SOEN 6611

SOFTWARE MEASUREMENT

Descriptive-Statistics Calculator

Author:
VEDANSHI PATEL *(40068754)*
MRIDUL PATHAK *(40078157)*
HEMANSHU CHOURASIA *(40085060)*
MEHUL PRAJAPATI *(40076930)*
BHARTI SAINI *(40089008)*
HARSHKUMAR RAVAL *(40076067)*

Lecturer:
PROF. PANKAJ KAMTHAN

Descriptive-Statistics Calculator

CONTENTS

1. INTRODUCTION

The purpose of this project was to develop a DESCRIPTIVE-STATISTICS Calculator to quantitatively describe a collection of data; and to develop a set of interrelated artifacts for conducting certain measurements related to DESCRIPTIVE-STATISTICS Calculator.

Functions supported by DESCRIPTIVE-STATISTICS Calculator

1. *Minimum (m)*

To find minimum element from given array of values, we initialized minimum to the first element and then traversed the array, comparing each element and updated minimum whenever we found a value which is less than current minimum.

2. *Maximum (M)*

To find maximum element from given array of values, we initialized maximum to the first element and then traversed the array, comparing each element and updated maximum whenever we found a value which is greater than current maximum.

3. *Mode (o)*

To find the elements which has the highest frequency of occurrence from a given array of values.We Calculate the count of all the elements in the array and then check the count to find the maximum repeated element. There can be more than one mode value.

4. *Median (d)*

In this function, firstly we sorted the given array using merge sort algorithm as it has *nlogn* time complexity. Then, we computed median as the middle number if number of elements is odd, and as arithmetic mean of the two middle numbers if number of elements is even.

5. *Arithmetic mean (μ)*

The equation of arithmetic mean is = $\frac{1}{n}\sum_{i=1}^{n} x_i$

So, to find arithmetic mean, we computed sum of given numbers and then we divided then sum with total number of elements.

6. *Mean absolute deviation (MAD)*

The equation of MAD is = $\frac{1}{n}\sum_{i=1}^{n} |x_i - \mu|$

In this function, firstly we computed the arithmetic mean value. After that, we stored absolute difference between each data point and mean in one array. Then, we computed the sum of that array elements. Lastly, we divided the sum by the number of data points.

7. *Standard deviation (σ)*

The equation of standard deviation is = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$

In this function, firstly we the arithmetic mean value. After that, we computed sum of squared difference between each data point and mean. Finally,

we divided the sum by the number of data points and computed the square root of the result value to get standard deviation value.

2. GOAL-QUESTION-METRIC

Goal : Analyze the statistic-calculator with respect to its quality for Deliverable 1 (June 16, 2020) for the purpose of characterization from the developer's point of view which includes quality of documentation, quality of the coding, implementation of the functionalities, accuracy of the calculated data and performance of system.

[Question 1] How many unwanted behaviors?
Metric 1.1: Number of detected failures
Metric 1.2: Number of detected faults
Metric 1.3: the ratio of critical/uncritical failures
[Question 2] What is the quality of the test?
Metric 2.1: Code coverage
Metric 2.2: Number of detected failures
Metric 2.3: Number of detected faults
Metric 2.4: the ratio of critical/uncritical failures
[Question 3] Number of equations that can be stored and used at one point of time so that it is not overwhelmingly complicated and complex for the user to understand and implement .
Metric 3.1: Subjective evaluation of number of Equations available for storage and use in general scientific calculators
Metric 3.2: Subjective rating of users on the equations included for usage
[Question 4] : How much cost and effort will be required to build the system?
Metric 4.1: Use case Cost Model (UCP)
Metric 4.2: Cost Estimation Model (COCOMO)
[Question 5] What are the features needed for the correct analysis of the data set?
Metric 5.1: Number of required features.
Metric 5.2: Number of supplementary features.
Metric 5.3: Number of views available for different users based on use case.
Metric 5.4: Ease of Understanding the data displayed for large data sets.
[Question 6] How accurate is the result?
Metric 6.1: Margin of error
Metric 6.2: Behaviour in case of outliers
[Question 7] What is the accuracy of the system?
Metric 7.1: Percentage of errors discovered
Metric 7.2: How many errors are discovered
Metric 7.3: Number of unwanted behaviour/system crashes
[Question 8] How fast is the system?
Metric 8.1: Response time of the system
[Question 9] . Which type of basic statistical functions to be derived to collect qualitative data?
Metric 9.1: Standard deviation
[Question 10] : How statistical functions will be applied to monitor quality

control?

Metric 10.1: Flow charts

[Question 11] Are the plans kept up to date?

Metric 11.1: Are more than one technique used and compared in the estimation?

Metric 11.2: The number of estimation techniques used per prediction.

[Question 12] Are the Chosen metrics applicable?

Metric 12.1: The set of selected Metrics

Metric 12.2: Dependency (eg. Y/N is the the metric independent of another metric)

Metric 12.3: Applicability (eg. Y/N is the metric applicable to the estimate)

3. USE CASE MODEL AND USE CASE STORIES

A. Use Case Model

Figure S1 shows a Use Case Model for Descriptive-statistic calculator.

