UNIVERSITÉ Concordia UNIVERSITY

SOEN 6611 Software Measurement

ID: 40193024
Mrinal Rai

Assignment -1

Submitted To: **Dr. Olga Ormandjieva**

Goal 1: Practice measurement data collection and analysis

Source code:

The following program is written in Java (The Source code is attached in the submission ***Assignment1.java***)

```java
package com.mrynalrai;

import java.io.IOException;

public class Assignment {
    public static void main(String args[]) {
        double lat1 = 53.32055556;
        double lon1 = -1.729722222;
        double lat2 = 53.31861111;
        double lon2 = -1.699722222;
        System.out.println(distance(lat1, lat2,
                lon1, lon2) + " K.M");
    }

    public static double distance(double lat1,
                                  double lat2, double lon1,
                                  double lon2)
    {
        try {
            if (!isValid(lat1, "latitude") || !isValid(lon1, "longitude") || !isValid(lat2, "latitude") || !isValid(lon2,
"longitude")) {
                throw  new Exception("Invalid Coordinate");
            }
        } catch (Exception msg) {
            System.out.println(msg);
            System.exit(1);
        }

        // The math module contains a function
        // named toRadians which converts from
        // degrees to radians.
        lon1 = Math.toRadians(lon1);
        lon2 = Math.toRadians(lon2);
        lat1 = Math.toRadians(lat1);
        lat2 = Math.toRadians(lat2);

        // Haversine formula
        double dlon = lon2 - lon1;
        double dlat = lat2 - lat1;
        double a = Math.pow(Math.sin(dlat / 2), 2)
                + Math.cos(lat1) * Math.cos(lat2)
                * Math.pow(Math.sin(dlon / 2),2);

        double c = 2 * Math.asin(Math.sqrt(a));

        // Radius of earth in kilometers. Use 3956
        // for miles
        double r = 6371;
    // calculate the result
    return(c * r);
  }

  public static boolean isValid (double coordinate, String type) {
    if (type == "latitude") {
      if (coordinate >= -90 && coordinate <= 90) {
        return true;
      } else {
        return false;
      }
    } else if (type == "longitude") {
      if (coordinate >= -180 && coordinate <= 180) {
        return true;
      } else {
        return false;
      }
    }
    return false;
  }
}
```

Task 2: Estimate the Effort

Estimated Effort = Number of Developer X Time (range of min-max)
Estimated Effort = 1 X (45-60 minutes)

Task 3: Record the Work Effort

Number of Developer =1.
time taken to write the code = 40 minutes.
time taken to test the code = 10 minutes.
time taken to fix the defect = 10 minutes.
Actual effort = 1 x (40 + 10 + 10) = 60 mins

Task 4: Record the Number of Defects

Defect #	Input	Expected	Actual	Priority	Status	Fix
1	lat1 = -92, lon1 = -200 lat2 = 50, lon2 = 50	java.lang.Exception: Invalid Coordinate	15487.22 K.M.	Medium	Fixed	Added isValid function to check if the coordinates are in valid range

Task 5: Calculate the length of your program in physical SLOC

a) Manually, documenting the rules for the counting of the lines of code that you used

Rules for the counting the physical source lines of code SLOC:

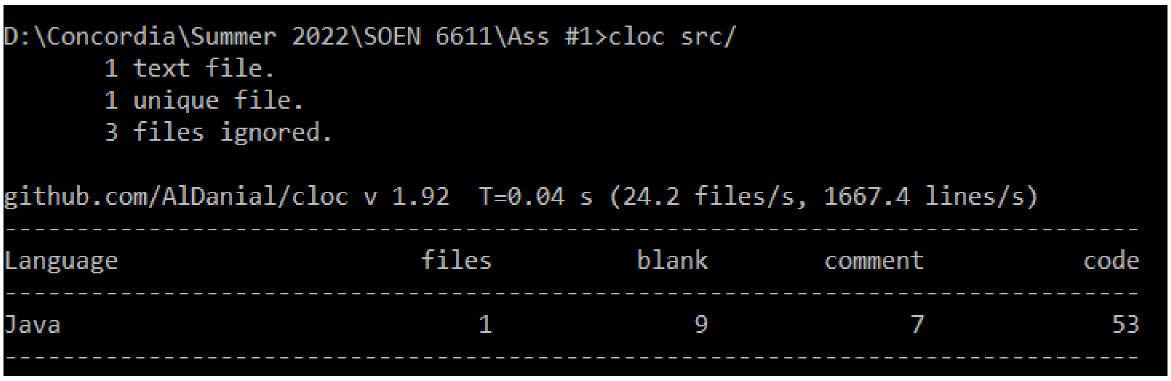
- Any line of program text regardless of the number of statements or fragments of statements on the line.
- Comments are not counted in SLOC
- Blank lines are not counted in SLOC
- Parentheses and curly braces are counted in SLOC

