Software Measurement – 6611

Assignment – 1

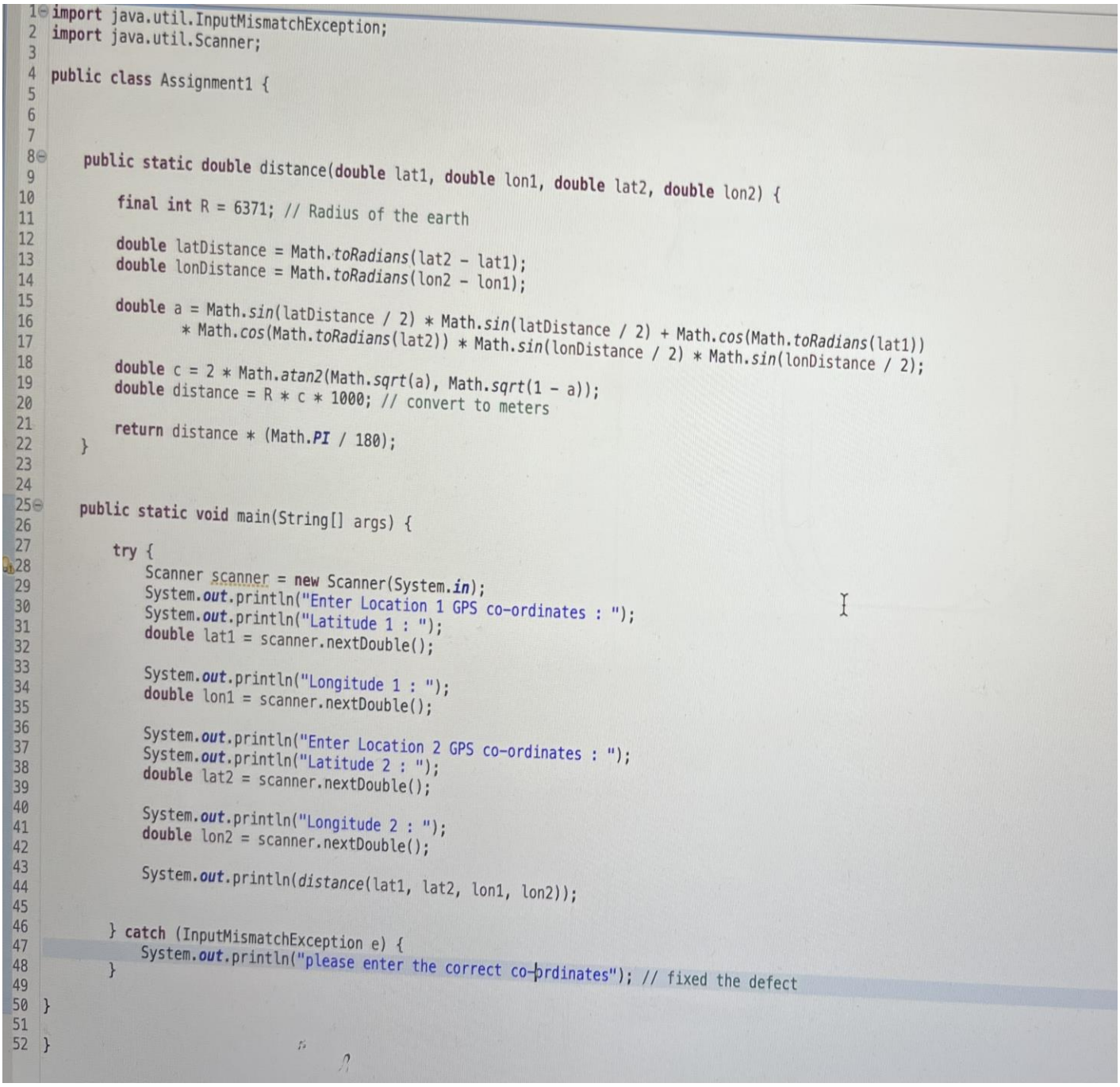
Saswati Chowdhury – 40184906

G1 Deliverables :

