Software Measurement (SOEN6611)

Summer 2023

Descriptive Statistics

Team "Amsterdam Cartel"

Deliverable 2

Unnati Chaturvedi, Mengqi Liu, Lei Zhou, Hema Reddy Mupppidi

July 28, 2023

Contents

List of Symbols and Abbreviations

GQM	Goal Question Metric
UC	Use Case
SLOC	Source Line of Codes
SLOC(L)	Logical SLOC
UCP	Use Case Point
PF	Productivity Factor
UUCP	Unadjusted Use Case Points
TCF	Technical Complexity Factor
Ecf	Environmental Complexity Factor
UAW	Unadjusted Actor Weight
UUCW	Unadjusted Use Case Weight
WTi	Technical Complexity Factor Weight
Fi	Perceived Impact Weight
WEi	Environmental Complexity Factor Weight

List of Figures

List of Tables

1 Background Information

To develop the Sales Analytics System, named METRICSTICS, a critical system will be implemented in order to comprehensively analyze sales performance.This system will examine statistical analysis of sales data to empower the sales management team in effectively monitoring sales trends over time, conducting thorough analyses of sales history, and making informed decisions based on the insights gained. Both sales staff and sales administration personnel will have access to METRICSTICS, allowing the sales team to diligently input sales data and the sales manager to effortlessly access statistical information for specific time periods. Additionally, METRICSTICS will enable the generation of comprehensive reports on a monthly, quarterly, and yearly basis, which will be presented to the board of members.

*Note: The key stakeholders for METRICSTICS are sales representatives, sales managers

2 Problem 3 : Use Case Points(UCP)

2.1 Effort Estimate using Use Case Point approach

Effort Estimation using use case points is given by,

$$EffortEstimate = UCP * PF \tag{2.1}$$

where UCP is Use Case Points, PF is Productivity Factor.

$$UCP = UUCP * TCF * ECF \tag{2.2}$$

where UUCP is Unadjusted Use Case Points, TCF is Technical Complexity Factor, ECF is Environmental Complexity Factor.

$$UUCP = UAW + UUCW \tag{2.3}$$

where UAW is Unadjusted Actor Weight, UUCW is Unadjusted Use Case Weight.
In our use case model, we have only one actor i.e; user who interacts with the system using graphical user interface. so, the actor is a complex actor with 3 points.

UAW=3

We have 5 classes in the system.so, it comes under average use case.

UUCW=10

From 2.3;

UUCP=UAW+UUCW
=3+10
=13

TCF concerns for the technical concerns that can impact the software project from its inspection to its conclusion, including delivery. There are 13 different Technical Complexity Factors, each with its own weight.
T1,T10,T11,T12,T13 has no influence;
T2,T3,T4 has average influence;
T5,T6,T7,T8,T9 has strong influence.

TCF Type	Description	Weight
T1	Distributed System	2
T2	Performance	1
T3	End User Efficiency	1
T4	Complex Internal Processing	1
T5	Reusability	1
T6	Easy to Install	0.5
T7	Easy to Use	0.5
T8	Portability	2
T9	Easy to Change	1
T10	Concurrency	1
T11	Special Security Features	1
T12	Provides Direct Access for Third Parties	1
T13	Special User Training Facilities are Required	1

Table 2.1: The Technical Complexity Factors in the UCP approach.

Value of no influence=0
Value of average influence=3
Value of strong influence=5

$$TCF = C1 + (C2 * \sum_{i=1}^{13}(WTi * Fi)) \tag{2.4}$$

c1=0.6 and c2=0.01
WTi is Technical Complexity Factor Weight
Fi is Perceived Impact Factor.

From 2.4;

TCF=0.6+(0.01*((2*0)+(1*3)+(1*3)+(1*3)+(1*5)+(0.5*5)+(0.5*5)+(2*5)+(1*5)+(1*0)+(1*0)+(1*0)+(1*0)))
=0.6+(0.01*44)
=1.04

The purpose of ECF is to account for the development team's personal traits, including experience. In general, the more the development team's experience, the more the impact of ECF on UCP.

Case: E1, E3, E4, E5, E6, and E8
0-No influence
1-Strong,Negative influence
3-Average influence

TCF Type	Description	Weight
E1	Familiarity with Use Case Domain	1.5
E2	Part-Time Workers	-1
E3	Analyst Capability	0.5
E4	Application Experience	0.5
E5	Object-Oriented Experience	1
E6	Motivation	1
E7	Difficult Programming Language	-1
E8	Stable Requirements	2

Table 2.2: The Environmental Complexity Factors in the UCP approach.

5-Strong,Positive influence

Case: E2, E7
0-No influence
1-Strong Favourable influence
3-Average influence
5-Strong, Unfavourable influence.

$$TCF = C1 + (C2 * \sum_{i=1}^{8}(WEi * Fi)) \tag{2.5}$$

c1=01.4 and c2=-0.03
WEi is Environmental Complexity Factor Weight
Fi is Perceived Impact Factor.

From 2.5;

TCF=1.4+(-0.03*((1.5*3)+(-1*0)+(0.5*5)+(0.5*5)+(1*5)+(1*5)+(-1*1)+(2*5)))
=1.4+(-0.03*28.5)
=0.145

From 2.2;

UCP=UUCP*TCF*ECF
=13*1.04*0.145
=1.9604

From 2.1;

Effort Estimate=UCP*PF
=1.9604*20
=39.208

2.2 Effort Estimate difference between using UCP approach and Actual Effort

The difference between the estimates obtained using the Use Case Points (UCP) approach and the actual effort towards the project can vary for several reasons:

Complexity Assessment: The UCP approach relies on the subjective assessment of various factors such as actor complexity, use case complexity, technical complexity, environmental complexity, and experience factor. Depending on the accuracy of these assessments, the UCP estimate may differ from the actual effort.

Changing Requirements: During the software development lifecycle, requirements may change or evolve, leading to modifications in use cases, actors, or system complexities. If these changes are not adequately reflected in the UCP estimate, it can lead to discrepancies with the actual effort.

Human Factors: The UCP approach depends on human judgment and expertise for assessing complexity factors. Individual biases, experience levels, and interpretations can impact the accuracy of the estimation.

External Factors: Factors outside the development team's control, such as external dependencies, integration challenges, unexpected issues, or resource constraints, can affect the actual effort required, even if the UCP estimate was reasonably accurate.

Skill and Efficiency: The actual effort can be influenced by the skill and efficiency of the development team. A highly skilled and experienced team may complete the project with less effort than estimated, while a less experienced team might take longer than expected.

Development Methodology: The UCP approach assumes a traditional waterfall-style development methodology, which may not be suitable for all projects. Agile or iterative development approaches, which emphasize adaptive planning and incremental development, can have different effort dynamics.

Project Management: The effectiveness of project management practices in monitoring and controlling the development process can significantly impact the actual effort. Properly managed projects are more likely to stay on track with the estimates.

3 Problem 4 : Introduction of Project METRICSTICS

3.1 Tech Stack

JavaFX is a powerful and versatile graphical user interface (GUI) toolkit and framework provided by Oracle for creating visually appealing and interactive desktop applications in Java. It emerged as a replacement for Swing, another GUI library for Java, and offers improved features and modern design elements to build rich and engaging user interfaces.

JavaFX can be used independently or alongside other Java technologies, such as Java Swing and Java 2D, to extend the functionality of Java applications. It provides a modern and versatile approach to building desktop applications, making it a popular choice for Java developers seeking to create dynamic and visually appealing user interfaces.