Sloc (Manual) = 53

b) Automatically, using a code length measurement tool of your choice

Cloc: [GitHub - AlDanial/cloc: cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.]()

SLOC (Automated tool: Cloc) = 53



```
D:\Concordia\Summer 2022\SOEN 6611\Ass #1>cloc src/
       1 text file.
       1 unique file.
       3 files ignored.

github.com/AlDanial/cloc v 1.92  T=0.04 s (24.2 files/s, 1667.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                             1              9              7             53
-------------------------------------------------------------------------------
```

SLOC (Manual)	**SLOC (Automated tool: Cloc)**	**Estimated effort (person-minutes)**	**Work effort (person-minutes)**	**Defects #**
53	53	60	70	1

Task 6: Interpret the results of 5-a. and 5-b.

The results for automated and manual physical lines of code are same 53.
This can be due to the similar rules used by the cloc counter for counting physical lines of code.
A physical SLOC is useful

- as it is used to predict the amount of effort that will be required to develop a program,
- as well as to estimate programming productivity or maintainability once the software is produced.
- It is language-independent as it does not take into account syntactic and other variations across different programming languages

But in terms of reliability it has drawbacks such as

- It is dependent of the stylistic conventions of the statements that are being counted.
- The rules are not related to the program's logic and cannot provide an accurate count of cases such as multiple logical statements residing on a single line, or that single logical statement spanning multiple lines

Importance of clearly defined rules in code length measurement:

It is highly important to have well-defined rules without any ambiguity in terms of code length measurement or any other productivity metric. If different tools are using different set of rules, it can lead to different results for the same source code which can create confusion and it will impact productive measures to be taken.

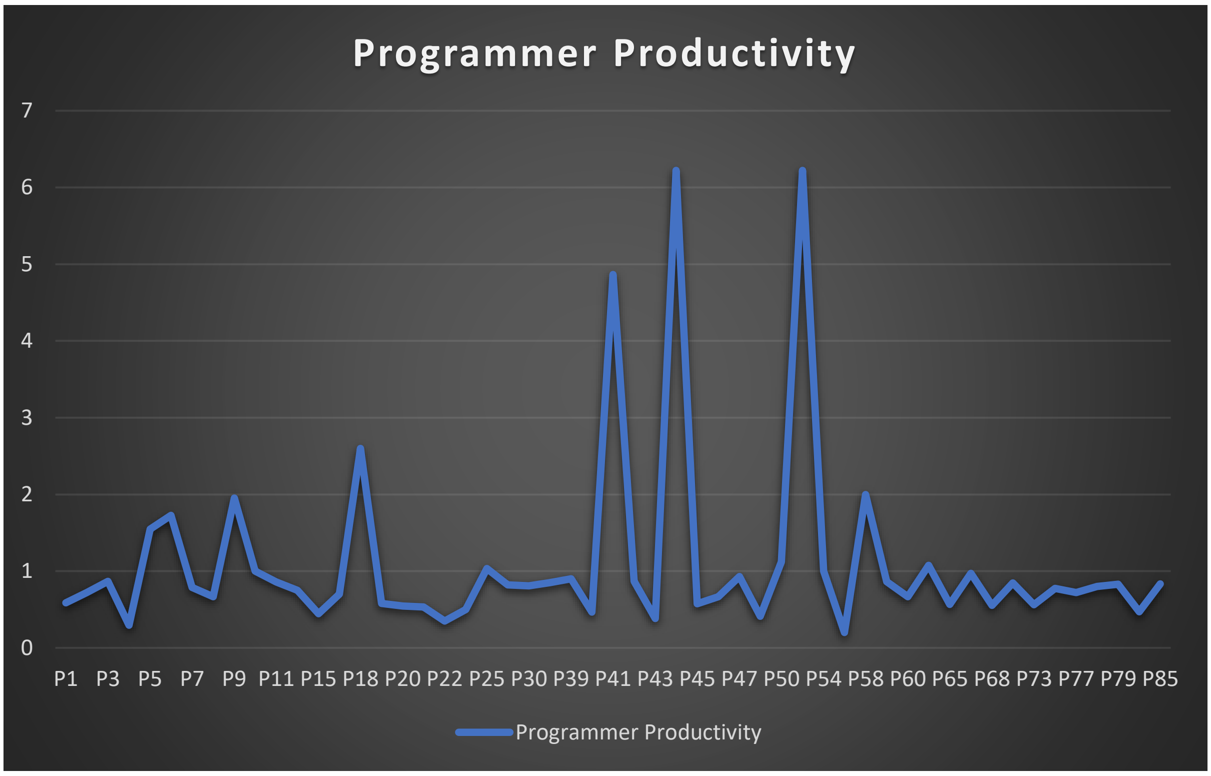
In addition, if there is an ambiguity in the rules defined for the calculation of length measurement, it will impact the count as it will lead to different results as the team will have to take assumptions and will have to work based on intuitions which can vary from team to team and as well as program to program. This will again skew the results and will impact the productive measures to be taken.

