERT_EQUAL(-6, divide(-30, 5));
}
void test_divide_testcase5(void)
{
 TEST_ASSERT_EQUAL(1, divide(2, 2));
}

/*PERCENTAGE*/
void test_percentage(void)
{
 TEST_ASSERT_EQUAL(100, percentage(1, 1));
}

void test_percentage_testcase2(void)
{
 TEST_ASSERT_EQUAL (40, percentage(2,5));
}

void test_percentage_testcase3(void)
```

```
{
 TEST_ASSERT_EQUAL(20,  percentage(20, 100));
}
void test_percentage_testcase4(void)
{
 TEST_ASSERT_EQUAL(40, percentage(2, 5));
}

void test_percentage_testcase5(void)
{
 TEST_ASSERT_EQUAL(20, percentage(2, 10));
}

/*GCD*/
void test_gcd(void)
{
 TEST_ASSERT_EQUAL(1, gcd(1, 2));
}

void test_gcd_testcase2(void)
{
 TEST_ASSERT_EQUAL (20, gcd(20, 40));
}

void test_gcd_testcase3(void)
{
 TEST_ASSERT_EQUAL (1, gcd(35, 64));
}

void test_gcd_testcase4(void)
{
 TEST_ASSERT_EQUAL (12, gcd(24, 36));
}

void test_gcd_testcase5(void)
{
 TEST_ASSERT_EQUAL(1, gcd(2, 5));
}

/*REMAINDER*/