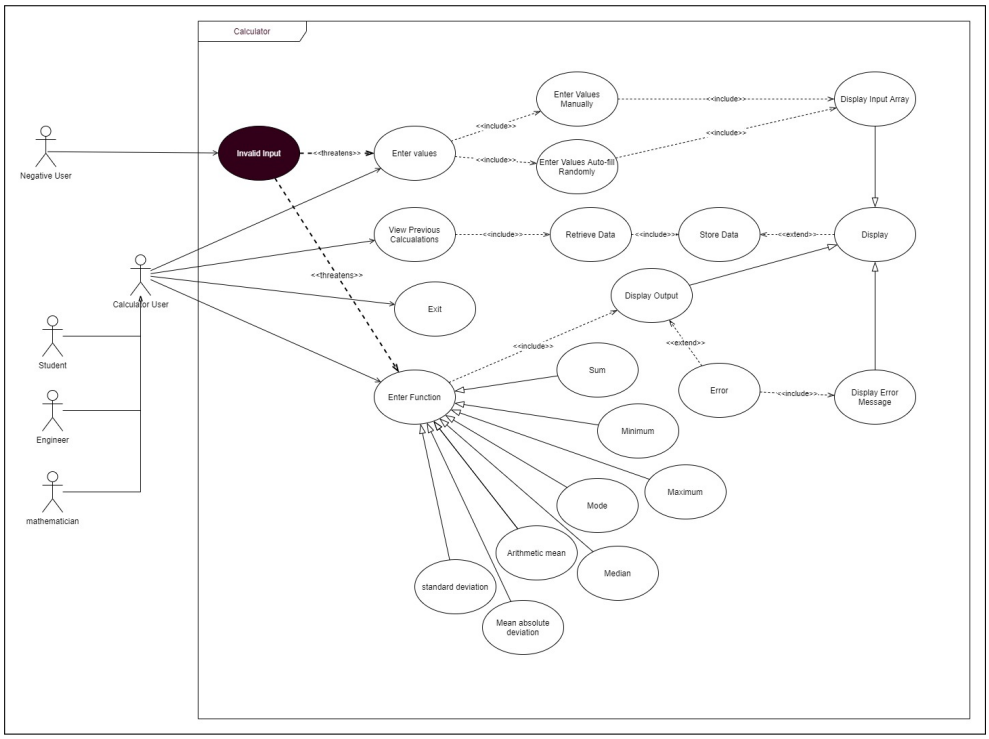


Fig. S1. Use Case Model for Descriptive-statistic calculator.

B. Use Case Stories

Table S1 to S6 shows Use Case Stories for for Descriptive-statistic calculator; where S1 represents the negative use case stories.

4. THE USE CASE POINTS

Use Case Points (UCP) is an estimation technique used for estimating effort based on use cases. Some of the advantages of using UCP approach for estimation are easy to learn and apply; this technique is broadly used; it is

Table S1. Use Case Story for UC1

Use case ID:	UC1
Use case name:	User's calculator session must be on.
Description:	Negative use case story: User is entering non-applicable function number (Invalid input).
Pre conditions:	User's calculator session must be on.
83cmStandard flow:	1. User is asked to enter '(1) To Enter values Manually' or '(2) To select values Randomly'. 2. User enters (2) to select values Randomly. 3. User is asked to 'Enter of values' 4. User enters '1000' 5. Calculator displays all the 1000 random values. 6. User is asked to enter "(1) Minimum (2) Maximum (3) Mode (4) Median (5) Arithmetic Mean(6) Mean Absolute Deviation (7) Standard Deviation (8) To Exit" 7. User enters '10' to calculate Median for the input array values. 8. Calculator will display an error message of Invalid function.
Post conditions:	Calculator will take the user back to step 6, and will ask the user to re-enter the function.
Comment:	- A user will be given a list of functions to select from. - The calculator is limited only to perform those operations. - If a user tries to enter the function beyond the list displayed, it won't match any case from the list and will result in notifying the user to only enter the operation listed.

Table S2. Use Case Story for UC2

Use case ID:	UC2
Use case name:	To display single a function
Description:	User selects a single function to view the calculation result of an input array value.
Pre conditions:	User's calculator session must be on.
83cmStandard flow:	1. User is asked to enter '(1) To Enter values Manually' or '(2) To select values Randomly'. 2. User enters (2) to select values Randomly. 3. User is asked 'How many numbers?' 4. User enters '1000' 5. Calculator displays all the 1000 random values. 6. User is asked to enter "(1) Minimum (2) Maximum (3) Mode (4) Median (5) Arithmetic Mean (6) Mean Absolute Deviation (7) Standard Deviation (8) Display All (9) To Exit" 7. User enters '4' to calculate Median for the input array values. 8. Calculator displays the result of the selected function.
Post conditions:	Calculator will take the user back to step 6, and will ask the user to re-enter the function. User can also choose to Exit.
Exception:	If the user selects other than given options, the calculator will display 'Exception: Invalid input'.

Table S3. Use Case Story for UC3

Use case ID:	UC3
Use case name:	To output for display result for all functions
Description:	User selects an option to view all functions results at once.
Pre conditions:	User's calculator session must be on.
83cmStandard flow	1. User is asked to enter '(1) To Enter values Manually' or '(2) To select values Randomly'. 2. User enters (2) to select values Randomly. 3. User is asked to 'Enter of values' 4. User enters '1000' 5. Calculator displays all the 1000 random values. 6. User is asked to enter "(1) Minimum (2) Maximum (3) Mode (4) Median (5) Arithmetic Mean (6) Mean Absolute Deviation (7) Standard Deviation (8) Display All (9) To Exit" 7. User enters '8' to calculate and display the result of all functions for the input array values. 8. Calculator will display the result of all seven listed functions.
Post conditions:	Calculator will take the user back to step 6, and will ask the user to re-enter the function. User can also choose to Exit.
Exception:	N/A

Table S4. Use Case Story for UC4

Use case ID:	UC4
Use case name:	The Exit function
Description:	User uses the Exit function to End the session.
Pre conditions:	User's current session must be ON.
83cmStandard flow:	1. User is asked to enter '(1) To Enter values Manually' or '(2) To select values Randomly'. 2. User enters (2) to select values Randomly. 3. User is asked to 'Enter of values' 4. User enters '1000' 5. Calculator displays all the 1000 random values. 6. User is asked to enter "(1) Minimum (2) Maximum (3) Mode (4) Median (5) Arithmetic Mean (6) Mean Absolute Deviation (7) Standard Deviation (8) Display All (9) To Exit" 7. User enters '4' to calculate Median for the input array values. 8. Calculation takes the user back to step-6 and asks to enter a function number. 9. User selects '9' to Exit the session. 10. Calculator will end the session.
Post conditions:	Calculator will save all the unsaved data.
Comment:	- The calculator will provide an exit function to the user. - The user may use the exit function and exit the calculator at any time. - The calculator application will close and all the unsaved data will be saved, and session will end.

Table S5. Use Case Story for UC5

Use case ID:	UC5
Use case name:	To retrieve previous Calculation history
Description:	User wants to retrieve saved previous calculation data
Pre conditions:	User's current session must be ON.
23cmStandard flow:	1. User selects option to retrieve previous data. 2. Calculator displays all the saved data.
Post conditions:	User is asked to select the next function.
Exception:	- If previous data history is empty, the calculator will display 'No saved data found.'

comparatively cheaper and does not required an extensive infrastructure to produce an estimate [1].