Goal 2: Simple analysis of coding productivity

Task 1: Calculate the Productivity of each Programmer:

Formula = Length of Code of Programmer X / Programmer X Effort

Plotted the productivity measurement results of the historical data on a graph, where x-axis represents the programmers and the y-axis is the productivity of the corresponding programmers (The excel file is attached in the submission ***A1-G2-HD-Prod-2022-Answer.xlsx***):



Task 2: Compute the mean and standard deviation:

Mean = Sum of Productivity values / Total Number of Programmers
= (0.588235294 +0.72 +0.866666667 +0.294444444 +1.545454545 +1.727272727 +0.789473684 +0.666666667 +1.95 +1 +0.857142857 +0.75 +0.444444444 +0.7 +2.6 +0.577777778 +0.544444444 +0.534883721 +0.35 +0.5 +1.033333333 +0.822222222 +0.80952381 +0.85 +0.9 +0.466666667 +4.866666667 +0.866666667 +0.383333333 +6.222222222 +0.575 +0.666666667 +0.928571429 +0.412698413 +1.12 +6.222222222 +1 +0.2 +2 +0.861111111 +0.666666667 +1.076923077 +0.566666667 +0.972972973 +0.555555556 +0.844444444 +0.56 +0.775 +0.722222222 +0.8 +0.830769231 +0.471698113 +0.833333333) / 53
= 58.89006432/53
= 1.111133289

Standard Deviation σ: 1.229568559

Calculation:

Count, N:	53
Mean, µ:	1.111133289

Steps

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}.$$

$$\sigma \text{ ^2} = \frac{\Sigma(xi - \mu)2}{N}$$

$$= \frac{(0.588235294 - 1.111133289) \text{ ^ } 2 + \ldots + (0.833333333 - 1.111133289) \text{ ^ } 2}{53}$$