Key features of JavaFX include:

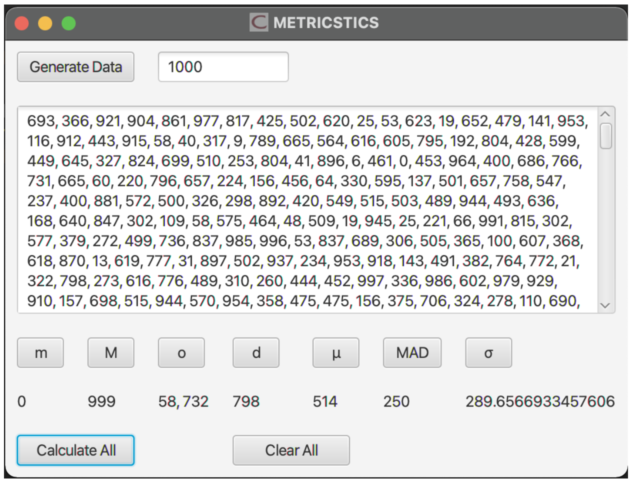
Rich UI Controls: JavaFX provides a wide range of pre-built UI controls such as buttons, text fields, labels, tables, charts, and more, making it easier to create feature-rich and intuitive interfaces.

Scene Graph: JavaFX utilizes a scene graph to represent the hierarchical structure of elements in the user interface. This approach allows for efficient rendering and manipulation of visual components.

CSS Styling: Styling in JavaFX is done using Cascading Style Sheets (CSS), just like web development. This separation of style from the application logic promotes better code organization and maintainability.

FXML: JavaFX supports FXML, an XML-based markup language that allows developers to define the user interface layout separately from the application logic. This enables a clear separation of concerns, making it easier for designers and developers to collaborate.

JavaFX includes powerful animation support, allowing developers to create smooth and visually appealing transitions and effects for their applications.



```
1000

693, 366, 921, 904, 861, 977, 817, 425, 502, 620, 25, 53, 623, 19, 652, 479, 141, 953,
116, 912, 443, 915, 58, 40, 317, 9, 789, 665, 564, 616, 605, 795, 192, 804, 428, 599,
449, 645, 327, 824, 699, 510, 253, 804, 41, 896, 6, 461, 0, 453, 964, 400, 686, 766,
731, 665, 60, 220, 796, 657, 224, 156, 456, 64, 330, 595, 137, 501, 657, 758, 547,
237, 400, 881, 572, 500, 326, 298, 892, 420, 549, 515, 503, 489, 944, 493, 636,
168, 640, 847, 302, 109, 58, 575, 464, 48, 509, 19, 945, 25, 221, 66, 991, 815, 302,
577, 379, 272, 499, 736, 837, 985, 996, 53, 837, 689, 306, 505, 365, 100, 607, 368,
618, 870, 13, 619, 777, 31, 897, 502, 937, 234, 953, 918, 143, 491, 382, 764, 772, 21,
322, 798, 273, 616, 776, 489, 310, 260, 444, 452, 997, 336, 986, 602, 979, 929,
910, 157, 698, 515, 944, 570, 954, 358, 475, 475, 156, 375, 706, 324, 278, 110, 690,

0         999       58, 732   798       514       250       289.6566933457606
```

Figure 3.1: METRICSTICS Application

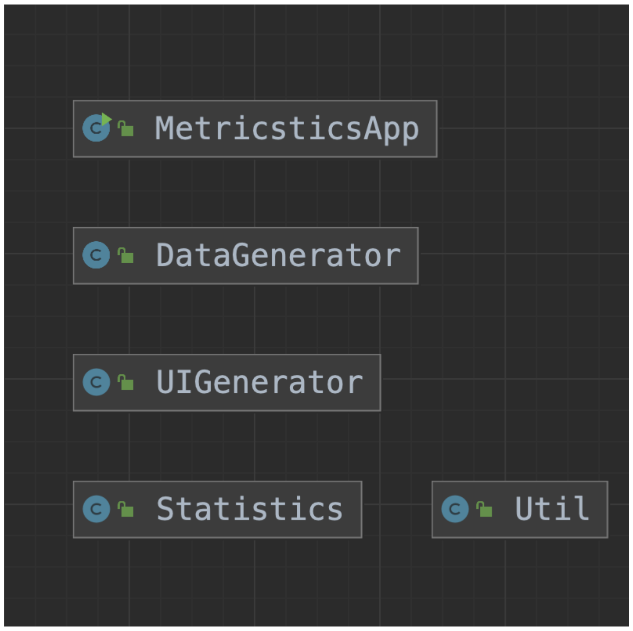
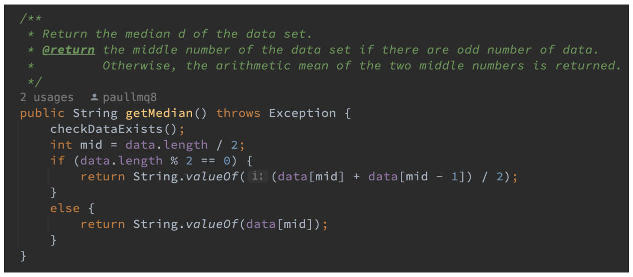


Figure 3.2: Java Classes



```java
/**
 * Return the median d of the data set.
 * @return the middle number of the data set if there are odd number of data.
 *         Otherwise, the arithmetic mean of the two middle numbers is returned.
 */
2 usages   paullmq8
public String getMedian() throws Exception {
    checkDataExists();
    int mid = data.length / 2;
    if (data.length % 2 == 0) {
        return String.valueOf( i: (data[mid] + data[mid - 1]) / 2);
    }
    else {
        return String.valueOf(data[mid]);
    }
}
```

Figure 3.3: Code Sample

4 Problem 5 : Cyclomatic number

4.1 Cyclomatic number

In this application, three distinct classes, namely MetricsticsApp, Statistics, and DataGenerator, have been implemented, with each assuming responsibility for distinct and specific functionalities. The MetricsticsApp class assumes the role of drawing and defining the interface for calculators, while the Statistics class is responsible for defining statistical functions. Lastly, the DataGenerator class fulfills the crucial task of generating data for the application.

To obtain the cyclomatic complexity for each class and its corresponding values, I utilized a tool known as **CodeMetrics**. This tool aids in analyzing the complexity of software code by calculating the cyclomatic number, which provides valuable insights into the code's structural intricacies and potential points of concern.