The equation for effort estimation consists of a variable, the Use Case Points (UCP), and a constant, the Productivity Factor (PF). Effort Estimate = UCP×PF [1].

A. The Use Case Points (UCP)

UCP equation consists of below three variables. Each variable is defined and calculated separately using weighted values, subjective values, and constraining constants.

1. The Unadjusted Use Case Points (UUCP)
2. The Technical Complexity Factor (TCF)
3. The Environment Complexity Factor (ECF).

UCP = UUCP × TCF × ECF

B. The Unadjusted Use Case Points (UUCP)

UUCP is an addition of Unadjusted Actor Weight (UAW) and the Unadjusted Use Case Weight (UUCW). UAW is based on the combined complexity of all the actors in all the use cases. UUCW is based on the total number of steps contained in all the use case scenarios. The UUCP is unadjusted because it does not account for the TCFs and ECFs [1].

UUCP = UAW + UUCW

B.1. The Unadjusted Actor Weight (UAW)

The UAW classifies Actor Types as simple, average and complex as shown in Table S7. The UAW is calculated by totalling the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the resulting products. As shown in Table S7, the Statistic-Calculator use case model has 1 average actor; therefore UAW = (1×0) + (2×1) + (3×0) = 2.

Table S6. Use Case Story for UC6

Use case ID:	UC6
Use case name:	To retrieve multiple functions results in multiple iterations
Description:	User retrieves multiple functions results in multiple iterations
Pre conditions:	User's current session must be ON.
23cmStandard flow:	1. User is asked to enter '(1) To Enter values Manually' or '(2) To select values Randomly'. 2. User enters (1) to enter values manually. 3. Calculator asks User to enter a list of values by space separated. 4. User enters numerical values. 5. User is asked to enter a function number to display calculation result '(1) Minimum (2) Maximum (3) Mode (4) Median (5) Arithmetic Mean(6) Mean Absolute Deviation (7) Standard Deviation (8)Display All (9) To Exit' 6. User enters '2' to find the maximum. 7. Calculator displays maximum value from the input data. 8. Calculator again asks the user to enter the next function number. 9. User enters '7' to calculate Standard Deviation. 10. Calculator displays Standard Deviation results. 11. Calculator again asks the user to enter the next function number. 12. User enters '9' to exit. 13. Calculator ends the session.
Post conditions:	Calculator will save all the unsaved data.
Exception:	N/A

Table S7. Computing UAW

Actor Type	Description	Weight	Number of Actors	Result = (Weight × Number of Actors)
Simple Actor	The actor represents another system with a defined application programming interface (API).	1	0	0
Average Actor	The actor is either another system that interacts through a protocol such as TCP/IP, or a person interacting through a text-based user interface(TUI).	2	1	2
Complex Actor	The actor is a person interacting through a graphical user interface (GUI).	3	0	0
Total UAW				2

B.2. The Unadjusted Use Case Weight (UUCW)

The UUCW classifies Use Case Types as simple, average and complex as shown in Table S8. Each use case is categorized by the number of transaction its scenario contains. The UUCW is calculated by totalling the number of use cases in each category, multiplying each total by its specified weighting factor, and then adding the resulting products. As shown in Table S8, the Statistic-Calculator use case model has 4 Simple and 2 Average Use Case; therefore UAW = (4×5) + (10×2) + (15×0) = 40.

The UUCP is calculated by adding UUCW and UAW [1]. For the data used in Tables S7 and S8, the UUCP = 40 + 2 = 42.

C. The Technical Complexity Factor (TCF)

TCF is used to account for technical concerns which can impact software development from its start to its end, including delivery. There are 13 standard technical factors used to estimate TCF that can impact on project productivity due to various technical issues [1]. Each factor is weighted according to its relative impact. Each factor's weight is multiplied by its alleged complexity factor to produce the calculated factor. The calculated factors are added to produce the Technical Total Factor.

TCF can be calculated as:

$$TCF = C_1 + [C_2 * \sum_{i=1}^{13}(W_T * F_T)] \tag{S1}$$

where C_1=0.6, C_2=0.01, W_T is the Technical Complexity Factor Weight, and $F_T is the Perceived Impact Factor$.

Assignment of the Perceived Impact Factor is subjective. Development team determines and assigns each Technical Complexity Factor a Perceived Impact Factor in the range of 0 to 5. A Perceived Impact Factor of 0 means the respective Technical Complexity Factor has no influence on the software project, 3 means average influence, and 5 means strong influence [1].

Table S8. Computing UUCW

Use Case Type	Description	Weight	Number of Use Cases	Result = (Weight × Number of Use Cases)
Simple	Simple user interface. Touches only a single database or knowledge entity. Its success scenario has 3 transactions or less. Its implementation involves less than 5 classes.	5	4 (UC1, UC2, UC3, UC5)	20
Average	More sophisticated user interface. Touches 2 or more database entities. Between 4 and 7 transactions. Its implementation involves between 5 and 10 classes.	10	2 (UC4, UC6)	20
Complex	Complex user interface or processing. Touches 3 or more database entities. More than 7 transactions. Its implementation involves more than 10 classes.	15	0	0
Total UUCW				45

Table S9 shows the Total Technical Factor for the Statistic-Calculator; therefore TCF = 0.6+(0.01*Total Technical Factor) = 0.6+(0.01*13.5)=0.735.

D. The Environmental Complexity Factor (ECF)

ECF is used to account for the development team's personal traits, including experience. More experienced teams will have a greater impact on the UCP computation than less experienced teams. There are 8 various Environmental Complexity Factors, each with its own weight [1].

The ECF equation is given by:

$$ECF = C_1 + [C_2 * \sum_{i=1}^{8} (W_E * F_E)] \quad \text{(S2)}$$

where $C_1 = 1.4, C_2 = 0.03$, W_E *is the Environmental Complexity Factor Weight and* F_E *is the Perceived Impact Factor corresponding to each Environmental Complexity Factor*.