$$= \frac{80.12745858791}{53}$$

= 1. 511838841281

σ = √1.511838841281

= 1.229568559

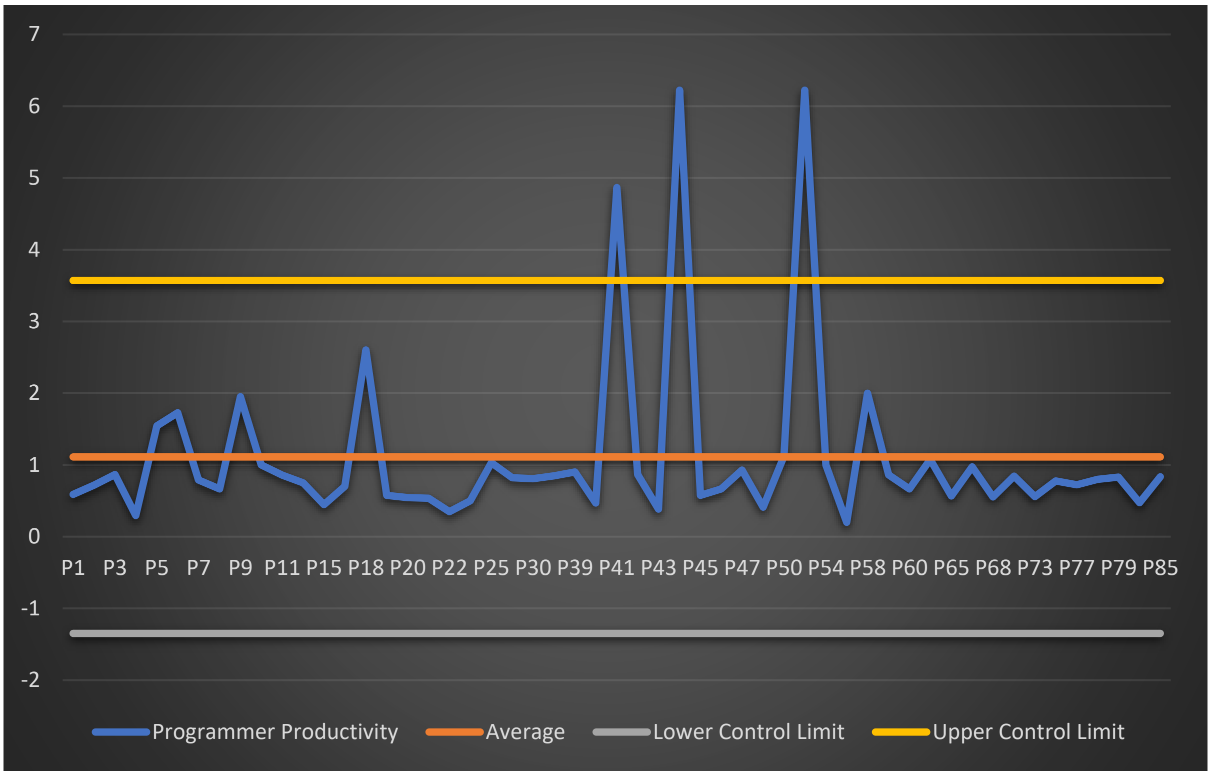
Control Limit:

The general rule of thumb for calculating control limits is:
(Average KPI Value) +/- (2 x (Standard Deviation))'

UCL (Upper Control Limit) = (Mean) + (2 X Standard Deviation)
= 1.111133289 + (2 X 1.229568559)
UCL = 3.570270407

LCL (Lower Control Limit) = (Mean) - (2 X Standard Deviation)
= 1.111133289 - (2 X 1.229568559)
LCL= -1.348003829

Graph with Average, Upper Control Limit and Lower Control Limit):