Here is the Cyclomatic number for each methods:

- **MetricsticsApp**:
  - main() - 1
  - start() - 1
- **DataGenerator**:
  - generate() - 4
- **Statistics**:
  - setData() - 1
  - clear() - 1
  - checkDataExists() - 6
  - getMin() - 5
  - getMax() - 5
  - getMode() - 11
  - getMedian() - 10
  - getArithmeticMean() - 4
  - getMeanAbsoluteDeviation() - 5
  - getStandardDeviation() - 8
  - abs() - 4
  - sqrt() - 26

  - calculateAndDisplay() - 10
  - calculateAll() - 1
- **UIGenerator**:
  - generator() - 11
  - start() - 3
- **Util**:
  - createOpertorButton() - 2
  - clearAll() - 1

[some screen short attached below]

4.2 Conclusions for each classes

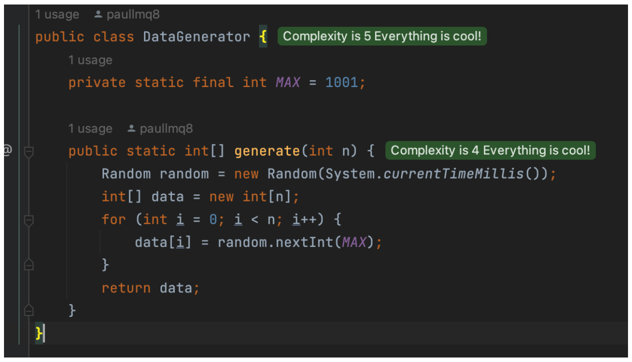
MetricsticsApp Class: MetricsticsApp is driver class of this application, the cyclomatic complexity of both methods in the MetricsApp class is low, with each method having a value of 1. In this case, both methods have a simple control flow with a single path, making them relatively easy to comprehend and maintain.

UIGenerator Class: The DataGenerator class exhibits slightly higher complexity than the MetricsApp class, with a cyclomatic number of 11 and 3. This difference can be attributed to the generate() method, which follows a recurring UI creation pattern utilizing various variables and parameters. To enhance code quality and reduce the cyclomatic complexity, it is advisable to refactor the method using the Factory Method pattern.

Statistics Class: The Statistics class has a highest Cyclomatic complexity among of others, indicating the presence of numerous decision points and complex control flow. For majority of the method in Statistics class, it is below 10, which indicate multiple decision points and branching paths. Howevevr, there is one method sqrt() holds 26 cyclomatic number, which need to refactor in order to reduce complexity and organizing control flow can make the code more readable and easier to maintain.

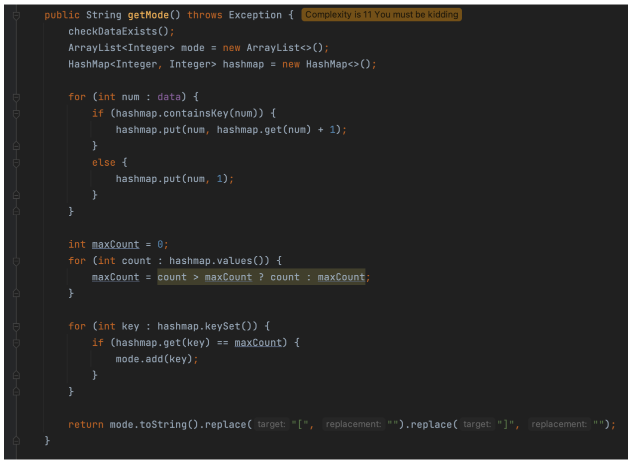
DataGenerator Class: The DataGenerator class has a moderate Cyclomatic complexity of 5, indicating a moderate level of control flow complexity. The class comprises only one method, generate(), which generates random numbers, and its Cyclomatic complexity is 4. Due to the straightforward nature of the class and method, they are likely to be easy to maintain and comprehend.

Util Class: The Util class exhibits a low Cyclomatic complexity of 2 and 1 for each of its methods. This suggests a relatively simple control flow, contributing to the class's ease of comprehension and maintainability. The class adheres to the principle of single responsibility by decomposing UI components into separate methods, further enhancing its clarity and structure.



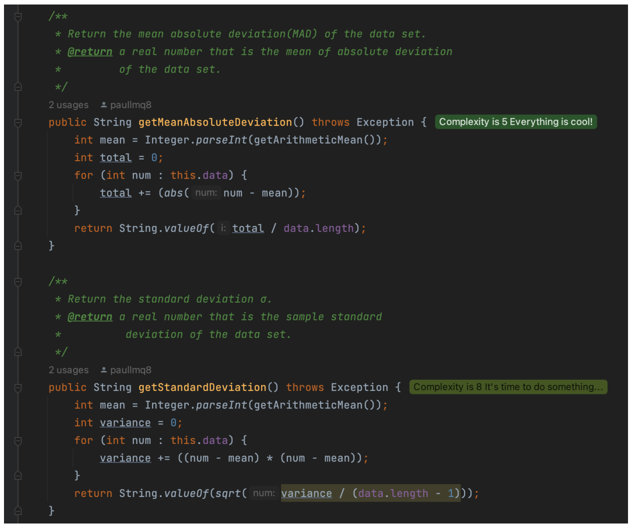
```
1 usage   paullmq8
public class DataGenerator {   Complexity is 5 Everything is cool!
    1 usage
    private static final int MAX = 1001;

    1 usage   paullmq8
    public static int[] generate(int n) {   Complexity is 4 Everything is cool!
        Random random = new Random(System.currentTimeMillis());
        int[] data = new int[n];
        for (int i = 0; i < n; i++) {
            data[i] = random.nextInt(MAX);
        }
        return data;
    }
}
```

Figure 4.1: Cyclomatic Number Code Sample



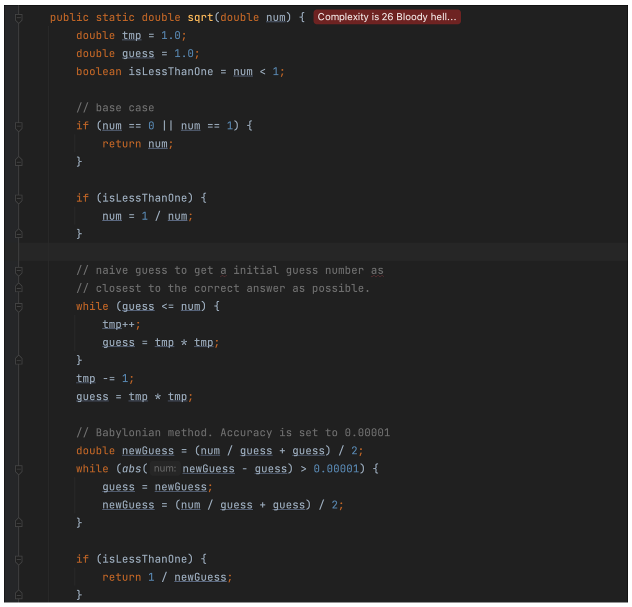
```
public String getMode() throws Exception {   Complexity is 11 You must be kidding
    checkDataExists();
    ArrayList<Integer> mode = new ArrayList<>();
    HashMap<Integer, Integer> hashmap = new HashMap<>();

    for (int num : data) {
        if (hashmap.containsKey(num)) {
            hashmap.put(num, hashmap.get(num) + 1);
        }
        else {
            hashmap.put(num, 1);
        }
    }

    int maxCount = 0;
    for (int count : hashmap.values()) {
        maxCount = count > maxCount ? count : maxCount;
    }

    for (int key : hashmap.keySet()) {
        if (hashmap.get(key) == maxCount) {
            mode.add(key);
        }
    }

    return mode.toString().replace( target: "[",  replacement: "").replace( target: "]",  replacement: "");
}
```