Similar to Perceived Impact Factor for TCF, Perceived Impact Factor for ECF assignment is subjective. Development team assigns the Environmental Complexity Factor in range of 0 to 5. A Perceived Impact Factor of 0 means the respective Environmental Complexity Factor has no influence on the software project, 3 means average influence, and 5 means strong influence [1]. Table S10 shows the Total Environmental Factor for the Statistic-Calculator; therefore ECF = 1.4+(-0.03*Total Environmental Factor) = 1.4+(-0.03*30.5)=0.485.

Table S9. Computing TCF in the UCP approach

TCF Type	Description	Weight	Perceived Impact Factor	Result=(Weight×Perceived Impact)
T1	Distributed System	2	0	0
T2	Performance	1	3	3
T3	End User Efficiency	1	3	3
T4	Complex Internal Processing	1	0	0
T5	Reusability	1	3	3
T6	Easy to Install	0.5	0	0
T7	Easy to Use	0.5	3	1.5
T8	Portability	2	0	0
T9	Easy to Change	1	3	3
T10	Concurrency	1	0	0
T11	Special Security Features	1	0	0
T12	Provides Direct Access for Third Parties	1	0	0
T13	Special User Training Facilities are Required	1	0	0
Total Technical Factor				13.5

Table S10. Computing TCF in the UCP approach

ECF Type	Description	Weight	Perceived Impact Factor	Result=(Weight×Perceived Impact)
E1	Familiarity with Use Case Domain	1.5	5	7.5
E2	Part-Time Workers	-1	0	0
E3	Analyst Capability	0.5	3	1.5
E4	Application Experience	0.5	3	1.5
E5	Object-Oriented Experience	1	5	5
E6	Motivation	1	5	5
E7	Difficult Programming Language	-1	0	0
E8	Stable Requirements	2	5	10
Total Environmental Factor				30.5

E. Calculating UCP

As a reminder, the UCP equation is: UCP = UUCP * TCF * ECF From the above calculations, the UCP variables have the following values: UUCP=42, TCF=0.735, ECF=0.485. As result, UCP = 42*0.735*0.485 = 14.971 or 15 use case points.

F. The Estimated Hours

The total estimated number of hours for the project is determined by multiplying the UCP by the PF.Because no historical data has been collected, and it is a brand-new team, a value of 20 for PF is used for the first project [1]. Total Effort Estimate = UCP * PF = 15*20 = 300 person-hours.

Hence, the estimated effort of the Descriptive Statistics using UCP approach is 300 person-hours.

5. BASIC COCOMO 81

Effort Estimation based on COCOMO Model COCOMO is a cost model. The Constructive Cost Model (COCOMO) is an algorithmic, parametric, software project cost estimation model. The cost models provide direct estimates of effort (or duration). They are usually based on empirical data that reflects factors that contribute to overall cost.
The equation for effort estimation in the Basic COCOMO is:

$$E = a(S)^b F$$

where F be the effort adjustment factor.In this equation, E represents effort in person-months, S is software size in KDSI, the coefficients a and b depend on the type of software project, and F is the adjustment factor. In Basic COCOMO, there is no adjustment, and so F = 1.

Our Project is of organic type. So **a = 2.4, b = 1.05 and S = 0.61**
The value of S is obtained from the table based on the estimation of the previous projects and looking at the requirements of our project.

On calculating, **E = 1.42826 person-months**.
If we are taking total working days in a month as 22 days and number of working-hours per day as 8. Then,

$$E = 251.3737 person - hours$$

$$D = c(E)^d$$

In this equation, D represents development time in months, and the coefficients c and d depend on the type of software project. Our Project is of organic type. So **c = 2.5, d = 0.38**. On Calculating **D=20.42027**

No.	Functions and Features	Approx. LOC
1	Taking input in all formats	40
2	Sum Function	10
3	Min Function	10
4	Max Function	10
5	Mode Function	50
6	Median Function	15
7	Arithmetic Mean	15
8	Mean Absolute Deviation	30
9	Standard Deviation	20
10	Display Output	30
11	Driver Function (Main)	60
12	Select Function Menu	50
13	Error Logging	50
14	History Functionality	100
15	Miscellaneous and Utilities	120
Total LOC		610

Table S11. Estimating LOC based on the features

A. CONCLUSION

The difference in estimation between UCP and COCOMO approach is 48.63 person-hours which is somewhat significant. The factors contributing in this difference are :
1. Values of Perceived Impact Factor which are subjective and are assigned as per developers knowledge.
2. The assumption regarding person-hours for calculation of Productivity Factor. As we did not have any previous historical data, PF of 20 was used which could have contributed in higher effort estimation in UCP.
3. For UCP we have taken in account various use cases. That has caused inflation in the value of the effort estimation using UCP approach. Considering the factor that some use cases are not highly complex, the actual value would be less.

6. IMPLEMENTATION OF DESCRIPTIVE STATISTICS

We have used Java programming language to develop a descriptive statistics calculator from the scratch. When a user runs the calculator, it will ask in advance how many random number she/he wants to generate. Then the calculator will generate random numbers and it will again ask the user about which operations she/he wants to perform on these numbers. On successful entering the operation, the calculator will display the result value. All of the calculator operations which are done by user are logged in a file.
To run the calculator from command line, user can execute below command.

```
java -jar src/calculator.jar
```

```
***** DESCRIPTIVE-STATISTICS CALCULATOR *****
(1) To Enter values Manually
(2) To Select values Randomly
(3) To Terminate Calculator
2
How many numbers?
10
Input array: [307.0, 6.0, 842.0, 76.0, 426.0, 235.0, 125.0, 711.0, 251.0, 509.0]
Select one operation from below,
(1) Minimum
(2) Maximum
(3) Mode
(4) Median
(5) Arithmetic Mean
(6) Mean Absolute Deviation
(7) Standard Deviation
(8) All
(9) To Exit
```

Fig. S2. Entering the value for random numbers generation

7. CYCLOMATIC NUMBER

The cyclomatic complexity of a method is the number of possible linearly-independent execution paths through that .It was originally introduced as a complexity measure by **Thomas McCabe**.