void test_rem(void)
{
```

```
 TEST_ASSERT_EQUAL(1, rem(1, 2));
}

void test_rem_testcase2(void)
{
 TEST_ASSERT_EQUAL (0, rem(45,5));
}

void test_rem_testcase3(void)
{
 TEST_ASSERT_EQUAL (2, rem(17,5));
}

void test_rem_testcase4(void)
{
 TEST_ASSERT_EQUAL (2, rem(18,4));
}

void test_rem_testcase5(void)
{
 TEST_ASSERT_EQUAL(1, rem(5, 4));
}

/*POWER*/
void test_power(void)
{
 TEST_ASSERT_EQUAL (1, power(1, 2));
}

void test_power_testcase2(void)
{
 TEST_ASSERT_EQUAL (27, power(3,3));
}

void test_power_testcase3(void)
{
 TEST_ASSERT_EQUAL (1, power(3,0));
}

void test_power_testcase4(void)
{
 TEST_ASSERT_EQUAL (1, power(1,4));
```

```
}
void test_power_testcase5(void)
{
 TEST_ASSERT_EQUAL(25, power(5, 2));
}
}
```

Git commits:

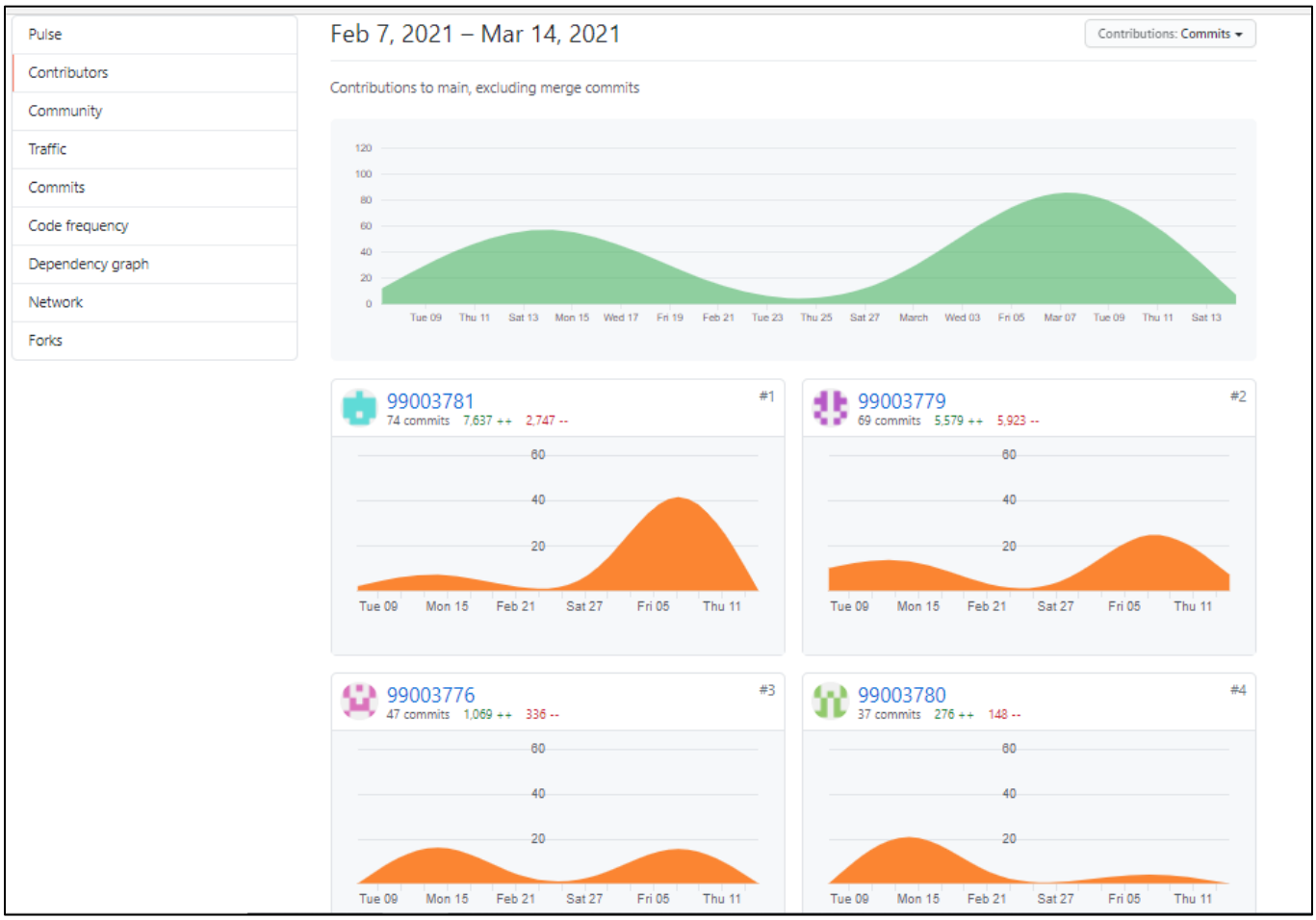

Figure 10: Graph of commits on Github

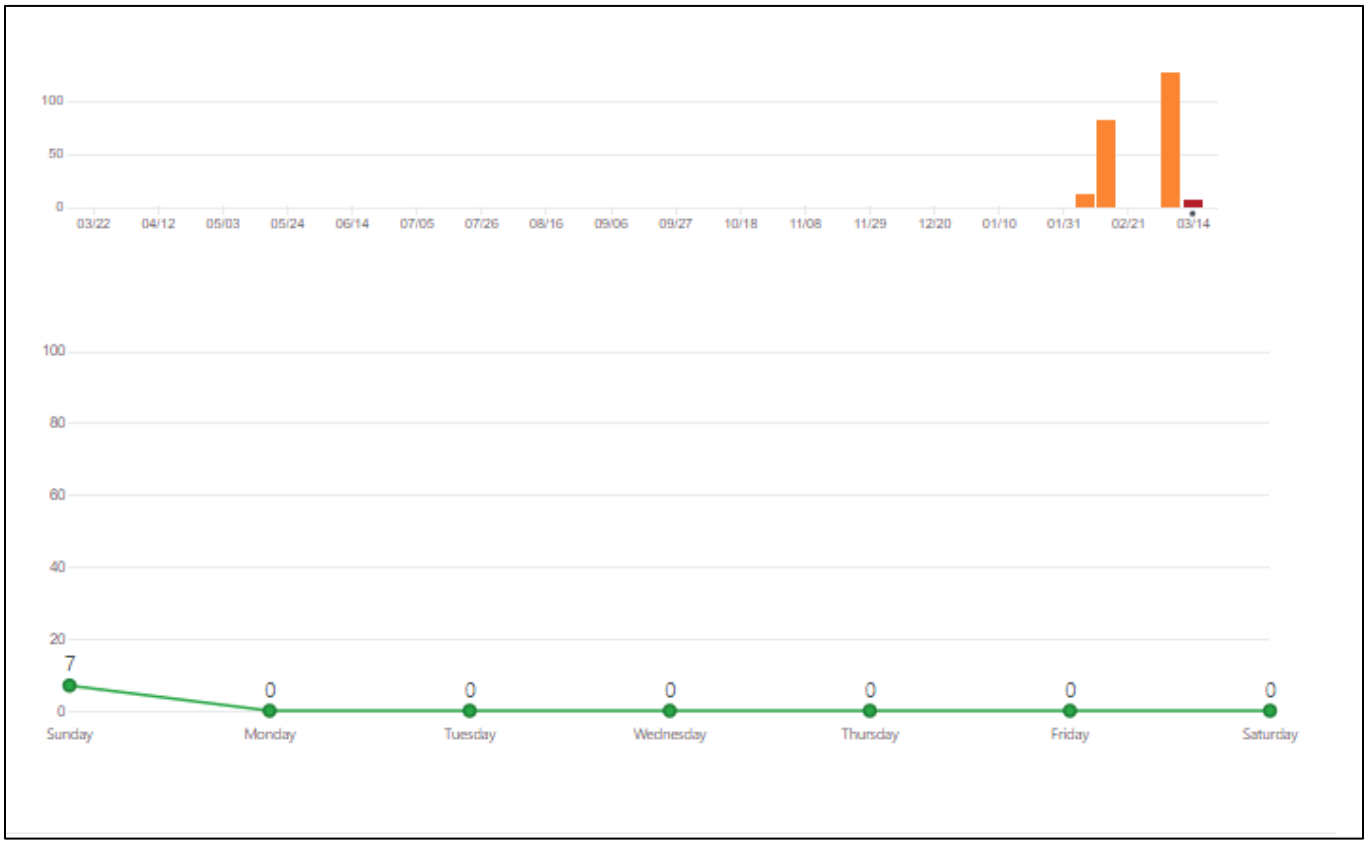

Figure 11: Plot of Git Commits on github