Figure 4.2: Cyclomatic Number Code Sample



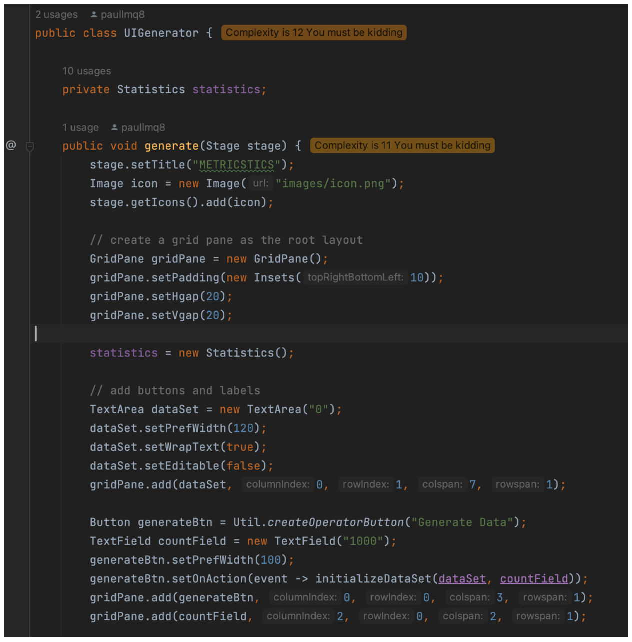
```
/**
 * Return the mean absolute deviation(MAD) of the data set.
 * @return a real number that is the mean of absolute deviation
 *          of the data set.
 */
2 usages   paullmq8
public String getMeanAbsoluteDeviation() throws Exception {   Complexity is 5 Everything is cool!
    int mean = Integer.parseInt(getArithmeticMean());
    int total = 0;
    for (int num : this.data) {
        total += (abs( num: num - mean));
    }
    return String.valueOf( i: total / data.length);
}

/**
 * Return the standard deviation σ.
 * @return a real number that is the sample standard
 *          deviation of the data set.
 */
2 usages   paullmq8
public String getStandardDeviation() throws Exception {   Complexity is 8 It's time to do something...
    int mean = Integer.parseInt(getArithmeticMean());
    int variance = 0;
    for (int num : this.data) {
        variance += ((num - mean) * (num - mean));
    }
    return String.valueOf(sqrt( num: variance / (data.length - 1)));
}
```

Figure 4.3: Cyclomatic Number Code Sample



```
public static double sqrt(double num) {  Complexity is 26 Bloody hell...
    double tmp = 1.0;
    double guess = 1.0;
    boolean isLessThanOne = num < 1;

    // base case
    if (num == 0 || num == 1) {
        return num;
    }

    if (isLessThanOne) {
        num = 1 / num;
    }

    // naive guess to get a initial guess number as
    // closest to the correct answer as possible.
    while (guess <= num) {
        tmp++;
        guess = tmp * tmp;
    }
    tmp -= 1;
    guess = tmp * tmp;

    // Babylonian method. Accuracy is set to 0.00001
    double newGuess = (num / guess + guess) / 2;
    while (abs( num: newGuess - guess) > 0.00001) {
        guess = newGuess;
        newGuess = (num / guess + guess) / 2;
    }

    if (isLessThanOne) {
        return 1 / newGuess;
    }
```

Figure 4.4: Cyclomatic Number Code Sample



```
2 usages  ≗ paullmq8
public class UIGenerator {  Complexity is 12 You must be kidding

    10 usages
    private Statistics statistics;

    1 usage  ≗ paullmq8
    public void generate(Stage stage) {  Complexity is 11 You must be kidding
        stage.setTitle("METRICSTICS");
        Image icon = new Image( url: "images/icon.png");
        stage.getIcons().add(icon);

        // create a grid pane as the root layout
        GridPane gridPane = new GridPane();
        gridPane.setPadding(new Insets( topRightBottomLeft: 10));
        gridPane.setHgap(20);
        gridPane.setVgap(20);

        statistics = new Statistics();

        // add buttons and labels
        TextArea dataSet = new TextArea("0");
        dataSet.setPrefWidth(120);
        dataSet.setWrapText(true);
        dataSet.setEditable(false);
        gridPane.add(dataSet,  columnIndex: 0,  rowIndex: 1,  colspan: 7,  rowspan: 1);

        Button generateBtn = Util.createOperatorButton("Generate Data");
        TextField countField = new TextField("1000");
        generateBtn.setPrefWidth(100);
        generateBtn.setOnAction(event -> initializeDataSet(dataSet, countField));
        gridPane.add(generateBtn,  columnIndex: 0,  rowIndex: 0,  colspan: 3,  rowspan: 1);
        gridPane.add(countField,  columnIndex: 2,  rowIndex: 0,  colspan: 2,  rowspan: 1);
```

Figure 4.5: Cyclomatic Number Code Sample

5 Problem 6 : WMC, CF, LCOM

5.1 WMC

The formula of WMC is to obtain the sum of weighted method per class. In this senicaio(Non-Normalization case), we will use we have cyclomatic numbers above to calculate the WMC

- WMC(MetricsticsApp) — 1 + 1 = 2
- WMC(DataGenerator) — 4
- WMC(Statistics) — 6 + 5 + 5 + 11 + 10 + 4 + 5 + 8 + 4 + 26 = 84
- WMC(UIGenrator) — 11 + 3 = 14
- WMC(Util) — 2 + 1 = 3

MetricsticsApp: It has a relatively low WMC of 2, which suggests that the class has a simple structure with only two methods, making it easy to understand and maintain

DataGenerator: With a WMC of 4, the DataGenerator class has a moderate complexity, indicating that it contains a few methods, but they are not overly complex. It should still be relatively straightforward to manage.

Statistics: The Statistics class has a high WMC of 84, signifying significant complexity due to a large number of methods and their complexities. This class may require careful attention during development and maintenance to ensure its correctness and readability.

UIGenerator: The UIGenerator class has a moderate WMC of 14, indicating that it is not excessively complex but still requires attention to maintain and understand the class.

Util: With a WMC of 3, the Util class has relatively low complexity. It contains a few methods, making it straightforward to handle and likely follows the principle of single responsibility

In summary, the complexity levels vary among the classes. While some classes have low complexity and are easy to maintain, others exhibit higher complexity, necessitating careful review and potential refactoring to improve code quality and readability. Developers should focus on managing the complexity of high WMC classes, especially the Statistics class with its WMC of 84, to ensure that it remains maintainable and comprehensible over time.

5.2 CF

By Figure 5.1, n = 6, (0+0+1+1+1+1)/(36 - 6) = 2/15 = 0.133333333333

the low coupling factor suggests that the software system's design encourages a well-structured and loosely connected architecture, which is beneficial for software development and maintenance, A coupling factor of 0.133333333333 signifies that the components or modules within the software system are loosely connected, with minimal dependencies between them

5.3 LOCM

LCOM(MetricsticsApp): (Figure 5.2) m = 2, a = 2

- μ (Stage) = 1
- μ (Application) = 1

LCOM(MetricsticsApp) = [(1/2) * (1+1) - 2)] / 1 - 2 = 1

LCOM(UIGenerator): (Figure 5.3) m = 2, a = 9

- μ (Stage) = 1
- μ (Image) = 1
- μ (GridPane) = 1
- μ (Textarea) = 2
- μ (Button) = 1
- μ (Label) = 1
- μ (Scene) = 1
- μ (Alert) = 1
- μ (Statistics) = 1