The cyclomatic complexity metric is always provided as a whole number. The lower the number is, the fewer linear paths of execution the code con-

tains. A lower number implies fewer unit tests in order to ensure a given piece of software has tests for every conceivable permutation. A low McCabe cyclomatic complexity score of 4 or 5 is considered ideal. For calculating the McCabe cyclomatic complexity metric (M) edges (E), nodes (N) and connected components (P), We used the following formula:

M = E - N + 2P

To Calculate cyclomatic complexity, we can use some rules to simplify the calculation:

- Assign one point to account for the start of the method
- Add one point for each conditional construct, such as an "if" condition
- Add one point for each iterative structure
- Add one point for each case or default block in a switch statement
- Add one point for any additional Boolean condition, such as the use of && or ||

With exceptions, add each throws, throw, catch or finally block as a single point when calculating the McCabe cyclomatic complexity metric. However, given Java's fairly verbose exception handling semantics, counting exception related flows can unfairly malign a well coded method, **so it's sometimes wise to ignore the additional points exception handling adds to the McCabe cyclomatic complexity metric total.**

We used the tool **LocMetrics** (figure S2 figure S3)to calculate the McCabe VG Complexity and confirmed it manually through a source code flowchart(figure5.2) as well.

The McCabe VG Complexity score we attained was 58

But while manually calculating we attained a score of 51

Mainly because some of the exception handling loops were ignored in the process because they added extra complexity (some methods were counted once instead of multiple times)

Overall		
Symbol	**Count**	**Definition**
Source Files	3	Source Files
Directories	3	Directories
LOC	526	Lines of Code
BLOC	88	Blank Lines of Code
SLOC-P	311	Physical Executable Lines of Code
SLOC-L	239	Logical Executable Lines of Code
MVG	58	McCabe VG Complexity
C&SLOC	0	Code and Comment Lines of Code
CLOC	127	Comment Only Lines of Code
CWORD	554	Commentary Words
HCLOC	0	Header Comment Lines of Code
HCWORD	0	Header Commentary Words

C:\Users\pazim\Desktop\teamh-master - FOLDERS											
Folder	**Files**	**LOC**	**SLOC Physical**	**SLOC Logical**	**MVG**	**BLOC**	**C&SLOC**	**CLOC**	**CWORD**	**HCLOC**	**HCWORD**
Total	3	526	311	239	58	88	0	127	554	0	0
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611	3	526	311	239	58	88	0	127	554	0	0
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611\calculator	2	469	281	214	57	76	0	112	488	0	0
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611\utilities	1	57	30	25	1	12	0	15	66	0	0

Fig. S3. Metrics to calculate McCabe VG Complexity

C:\Users\pazim\Desktop\teamh-master - FILES										
File	**LOC**	**SLOC Physical**	**SLOC Logical**	**MVG**	**BLOC**	**C&SLOC**	**CLOC**	**CWORD**	**HCLOC**	**HCWORD**
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611\calculator\Calculator.java	106	78	63	11	28	0	0	0	0	0
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611\calculator\CalculatorFunctions.java	363	203	151	46	48	0	112	488	0	0
C:\Users\pazim\Desktop\teamh-master\src\main\com\soen6611\utilities\Logger.java	57	30	25	1	12	0	15	66	0	0

Fig. S4. Metrics to calculate McCabe VG Complexity

Class Name	Cyclomatic Number
Calculator.java	11
CalculatorFunctions.java	46
Logger.java	1