1.2. Source Code - distance between two GPS co-ordinates using latitude and longitude in radians :

Estimated Effort: 30 person-minutes to 45 person-minutes

[Also attached zip file]



```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class Assignment1 {



    public static double distance(double lat1, double lon1, double lat2, double lon2) {

        final int R = 6371; // Radius of the earth

        double latDistance = Math.toRadians(lat2 - lat1);
        double lonDistance = Math.toRadians(lon2 - lon1);

        double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2) + Math.cos(Math.toRadians(lat1))
                * Math.cos(Math.toRadians(lat2)) * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);

        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        double distance = R * c * 1000; // convert to meters

        return distance * (Math.PI / 180);
    }


    public static void main(String[] args) {

        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter Location 1 GPS co-ordinates : ");
            System.out.println("Latitude 1 : ");
            double lat1 = scanner.nextDouble();

            System.out.println("Longitude 1 : ");
            double lon1 = scanner.nextDouble();

            System.out.println("Enter Location 2 GPS co-ordinates : ");
            System.out.println("Latitude 2 : ");
            double lat2 = scanner.nextDouble();

            System.out.println("Longitude 2 : ");
            double lon2 = scanner.nextDouble();

            System.out.println(distance(lat1, lat2, lon1, lon2));


        } catch (InputMismatchException e) {
            System.out.println("please enter the correct co-ordinates"); // fixed the defect
        }

}

}
```

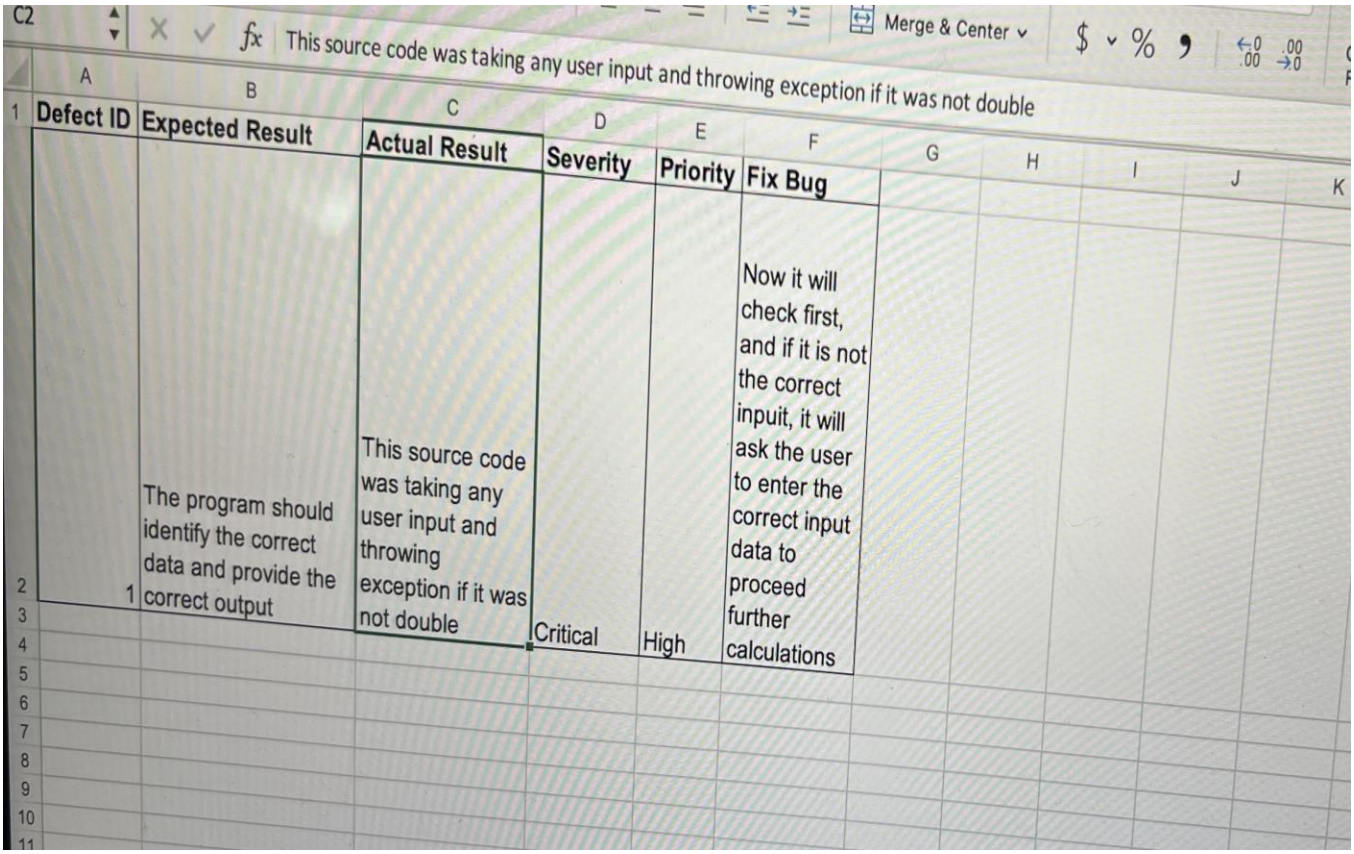
1.3. Exact Time to write the code : 40 person-minutes [including writing the code, testing, finding the defect and fixing the defect]