LCOM(Statistics) = [(1/9) * (1+1+1+2+1+1+1+1+1) - 2] / (1-2) = 8/9 = 0.888889

LCOM(DataGenerator): (Figure 5.4) m = 1, a = 3

- μ (Random) = 1
- μ (Max) = 1
- μ (System) = 1

LCOM(DataGenerator) = [(1/3) * (1+1+1) - 1] / (1 - 1) = 0

LCOM(Statistics): (Figure 5.5) m = 12, a = 4

- μ (data) = 8
- μ (number) = 2
- μ (Alert) = 2
- μ (Label) = 1

LCOM(Statistics) = [(1/4) * (8+2+2+1) - 12] / (1 - 12) = (35/4)/11 = 0.795454545455

LCOM(Util): (Figure 5.6) m = 2, a = 2

- μ (Button) = 1
- μ (Label) = 1

LCOM(Util) = [(1/2) * (1+1) - 1] / (1 - 2) = 1

Conclusion

There 5 different LOCM value, 1, 0.888889, 0, 0.795454545455, 1

- MetricsticsApp has an LCOM of 1, indicating good cohesion, meaning its methods are well-related and work together effectively.
- UI has an LCOM of 0.888889, which is close to 1, suggesting reasonable cohesion, but some methods might not be as closely related to others.
- DataGenerator has an LCOM of 0, meaning the methods in the class are highly cohesive, and they likely work closely together.
- Statistics has an LCOM of 0.795454545455, which is relatively close to 1, suggesting decent cohesion but with some room for improvement.
- Util has an LCOM of 1, indicating good cohesion, implying its methods are well-organized and work together effectively.

In summary, most of the classes exhibit reasonable to good cohesion based on their LCOM values. However, the Statistics class might benefit from a bit more organization to improve its cohesion further.

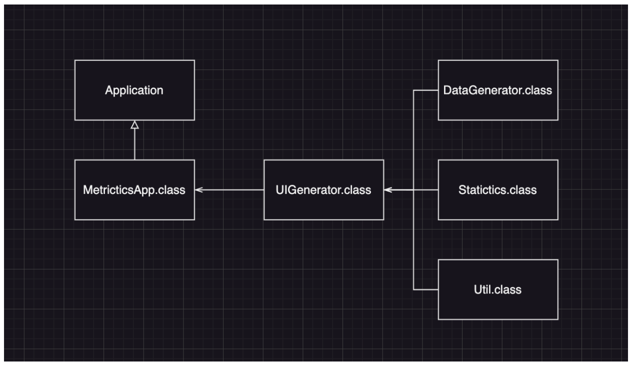


Figure 5.1: CF

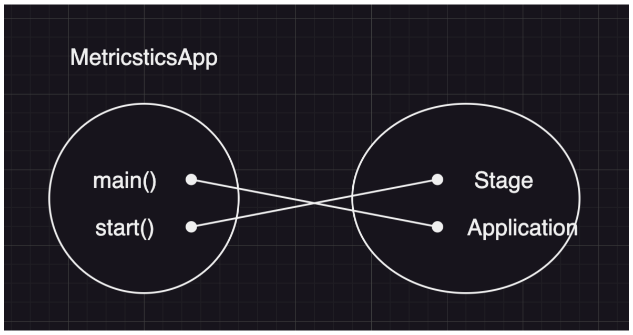


Figure 5.2: LCOM MetricsticsApp

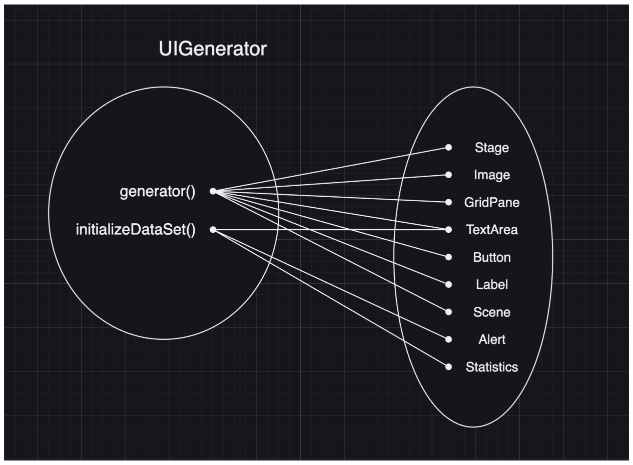


Figure 5.3: LCOM UIGenerator

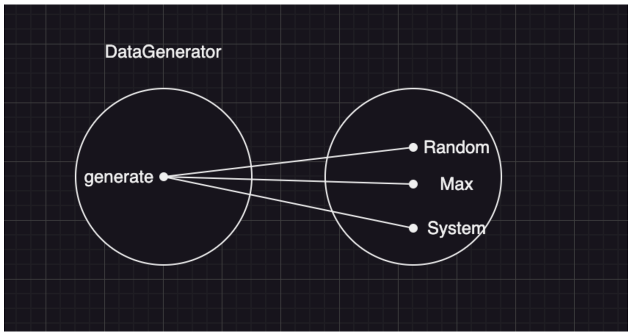


Figure 5.4: LCOM DataGenerator

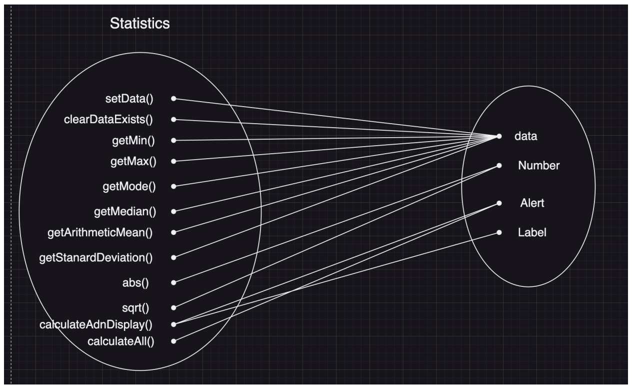


Figure 5.5: LCOM Statistics

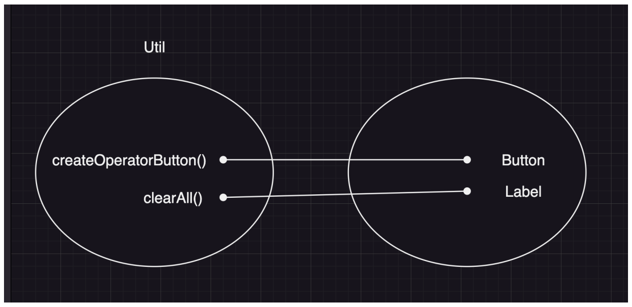


Figure 5.6: LCOM Util

6 Problem 7 : Physical And Logical SLOC

6.1 Goal

Calculating the Physical SLOC and Logical SLOC for METRICSTICS. For Physical SLOC a tool named CLOC is used and for Logical SLOC a manual counting approach was applied.

6.2 Physical SLOC

refers to the number of lines in the source code of a software project that contain executable statements, meaning lines that have actual code instructions and are not blank lines or comments.[Chatgpt]

Tool-CLOC: CLOC is used to count blank lines, comment lines, and physical lines of source code in many programming languages. This project uses the CLOC tool to calculate the physical SLOC.

Installing CLOC: CLOC documentation provides numerous way to install the tool. Installing via package manager can vary depending on the operating system. "npm install -g cloc " can be used to globally install the tool on your terminal.

Using CLOC: CLOC provides a very user friendly and simple command line tool. To calculate the physical SLOC the main directory was selected and using the command "cloc main" provides the user with a report.

Advantages of Physical SLOC: Physical SLOC is language-independent. SLOC also offer insights into code size and complexity.