Table S12. Cyclometric Number

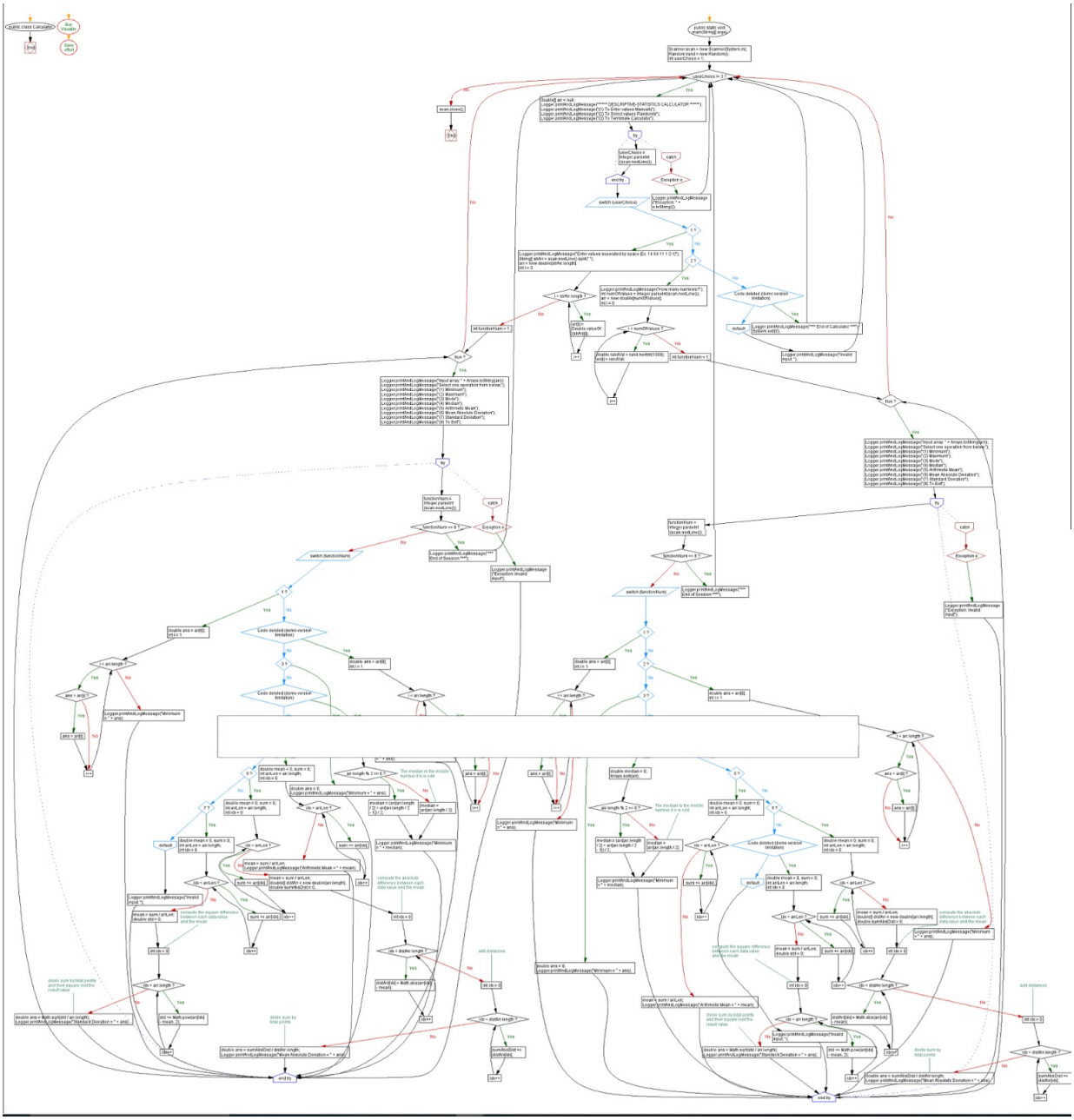


Fig. S5. Source Code FlowDiagram

Number of Nodes(N)=160 Number of edges(E)=205 Number of connected components(P)=3

Cyclomatic Complexity = E-N+2P
= 205-160+2*3
=51

Conclusion: Our calculated cyclomatic value falls under category of High risk assessment. This contributed as the major factor for not implementing a Graphical user Interface since the Cyclomatic Complexity obtained was 58.

8. OBJECT ORIENTED METRICS

Weighted Method Per Class (WMC)

Let C be a class with methods M1, . . . , Mn. Let c1(M1), . . . , cn(Mn) be the complexity (weights) of the methods. Then, Weighted Method Per Class (WMC) is calculated by:

$$WMC = \sum_{i=1}^{n} C_i(M_i) \tag{S3}$$

For our project as stated in part 5 we can see that,

Cyclomatic complexity of Calculator class = 11
Cyclomatic complexity of CalculatorFunctions class = 46
Cyclomatic complexity of Logger class = 1

As per the definition of WMC metrics it is sum of total of cyclomatic complexity of all the classes. This means the total complexity of project.
So, WMC of our project is given as: WMC = 11 + 46 + 1 WMC = 58

Coupling Factor (CF)

The coupling factor (CF) [Abreu, 1995b; Harrison, Counsell, Nithi, 1998] measures the average coupling between classes excluding coupling due to inheritance. Let C1, . . . , Cn be classes in an OOD, where n > 1.

Let IsClient(Ci, Cj) = 1, if class Ci has a relationship with class Cj; otherwise, it is 0.

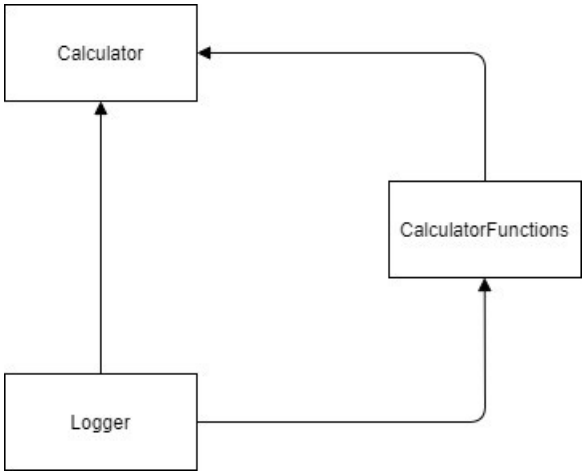
The semantics of the (is-client) relationship might be that an object of Ci calls a method in Cj, or has a reference to Cj, or to an attribute in Cj. However, the requirement is that this relationship cannot be inheritance.

Then, n2

(S4)

CF = $\sum_{i=1}^{n}(\sum_{j=1}^{n} IsClient(Ci, Cj)) \div ((nxn) - n)$

For our project we have three classes namely "Calculator, CalculatorFunctions, Logger" and there is client relation is shown below:



From diagram we can see that IsClient relation for Calculator class = 1.

From diagram we can see that IsClient relation for CalculatorFunctions class = 1.

From diagram we can see that IsClient relation for Logger class = 0.

Here n =3 that is number of classes.

As per formula, CF = (2 + 1 + 0) / 9 -3

= (3) / 6

CF = 0.5

Lack of Cohesion in Methods (LCOM)

DEFINITION: Let there be m methods M1,. . . , Mm. These methods access a attributes A1,. . . ,Aa.

Let (Ak) be the number of methods that access an attribute Ak.

Then,

$$LCOM = \frac{1}{a}[\sum_{i=\mu}^{a} A_i] - m/(1-m) \quad \text{(S5)}$$

So, for our project we have 2 methods for Logger class, 14 methods for CalculatorFunctions class and 2 for Calculator class.

Total methods = 2 + 14 + 2 = 18 methods.

Total number of attributes and number of methods that access those attributes are given below:

For class Logger total number of attributes = 9

For class CalculatorFunctions total number of attributes = 39

For class Calculator total number of attributes = 10

From this attributes LogInfo (Logger class) attribute of Logger class has been accessed by 3 methods.

From this attributes functionNum (CalculatorFunctions class) attribute of Logger class has been accessed by 2 methods.

From this attributes Scan (Calculator class, performanceStatistics method) attribute of Logger class has been accessed by 2 methods.

From this attributes arr (Calculator class, performanceStatistics method) attribute of Logger class has been accessed by 3 methods.

From this attributes arr (CalculatorFunctions class, getFunction method) attribute of Logger class has been accessed by 9 methods.

Other left attributes (53 attributes) are accessed 1 time only.

For the calculation m = 18 , a = 58

$\sum_{i=\mu}^{a} \mu(A_i)$ =53 + 2+3 + 2+3 + 9 = 72

LCOM* = [(72 / 58) – 18] / (1 – 18)

= [1.24 – 18] / (-17)

= [-16.75] / (-17)

LCOM* = 0.98

A. QUALITATIVE CONCLUSION

The quality attributes relevant to the metric are analyzability, modifiable, and testability.

WEIGHTED METHOD PER CLASS (WMC):

The number of methods and their complexity is an indicator of how much time and effort is required to maintain an individual class.