1.4. Number of Defects : 1

Defect description: Initially there was no restriction taking input data from the users, and that's why any input, which is not double, if entered, was going to throw an exception.

Fix defect: If input data is not double, it will ask the user to enter the correct data to proceed further calculations.

[Explanation in Sheet2 - attached]



1.5.a) Physical SLOC :

Rule:

- Count the statements in the source code
- Do not count the blank lines
- Do not count the comment lines

SLOC - Manually :

SLOC : 32

Comment lines : 0

Blank Lines : 20

1.5.b)

SLOC By Tool :

Software : Source Monitor

SLOC : 32

Comment lines : 0

Blank Lines : 20

File Detail by Folder - Sorted by SLOC

Folder: C:\Users\User\Desktop\SOEN-6611 back

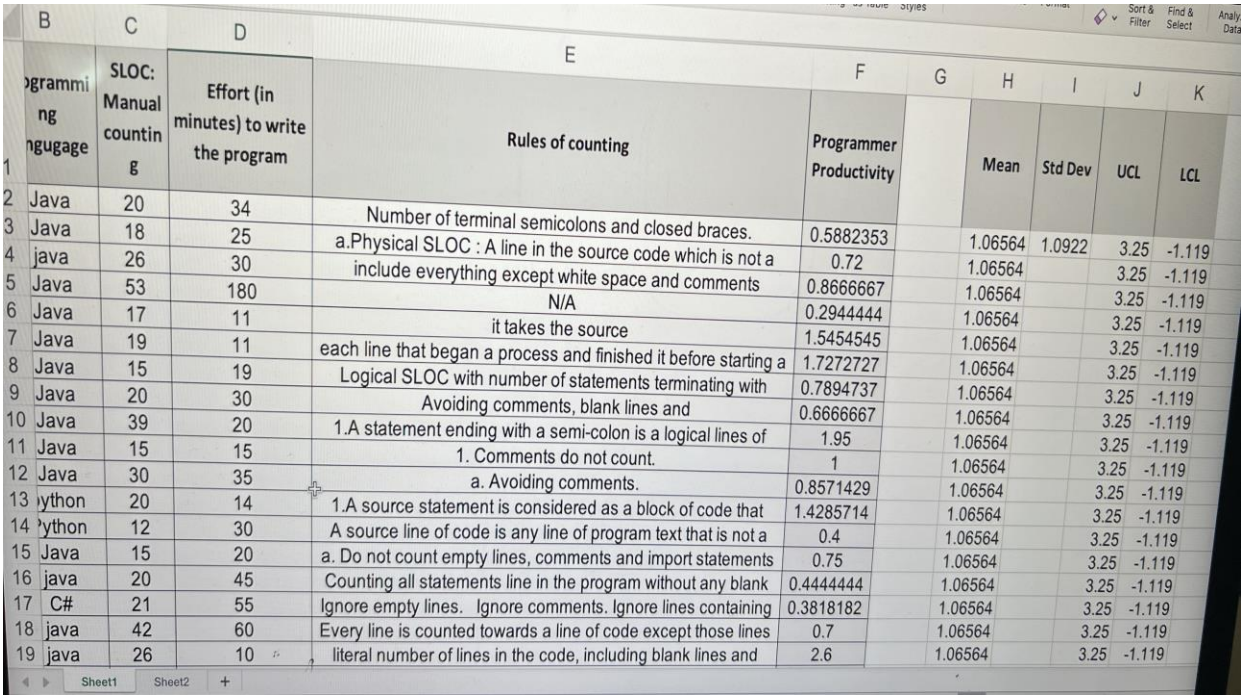
File	SLOC %	SLOC	Comments	BlankLines	Total
Assignment1.java	100.00 %	32	0	20	52
Subtotal - C:\Users\User\Desktop\SOEN-6611	100.00 %	32	0	20	52