Inference: 336 Lines of code indicates a smaller size system. The SLOC value of this project suggests less complexity and due to its small size it would be easier to maintain.

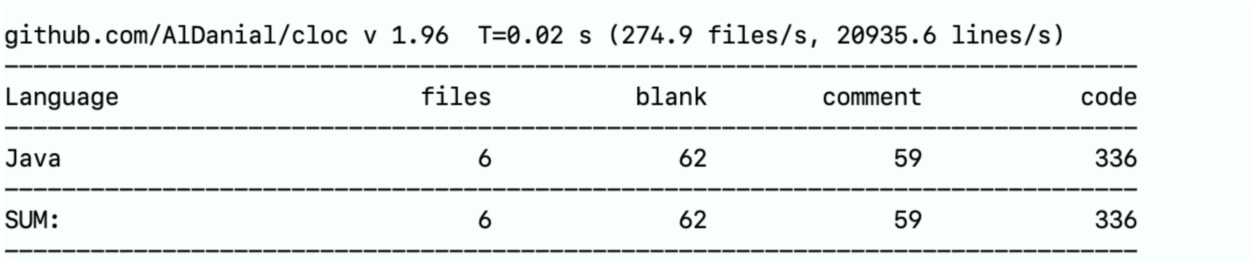

```
github.com/AlDanial/cloc v 1.96  T=0.02 s (274.9 files/s, 20935.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                             6             62             59            336
-------------------------------------------------------------------------------
SUM:                             6             62             59            336
-------------------------------------------------------------------------------
```

Figure 6.1: Physical SLOC

6.3 Logical SLOC

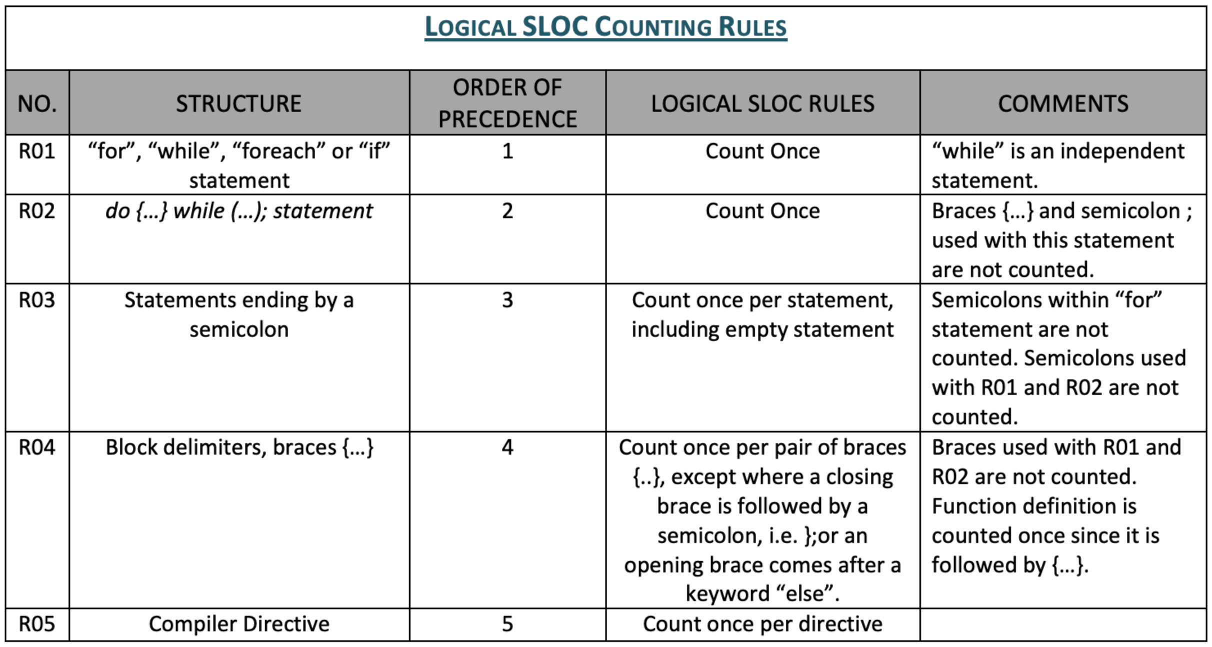
It is a measure of the size or complexity of a software project based on the number of logical statements or instructions in the source code that affect the program's control flow or behavior

Manual Method: The project's logical SLOC is calculated manually. following the UCC (Unified Code Counter) Logical SLOC Counting Rules for Java.The rules are shown in Figure 4.2 and 4.3. Total SLOC(L) is a summation of SLOC(L) of all classes present in the system. The total Logical SLOC is 83

- Total logical SLOC(MetricssticsApp): 9
  - Package declaration: 1 SLOC
  - Import statements: 3 SLOCs
  - Class declaration (MetricssticsApp): 1 SLOC
  - main method: 1 SLOC
  - launch method call: 1 SLOC
  - start method: 1 SLOC
  - Instantiation of UIGenerator class: 1 SLOC
- Total logical SLOC(DataGenerator): 6
  - Import statements: 2 SLOCs
  - Class declaration (DataGenerator): 1 SLOC
  - Constant declaration (MAX): 1 SLOC
  - Method declaration (generate): 1 SLOC
  - For loop: 1 SLOC
- Total logical SLOC(Statistics): 41
  - Package declaration: 1 SLOC

  - Import statements: 7 SLOCs
  - Class declaration (Statistics): 1 SLOC
  - Setter method (setData): 1 SLOC
  - Method (clear): 1 SLOC
  - Method (checkDataExists): 1 SLOC (method declaration)
  - If Statement: 1 SLOC
  - Method (getMin): 1 SLOC (method declaration)
  - For loop: 1 SLOC
  - Method (getMax): 1 SLOC (method declaration)
  - For loop: 1 SLOC
  - Method (getMode): 1 SLOC (method declaration)
  - For loop: 1 SLOC
  - For loop: 1 SLOC
  - Method (getMedian): 1 SLOC
  - If Statement: 1 SLOC
  - Method (getArithmeticMean): 1 SLOC
  - For Loop: 1 SLOC
  - Method (getMeanAbsoluteDeviation): 1 SLOC
  - For Loop: 1 SLOC
  - Method (getStandardDeviation): 1 SLOC
  - For Loop: 1 SLOC
  - Static method (abs): 1 SLOC
  - Static method (sqrt): 1 SLOC
  - If Statement: 2 SLOC
  - While Loop: 1 SLOC
  - Assigning Value: 2 SLOC
  - While Loop: 1 SLOC
  - If Statement: 1 SLOC
  - Method (calculateAndDisplay): 1 SLOC
  - Try-Catch: 1 SLOC
  - Method (calculateAll): 1 SLOC
  - Try-Catch: 1 SLOC

- Total logical SLOC (UIGenerator): 19



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	

Figure 6.2: Logical SLOC Counting Rule Table

  - Package declaration: 1 SLOC
  - Import statements: 14 SLOCs
  - Class declaration (UIGenerator): 1 SLOC
  - Method (generate): 1 SLOC
  - Method (initializeDataSet): 1 SLOC
  - Try-Catch: 1 SLOC

- Package declaration: 1 SLOC
  - Import statements: 14 SLOCs
  - Class declaration (UIGenerator): 1 SLOC
  - Method (generate): 1 SLOC
  - Method (initializeDataSet): 1 SLOC
  - Try-Catch: 1 SLOC
  - Total logical SLOC: 19

Advantages of Logical SLOC: Measures essential code which indicates functionality, maintainability, and performance of the code.