In particular, higher WMC means that more test cases can be required for decision coverage and that there is higher probability of finding bugs

So for our project we found out that class "CalculatorFunctions" have higher complexity as it has many methods and hence it will take more time and effort to maintain and it limits its reuse.

For other two classes as the complexity seems low they are easy to maintain and good for reuse.

COUPLING FACTOR (CF) :

High CF reflects deliberate high coupling to other classes. This, in turn, can be a negative to maintainability.

For our project CF value is 0.5 so its moderate which shows that project is moderate to maintain.

LACK OF COHESION IN METHODS (LCOM*) :

High LCOM* or LCOM* closer to 1 is an indication of low cohesion. This, in turn, can be a negative to maintainability (analyzability, modifiability, and testability .

There can be a number of reasons for a class to have low cohesion. The possible remedies include splitting the class into multiple child classes.

For our project **LCOM* value is 0.98** which indicate low cohesion and indicate that our project needs to be divided into multiple classes to increase cohesion.

9. PHYSICAL SLOC AND LOGICAL SLOC

There are many tools (LocMetrics, Unified Code Count etc) available to count the PSLOC and the LSLOC. The tool used here is `Unified Code Count`.

A. Counting Scheme

A.1. Rules for counting Physical SLOC

One Physical line starts with a character and ends with a carriage return or an end of line marker and excludes comments and blank lines.

A.2. Rules for counting Logical SLOC

The code can have statements or blocks. A statement is the one which terminates with a semicolon. It is counted as a single Logical line of code. A block can be a if-else block , a do-while loop, a for loop etc and it can have any number of statements but in Logical SLOC, a block is counted as only 1

logical line of code.

Below is a table of rules of counting taken from the UCC-Java-2018.05-master's documentation of coding standards for java for UCC. (SLOC Counting Standards for Java) [source : https://cssedr.usc.edu:4443/csse-tools/ucc-j/UCC-Java-2018.05/-/blob/master/docs/Counting%20Standards/Counting_Standards_Java.pdf]

PHYSICAL SLOC COUNTING RULES

MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One Per line	Defined in 1.8
Non-executable Lines			
Declaration (Data) lines	2	One per line	Defined in 1.4
Compiler Directives	3	One per line	Defined in 1.5
Comments			Defined in 1.7
On their own lines	4	Not Included (NI)	
Embedded	5	NI	
Banners	6	NI	
Empty Comments	7	NI	
Blank Lines	8	NI	Defined in 1.6

LOGICAL SLOC COUNTING RULES

NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	"for", "while", "foreach" or "if" statement	1	Count Once	"while" is an independent statement.
R02	*do {...} while (...); statement*	2	Count Once	Braces {...} and semicolon ; used with this statement are not counted.
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement	Semicolons within "for" statement are not counted. Semicolons used with R01 and R02 are not counted.
R04	Block delimiters, braces {...}	4	Count once per pair of braces {..}, except where a closing brace is followed by a semicolon, i.e. };or an opening brace comes after a keyword "else".	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by {...}.
R05	Compiler Directive	5	Count once per directive	

Fig. S6. SLOC Counting Rules for SLOC from UCC documentation for Java [Source : https://cssedr.usc.edu:4443/csse-tools/ucc-j/UCC-Java-2018.05/-/blob/master/docs/Counting%20Standards/Counting_Standards_Java.pdf]

So, to calculate the Physical and Logical SLOC, after downloading the Unified Code Count, we used the cmd line feature to calculate the SLOC by running the command:

```
java -jar ucc-j_2018.05.jar -dir F:/project_data/teamh-master/teamh-master/src/main
```

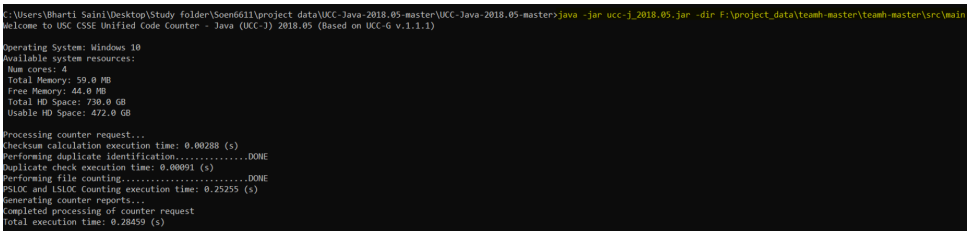


Fig. S7. A screenshot from cmd after executing the above command

High resolution image: https://drive.google.com/file/d/1ETFFXFkMkWzuvqvRepUoltF9HUlEBY
view?usp=sharing

The teamh-master was the source directory of our code. The main folder had the source code of the project. The test folder was not included in our SLOC calculations.

After running the command in the cmd prompt, we found that a few files were created in the source folder of our UCC tool. These included `JAVA_outfile.csv`, `outfile_cplx.csv`, `outfile_cyclomatic_cplx.csv`, `outfile_maintainabil` `outfile_summary.csv`.

Below is an attached screenshot of the java_outline.csv file generated by the tool for our project's main source code :

SLOC COUNT RESULTS
Generated by USC CSSE Unified Code Counter - Java (UCC-J) 2018.05 (Based on UCC-G v.1.1.1) on 06/14/2020
UCC-G -dir F:/project_data/teamh-master/teamh-master/src/main

Counting rules based on JAVA Version: Java 8

RESULTS FOR JAVA FILES

Total Lines	Blank Lines	Comments Whole	Comments Embedded	Compiler Direct.	Data Decl.	Exec. Instr.	Logical SLOC	Physical SLOC	File Type	Module Name
105	27	0	0	3	6	52	61	78	CODE	F:\project_data\teamh-master\teamh-master\src\main\com\soen6611\calculator\Calculator.java
362	47	112	0	11	23	117	151	203	CODE	F:\project_data\teamh-master\teamh-master\src\main\com\soen6611\calculator\CalculatorFunctions.java
56	11	15	0	4	3	17	24	30	CODE	F:\project_data\teamh-master\teamh-master\src\main\com\soen6611\utilities\Logger.java

RESULTS SUMMARY

Total Lines	Blank Lines	Comments Whole	Comments Embedded	Compiler Direct.	Data Decl.	Exec. Instr.	SLOC	File Type	SLOC Definition
523	85	127	0	18	32	261	311	CODE	Physical
523	85	127	0	18	32	186	236	CODE	Logical

Fig. S8. SLOC Results from UCC Tool

High resolution image: https://drive.google.com/file/d/1j_Kdzmug_S5uidwzDPW8rGKWzj5JMAmV/view?usp=sharing

Table S13. SLOC Tool Results

File Name	Physical SLOC	Logical SLOC
Calculator.java	78	61
CalculatorFunctions.java	203	151
Logger.java	30	24
Total SLOC	311	236

Physical SLOC = 311
Logical SLOC = 236

10. SCATTER PLOT

A scatter plot reveals relationships or association between two variables. The main objective using scatter plot is to view the relative positions of pairs of data points of WMC and Logical SLOC on the 2D plane. This plot helps to find patterns and trends in data.