Top

1.6. Interpret the Result : Source Monitor tool is used here to find SLOC between 5.a) and 5.b) and they are identical, which means they match 100%. Therefore, we can say that this tool follows all the rules of physical SLOC and gives the correct output and that's why Source Monitor can be considered to measure the physical SLOC for any language.

G2 Deliverables:

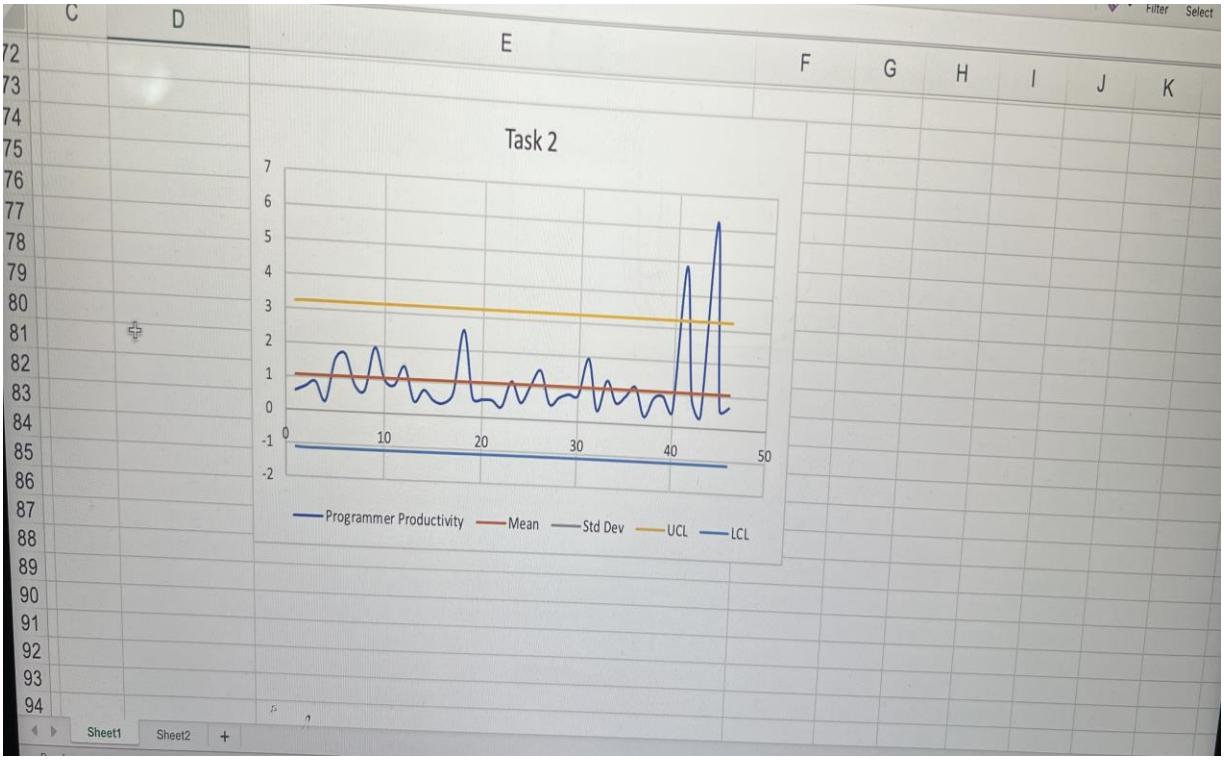
2.1. Attached excel sheet1



	B	C	D	E	F	G	H	I	J	K
1	›grammi ng ngugage	SLOC: Manual countin g	Effort (in minutes) to write the program	Rules of counting	Programmer Productivity		Mean	Std Dev	UCL	LCL
2	Java	20	34	Number of terminal semicolons and closed braces.	0.5882353		1.06564	1.0922	3.25	-1.119
3	Java	18	25	a.Physical SLOC : A line in the source code which is not a	0.72		1.06564		3.25	-1.119
4	java	26	30	include everything except white space and comments	0.8666667		1.06564		3.25	-1.119
5	Java	53	180	N/A	0.2944444		1.06564		3.25	-1.119
6	Java	17	11	it takes the source	1.5454545		1.06564		3.25	-1.119
7	Java	19	11	each line that began a process and finished it before starting a	1.7272727		1.06564		3.25	-1.119
8	Java	15	19	Logical SLOC with number of statements terminating with	0.7894737		1.06564		3.25	-1.119
9	Java	20	30	Avoiding comments, blank lines and	0.6666667		1.06564		3.25	-1.119
10	Java	39	20	1.A statement ending with a semi-colon is a logical lines of	1.95		1.06564		3.25	-1.119
11	Java	15	15	1. Comments do not count.	1		1.06564		3.25	-1.119
12	Java	30	35	a. Avoiding comments.	0.8571429		1.06564		3.25	-1.119