Inference: In conclusion, a logical SLOC of 83 suggests that the software project is small in comparison to generic systems and may have a simpler and more manageable source code. Logical SLOC however does not provide the entire picture and so to get a more comprehensive understanding of the project, it's essential to consider other metrics and factors as well.

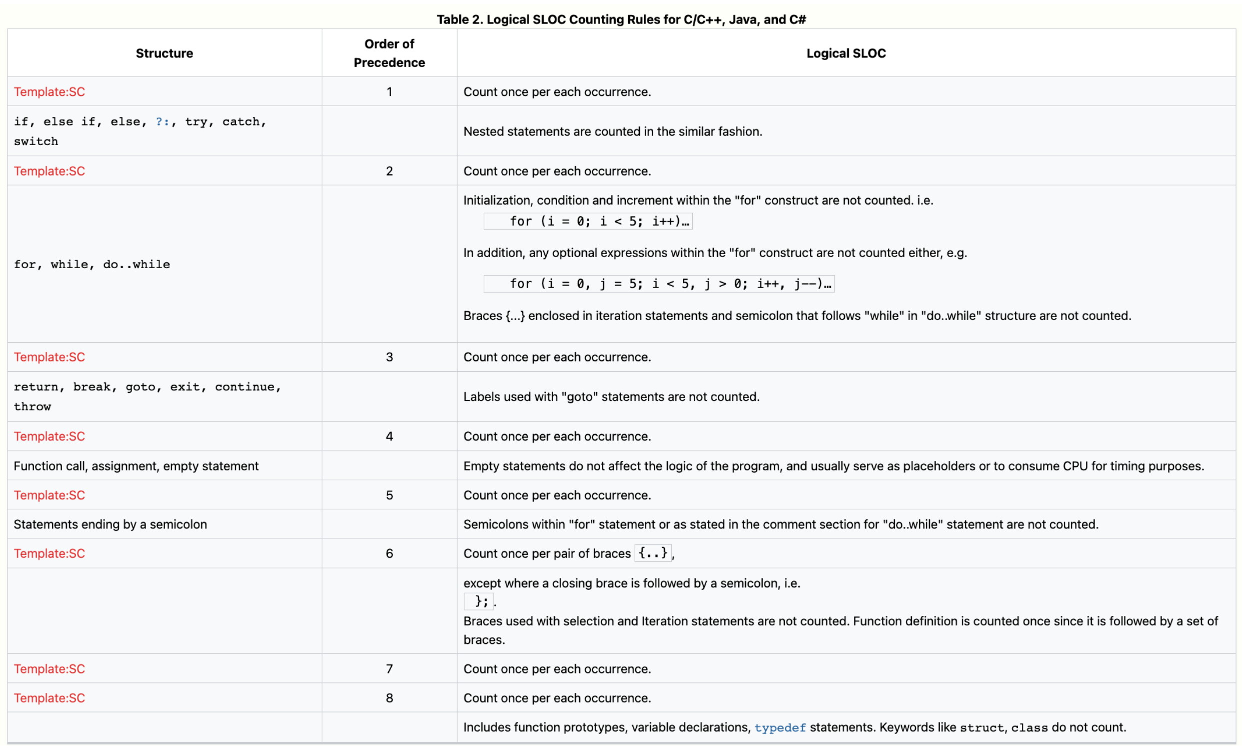


Table 2. Logical SLOC Counting Rules for C/C++, Java, and C#

Structure	Order of Precedence	Logical SLOC
Template:SC	1	Count once per each occurrence.
`if, else if, else, ?:, try, catch, switch`		Nested statements are counted in the similar fashion.
Template:SC	2	Count once per each occurrence.
`for, while, do..while`		Initialization, condition and increment within the "for" construct are not counted. i.e. `for (i = 0; i < 5; i++)…` In addition, any optional expressions within the "for" construct are not counted either, e.g. `for (i = 0, j = 5; i < 5, j > 0; i++, j--)…` Braces {…} enclosed in iteration statements and semicolon that follows "while" in "do..while" structure are not counted.
Template:SC	3	Count once per each occurrence.
`return, break, goto, exit, continue, throw`		Labels used with "goto" statements are not counted.
Template:SC	4	Count once per each occurrence.
Function call, assignment, empty statement		Empty statements do not affect the logic of the program, and usually serve as placeholders or to consume CPU for timing purposes.
Template:SC	5	Count once per each occurrence.
Statements ending by a semicolon		Semicolons within "for" statement or as stated in the comment section for "do..while" statement are not counted.
Template:SC	6	Count once per pair of braces `{..}`,
		except where a closing brace is followed by a semicolon, i.e. `};`. Braces used with selection and Iteration statements are not counted. Function definition is counted once since it is followed by a set of braces.
Template:SC	7	Count once per each occurrence.
Template:SC	8	Count once per each occurrence.
		Includes function prototypes, variable declarations, `typedef` statements. Keywords like `struct`, `class` do not count.

Figure 6.3: Logical SLOC Counting Rule Table

7 Problem 8 : Establishing Relation between Logical SLOC and WMC Obtained from METRICSTICS

7.1 Scatter Plot

Figure 5.1 shows the scatter plot between the data for Logical SLOC and WMC obtained from METRICSTICS. By visualizing the data points on a scatter plot, we can observe patterns and trends.
WMC stands for "Weighted Methods per Class." It is a metric used to measure the complexity of a class and also provides an indication of how many methods within a class are present and how complex those methods are.
Logical SLOC is a measure of the size or complexity of a software project based on the number of logical statements or instructions in the source code that affect the program's control flow or behavior Plotting these values on scatter plot provides us with an insight on the affect of SLOC on WMC for a given system.

Class	**WMC**	**SLOC (L)**
MetricsticsApp	0	9
Util	3	8
DataGenerator	4	6
UIGenerator	14	19
Statistics	84	41

Table 7.1: Class Metrics

7.2 correlation coefficient

In this section we find the correlation coefficient between WMC and SLOC(L) . By plotting the WMC values and SLOC(L) values separately on a histogram, we can conclude that these values are not normally distributed .