A. Analysis of Scatter Plot

We have 3 pairs of data to generate a correlation between logical SLOC and WMC. As it can be seen from the plot, there is a linear non-decreasing relation between logical SLOC and WMC.

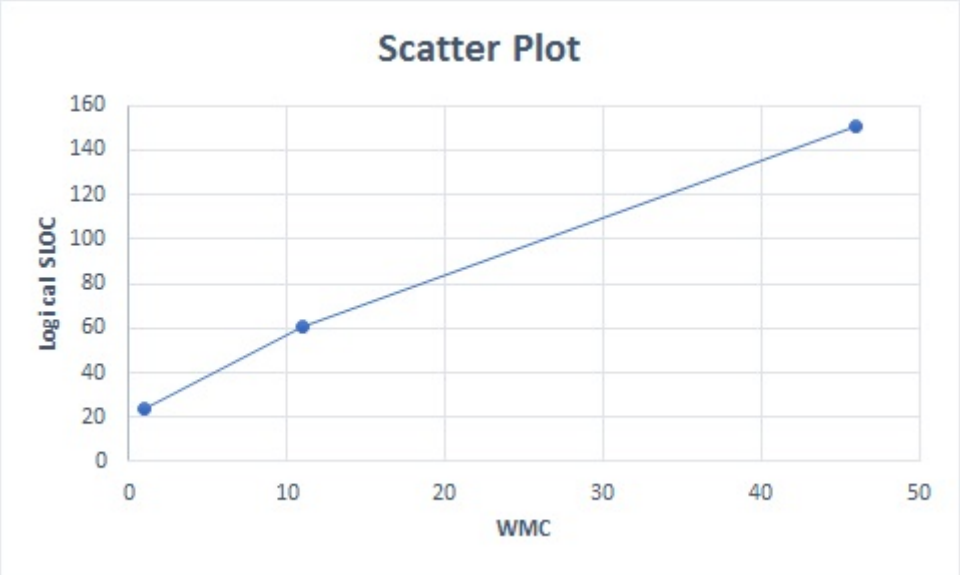


Fig. S9. WMC-Logical SLOC scatter plot

B. Correlation coefficient

A Pearson correlation is a number between -1 and $+1$ that indicates how strongly two variables are linearly related.

Table S14. values for calculating correlation

x	y	xy	x^2	y^2
1	24	24	1	576
11	61	671	121	3721
46	151	2346	2116	2601

$$r = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{n(\sum x^2) - (\sum x)^2}\sqrt{n(\sum y^2) - (\sum y)^2}}$$

After substituting values in this formula, we got r = 0.5376
We have a positive correlation between the two variables. Positive value indicate a relationship between x and y variables such that as values for x increases, values for y also increase. So, we can conclude that there is a strong positive correlation between logical SLOC and WMC.

11. CONTRIBUTION TABLE

Table S15. Task done by each team member :

Team Member	Contribution
Vedanshi Patel	- GQM - Use Case Model - Use Case Stories - Latex Documentation - Effort Estimation using The UCP - Coded initial standard flow for the calculator - Developed the Minimum and Maximum functions
Mridul Pathak	-GQM - Latex Documentation - Worked on Mean function - Calculation of Cyclomatic complexity
Hemanshu Chourasia	- GQM - Basic COCOMO - Developed Mode function - Latex Documentation
Mehul Prajapati	- Code review and code merging on GitHub - Analysis of Scatter Plot - Use case story to display a single function - Calculation of Correlation coefficient - Unit test cases - Latex Documentation - Developed Median, Mean, Mean absolute deviation and Standard deviation functions - Developed the logic for session restart and logging calculator operations
Bharti Saini	- GQM - Unit Test cases - Latex Documentation - Physical and Logical SLOC
Harshkumar	- Latex Documentation - WMC, CF and LCOM* calculations

12. GLOSSARY

Table S16. Glossary :

Term	Definition
GQM	Goal Question Metric
WMC	Weighted Method Per Classes
CF	Coupling Factor
LCOM	Lack of Cohesion in Methods
SLOC	Source Line of Code
API	Application Program Interface
UCM	Use Case Model
UCP	Use Case Points
UUCP	Unadjusted Use Case Points
UAW	Unadjusted Actor Weight
UUCW	Unadjusted Use Case Weight
UPreW	Unadjusted Precondition Weight
TCF	Technical Complexity Factor
ECF	Environment Complexity Factor
PF	Productivity Factor
WTi	Technical Complexity Factor Weight
Fi	Perceived Impact Factor
WEi	Environmental Complexity Factor
COCOMO	Constructive Cost Model
KDSI	Kilo Delivered Source Instructions
UCC	Unified Code Count
Cyclomatic Number	It is the parameter used as an indicator of the internal complexity i.e. complexity related to the structure of the source code.

REFERENCES

1. Clemmons, Roy. *Project Estimation With Use Case Points* CrossTalk, The Journal of Defense Software Engineering:19, 2006.

2. http://www.locmetrics.com/index.html Tool used to calculate Mc.Cabe Vg complexity

3. https://cssedr.usc.edu:4443/csse-tools/ucc-j/UCC-Java-2018.05/-/blob/master/docs/Counting%20Standards/Counting_Standards_Java.pdf

4. For COCOMO insight refereed to the following : https://www.researchgate.net/publication/3249859721_Performance_Evaluation_of_COCOMO_Vs_UCP

5. https://www.overleaf.com/learn/latex/Commands Latex documentation and format reference