13	›ython	20	14	1.A source statement is considered as a block of code that	1.4285714		1.06564		3.25	-1.119
14	›ython	12	30	A source line of code is any line of program text that is not a	0.4		1.06564		3.25	-1.119
15	Java	15	20	a. Do not count empty lines, comments and import statements	0.75		1.06564		3.25	-1.119
16	java	20	45	Counting all statements line in the program without any blank	0.4444444		1.06564		3.25	-1.119
17	C#	21	55	Ignore empty lines. Ignore comments. Ignore lines containing	0.3818182		1.06564		3.25	-1.119
18	java	42	60	Every line is counted towards a line of code except those lines	0.7		1.06564		3.25	-1.119
19	java	26	10	literal number of lines in the code, including blank lines and	2.6		1.06564		3.25	-1.119

Sheet1 Sheet2 +

2.2. Attached excel sheet1



2.3. Interpret the Result : All the programmers except two (P41 and P44) are within the control limits(Between UCL and LCL).

Two programmers' (P41 and P44) wrote so much code in less time. Their productivity are 4.8666667 and 6.2222222 respectively which are more than UCL(Upper Control Limits which is 3.25 here), and that's why these two programmers' productivity graph exceeds UCL graph.

So we can conclude that there are two developers with an unusually high productivity. Hence, we can not reliably use the average productivity for further analysis as it falls outside the control limits which is a bigger red flag in terms of something highly impactful going wrong.

Reference : Professor Notes and videos