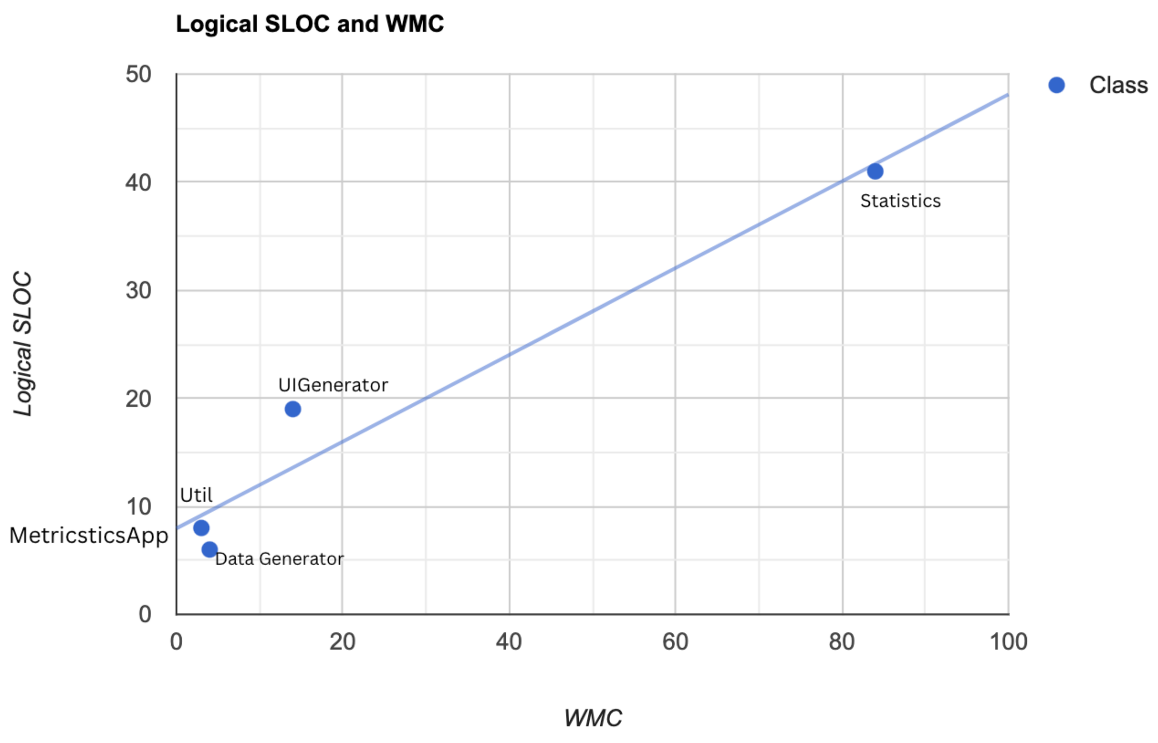


Figure 7.1: Scatter Plot (Logical SLOC and WMC

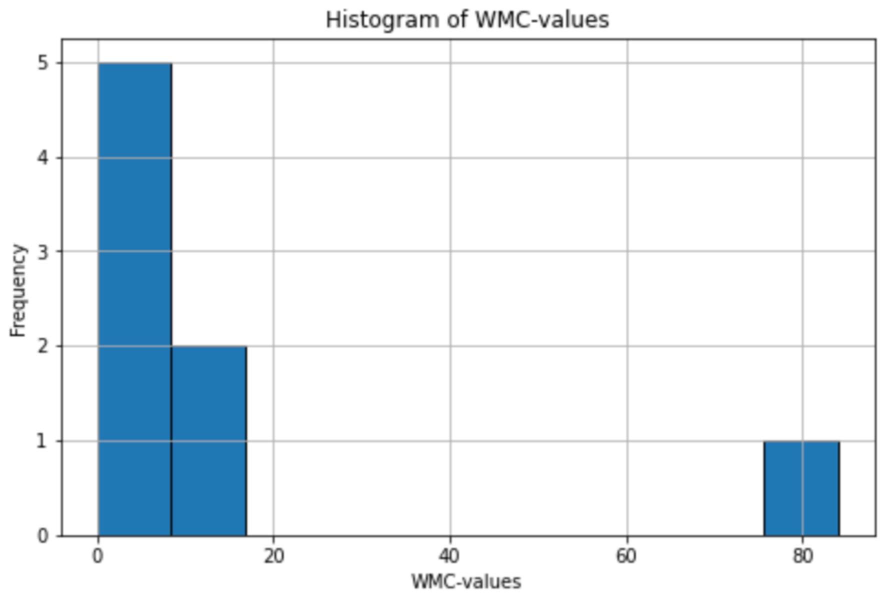
Since the values of x's [WMC] and y's are non-normally distributed as shown in the figure[5.2], The Spearman's Rank Correlation Coefficient (rs) can be used to find the correlation coefficient . The Spearman's Rank Correlation Coefficient (rs) is a measure of association for attributes values that are not distributed normally [Lecture Slides].
Let the data in each set of WMC(x) and SLOC(L)(y) be ranked separately in ascending order by rank (xi) and rank (yi).
n is the number of pairs of (x,y)
di = rank (xi) – rank (yi).
Then, rs is given by

$$r_s = 1 - \frac{6\sum_{i=n}^{n} d_i^2}{n^3 - n}$$

The value of n is 5 for this particular system. All calculations are done based on the values from table 5.2

Spearman's Rank Correlation Coefficient can range from -1 to 1. The closer the value is to 1 the stronger the correlation it will have. For the system of METRICSTICS the coefficient value is 0.6 which shows a strong correlation between WMC and SLOC(L).

The value of coefficient suggest in case the logical SLOC increases there is a high chance WMC will increase too. But conclusively we cannot say only SLOC has an impact on WMC



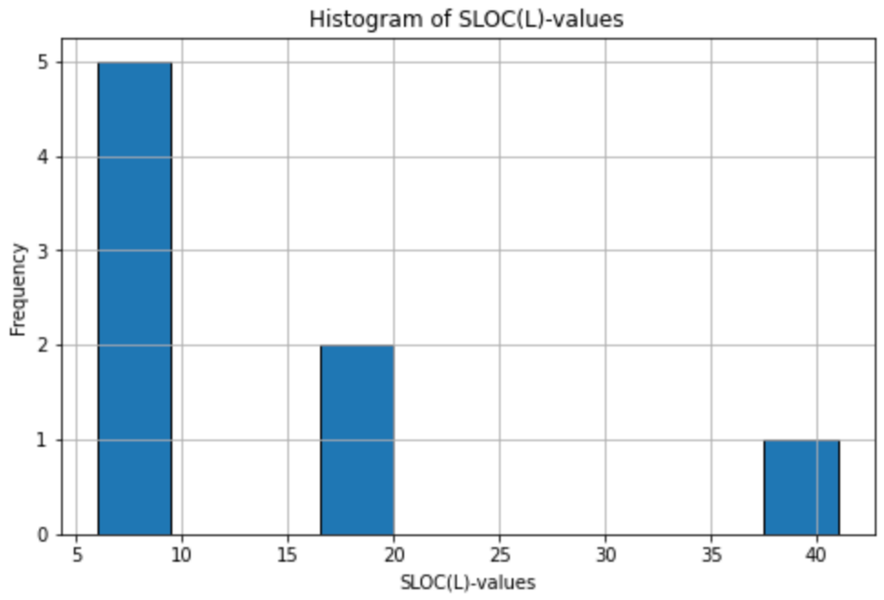


Figure 7.2: Scatter Plot (Logical SLOC and WMC

or vice versa but many other underlying factors may also have an affect on these values. It is important to take in account all metrics of software to make justification of change.

$WMC(x_i)$	Rank(x_i)	SLOC(L)(y_i)	Rank(y_i)	d	d^2
0	1	9	3	-2	4
3	2	8	2	0	0
4	3	6	1	2	4
14	4	19	4	0	0
84	5	41	5	0	0

Table 7.2: Data Table

References

Lecture Slides *Lecture slides"SOEN6611 Course Website"*.
ChatGpt *ChatGpt*.
Metrics.,
    https://www.geeksforgeeks.org/software-measurement-and-metrics/
Use Case Diagram.,
    https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what is use case diagram/
Physical SLOC,
    https://github.com/AlDanial/cloc
Logical SLOC,
    https://handwiki.org/wiki/Software:Unified_Code_Count_(UCC)
JavaFX,
    https://en.wikipedia.org/wiki/JavaFX

Github Repository
https://github.com/hemareddy123/SOEN_6611_Summer2023/tree/main