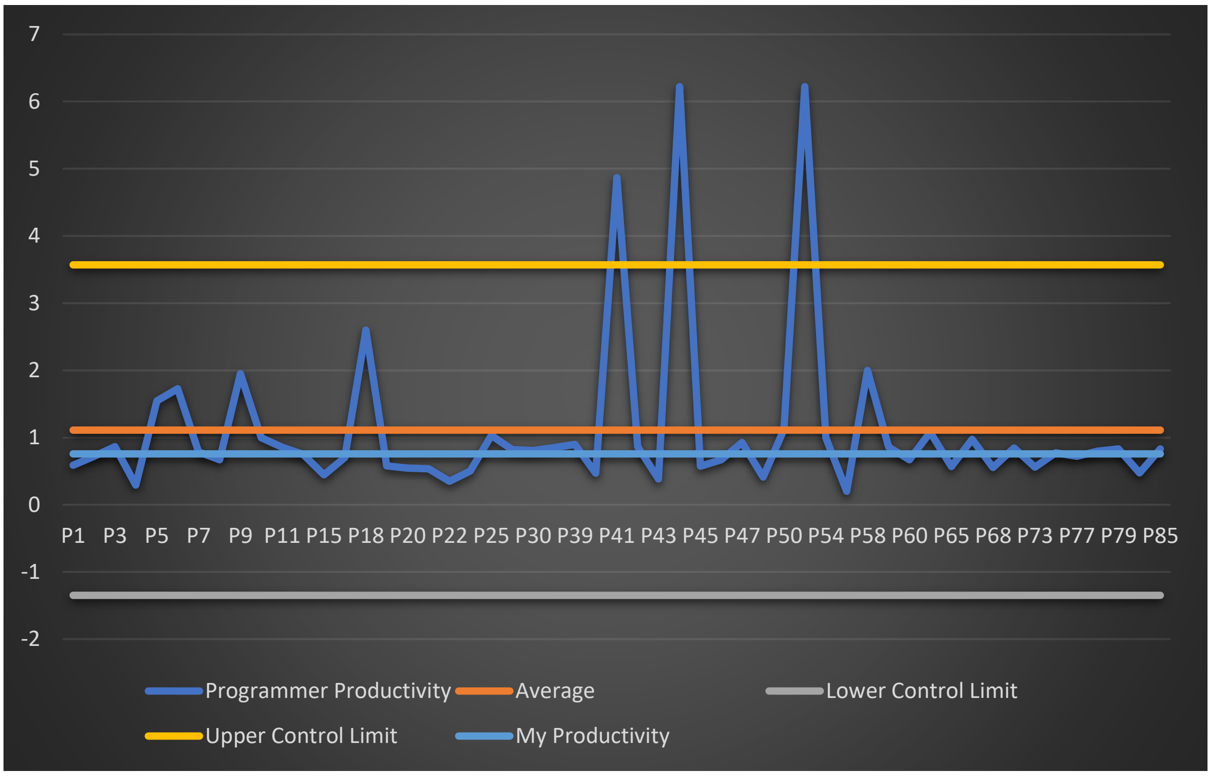
Task 3: Plot your actual productivity result on the graph from G2 Task 2

Actual Productivity of Programmer X = Length of Code of Programmer X / Actual Programmer X Effort

Length of Code of Programm = 53
Actual Time taken =70

Actual Productivity = 53/70
Actual Productivity = 0.757142857142

Graph with my productivity:



Analysis and Interpretation of the results:

It can be observed from the graph that my productivity is lower than the average productivity of the given programmers. The average productivity is of 1.111133289 whereas my productivity is 0.75714286.
The physical lines of code for my program was 53 whereas average SLOC for the given Concordia students was 41.83. Despite having more SLOC, my productivity was lower. It was due to the time taken for writing the code and time and effort spent in fixing the bug. Although SLOC leads to a simplistic measure of productivity but it does not take into account the 'effective' use of resources and creativity. Thus, it encourages quantity over quality.

As for the productivity for the last year Concordia students, the productivity for all the students were higher than the Lower Control Limit. Whereas for some programmers like **P41, P44 and P51**, the productivity was beyond the upper control limit. These are considered **outliers**.

- According to the data provided, P44 and P51 implemented 280 lines of code in 45 minutes and P41 implemented 73 lines of code in 15 minutes. These are special causes of variation.
- So further analysis of the code of these programmers should be done as to find what caused these cases of variation.
  - Was it due to programming language used?
  - Was it because of the rules decided by these programmers for counting SLOC?'

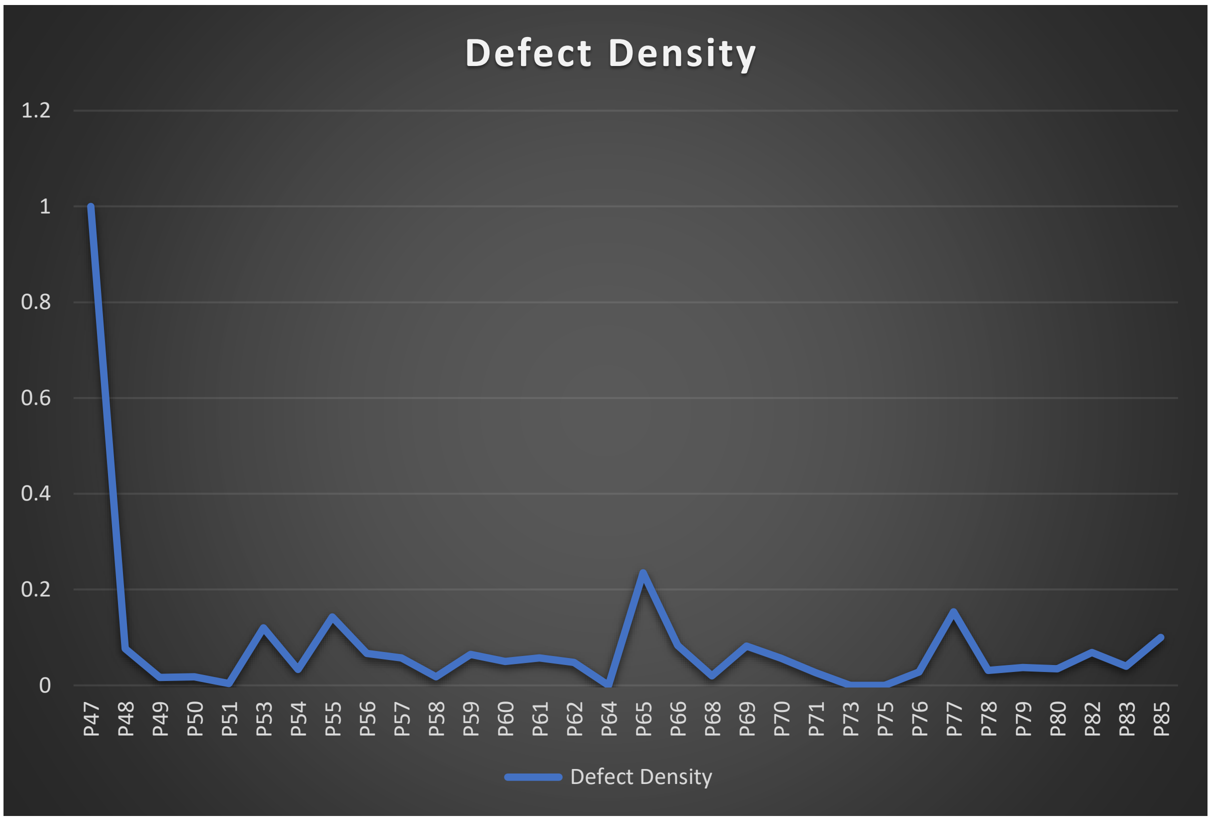
Questions like these helps us to understand the performance and productivity of the team.
If these cases were an exception, then it would better to remove them from the dataset otherwise they can impact the average productivity of the team and we cannot reliably use the average productivity for further analysis.

Goal 3:

Task 1: Calculate the defect density of each Programmer:

Formula = Programmer X # Defects / Programmer X Length of Code

The graph (The excel file is attached in the submission ***A1-G3-HD-Defects-2022-Answer.xlsx***):



Task 2: Compute the mean and standard deviation:

Mean = Sum of Defect Desnity/ Total Number of Programmers

= (1 +0.076923077 +0.016393443 +0.017857143 +0.003571429 +0.12 +0.033333333 +0.142857143 +0.066666667 +0.057142857 +0.017857143 +0.064516129 +0.05 +0.057142857 +0.047619048 +0 +0.235294118 +0.083333333 +0.02 +0.082089552 +0.056603774 +0.026315789 +0 +0 +0.027777778 +0.153846154 +0.03125 +0.037037037 +0.034482759 +0.068965517 +0.04 +0.1) / 32

= 2.768876078 /32

= 0.08653

Standard Deviation σ: 0.17146

Calculation:

Count, N:	32
Mean, μ:	0.08653

Steps

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}.$$

σ ^2 = $\frac{\Sigma(xi - \mu)2}{N}$

= $\frac{(1 - 0.08653) ^ 2 + ... + (0.1- 0.08653) ^ 2}{32}$

= $\frac{0.940758677}{32}$

= 0.029398708656

σ = √0.029398708656

= 0.17146

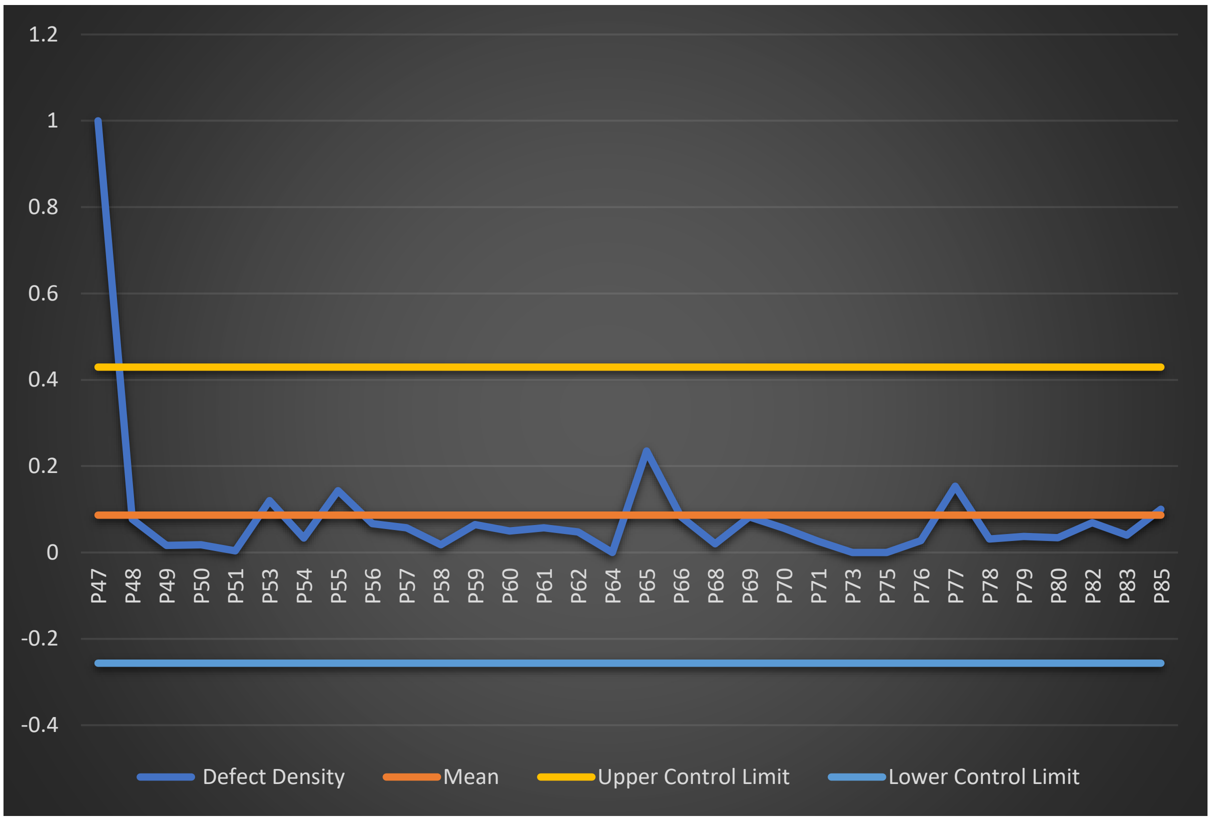
Control Limit:

The general rule of thumb for calculating control limits is:
(Average KPI Value) +/- (2 x (Standard Deviation))'

UCL (Upper Control Limit) = (Mean) + (2 X Standard Deviation)
= 0.086527377 + (2 X 0.171460516)
UCL = 0.42945

LCL (Lower Control Limit) = (Mean) - (2 X Standard Deviation)
= 0.086527377 - (2 X 0.171460516)
LCL= -0.2564

Graph with Average, Upper Control Limit and Lower Control Limit):



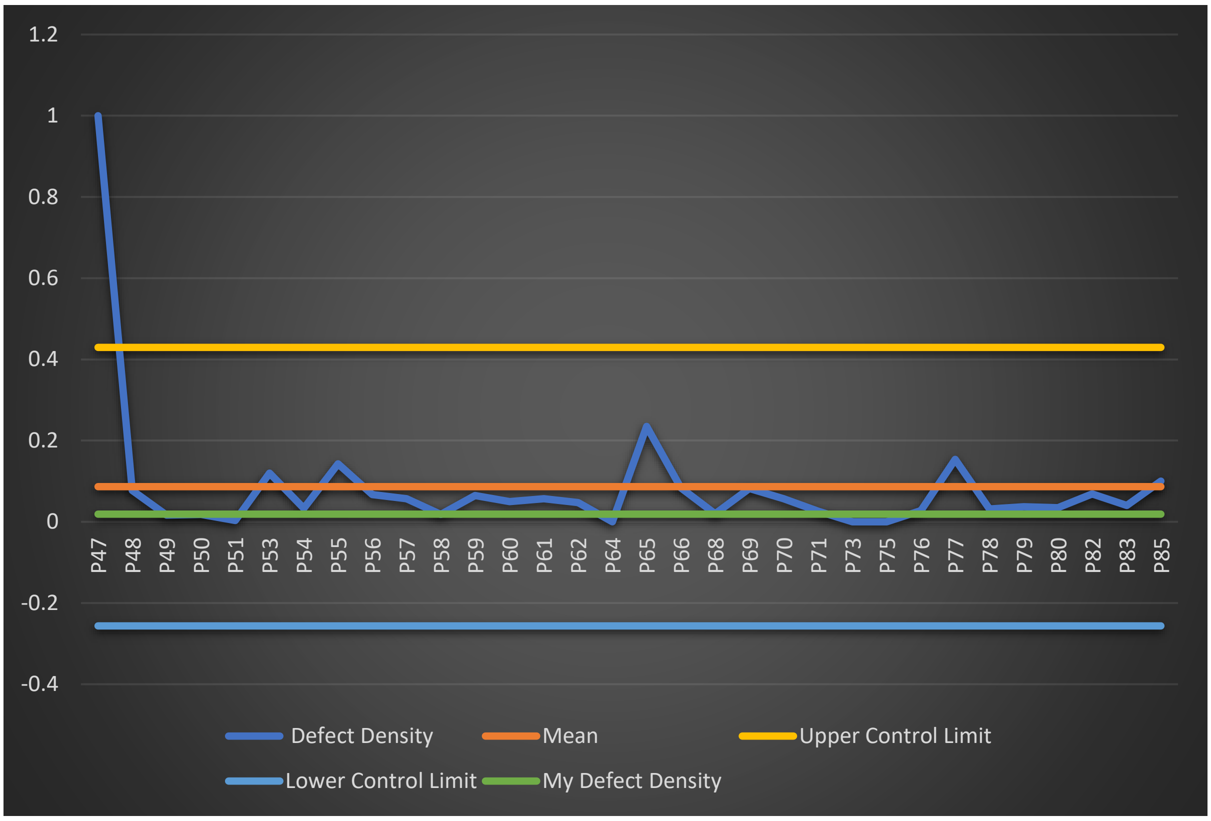
Task 3: Plot your defect density result on the graph from G3 Task 2

Defect Density of Programmer X = Programmer X # Defects / Programmer X Length of Code

# Defects = 1
Length of Code = 53

My Defect Density = 1/53
My Defect Density = 0.018867925

Graph with my productivity:



Analysis and Interpretation of the results:

It can be observed from the graph that my defect density is below mean which is a good sign.
As for the Concordia students, all the results are above lower control limit and only **P47** programmer has defect density higher than the control limit.
According to the data provided, defect density for **P47 is 1 with SLOC value of 26 and #defects of 26.** This is an outlier and requires further analysis to figure out the reason behind this high defect density.

- It raises the question if the sloc rules defined were correct or not?
- As well as if the defects counted were qualified?

If this was an exception, then it will be better to remove this from the historical dataset as it is impacting the overall average defect density of the students and we cannot reliably use the average defect density for further analysis
Apart from the above-mentioned special case, other programmers have defect density close to the mean. Which means, once dealing with the outlier, we can normalize the quality of the code using these defect densities.
Hence it is a good indicator for decision-smaking.