Project Deliverable 2

Software Measurement (SOEN 6611)

METRICSTICS

by
Anagha Harinath (40224578)
Madiha Itrat (40216270)
Mohammed Huzaifa (40242080)
Sameer Kamble (40204785)
Srikar Hasthi (40230004)

Under the Supervision of
Professor: Pankaj Kamthan
TA: Iymen Abdella

Department of Computer Science and Software Engineering

November 22, 2023

Contents

1 GQM for Descriptive Statistics

Using the **Goal-Question-Metric (GQM)** approach (or one of its extensions), present one goal specific to METRICSTICS and articulate 2N questions related to that goal, where N is the team size. Discuss whether any metrics help answer those questions.
Note. The goals must aim to be SMART.

1.1 Goal

Following the template given in the lecture notes [1]:

Purpose

To *develop within the given time limit from 26th September 2023 to 24th November 2023*, the *artifacts of a comprehensive system capable of taking a random set of data values as input* and generating the following descriptive statistics:

1. Minimum (m),
2. Maximum (M),
3. Mode (o),
4. Median (d),
5. Mean absolute deviation (MAD), and
6. Standard deviation (σ).

in order to *provide a numerical description of an arbitrary set of data values.*

Perspective

Examine *the accuracy of Descriptive Statistics from a technical user's (researchers, analysts, and data scientists) perspective.*

Environment

In the context of the *development and testing phase of a project's lifecycle.*

1.2 The goal in terms of a SMART Goal:

Specific

By the end of the project duration, we would have developed the METRICSTICS Descriptive Statistics System, which will accurately calculate key statistical measures (minimum, maximum, mode, median, mean absolute deviation, and standard deviation) for input datasets, both real and artificially generated, through an intuitive user interface.

Measurable

We will measure the success of this goal by using metrics mentioned in 1.3

Achievable

To achieve this goal, we will allocate adequate resources, including a dedicated development team, a testing resource, and project management oversight.

Relevant

Developing the METRICSTICS Descriptive Statistics System is directly aligned with our team's mission to address the growing need for efficient and accurate descriptive statistical analysis, essential for technical users such as researchers, analysts, and data scientists across various industries.

Time-Bound

We will complete the development, testing, release, and documentation of METRICSTICS within the next 3 months. This timeline allows for comprehensive development, testing, refinement, and documentation while ensuring timely delivery to meet user needs.

By adhering to this SMART goal, we aim to deliver a valuable tool that enhances the capabilities of technical users to perform descriptive statistical analysis quickly, accurately, and efficiently.

1.3 Questions and Metrics

1. Is it possible to complete the project within the allotted time limit?
   - M1: Task completion rate.
   - M2: Burn down chart.
   - M3: Leadtime [2] (time required to go from idea to delivered software)
2. Does the existing functionality align with the expectations of a technical user?
   - M4: Technical User's Heuristic Assessment
3. What are the program's dimensions?
   - M6: Storage capacity needed to run the software application.
   - M7: Lines of Code (LOC)
4. Is the code efficient and well-optimized?
   - M8: Number of duplicated Lines of Code
   - M9: The computational time complexity of functions
5. Is the program easy to maintain?
   - M10: Documentation [3]
   - M11: Maintainability Index [4]
6. How efficient is the team's performance?
   - M12: Velocity
   - M13: Average Cycle Time [5]
7. What is the level of satisfaction among the project's stakeholders?
   - M14: Team Satisfaction
   - M15: Customer Satisfaction
8. Do all the functions produce the expected output?
   - M16: Unit test code coverage
9. Are there any defects during the development?
   - M17: Bugs and defect count
10. Is the project equipped to handle errors and invalid inputs during its execution?
    - M18: Robustness Index [6]
    - M19: Function robustness [6]
11. What are the user stories' priority?
    - M20: User Story Point
12. Is the code designed to be independent of any specific IDE (Integrated Development Environment)?
    - M21: Portability [7]

2 Use Case Model

Using the given description, construct a use case model for DESCRIPTIVE STATISTICS.
Note. There can be several use cases, including saving data in memory, restarting a session, and so on. A statistical calculator could be used as a motivation to 'elicit' necessary use cases.

2.1 Use Case Model Diagram

Following the use case templates [8] and use case diagram example [9] we get the following:

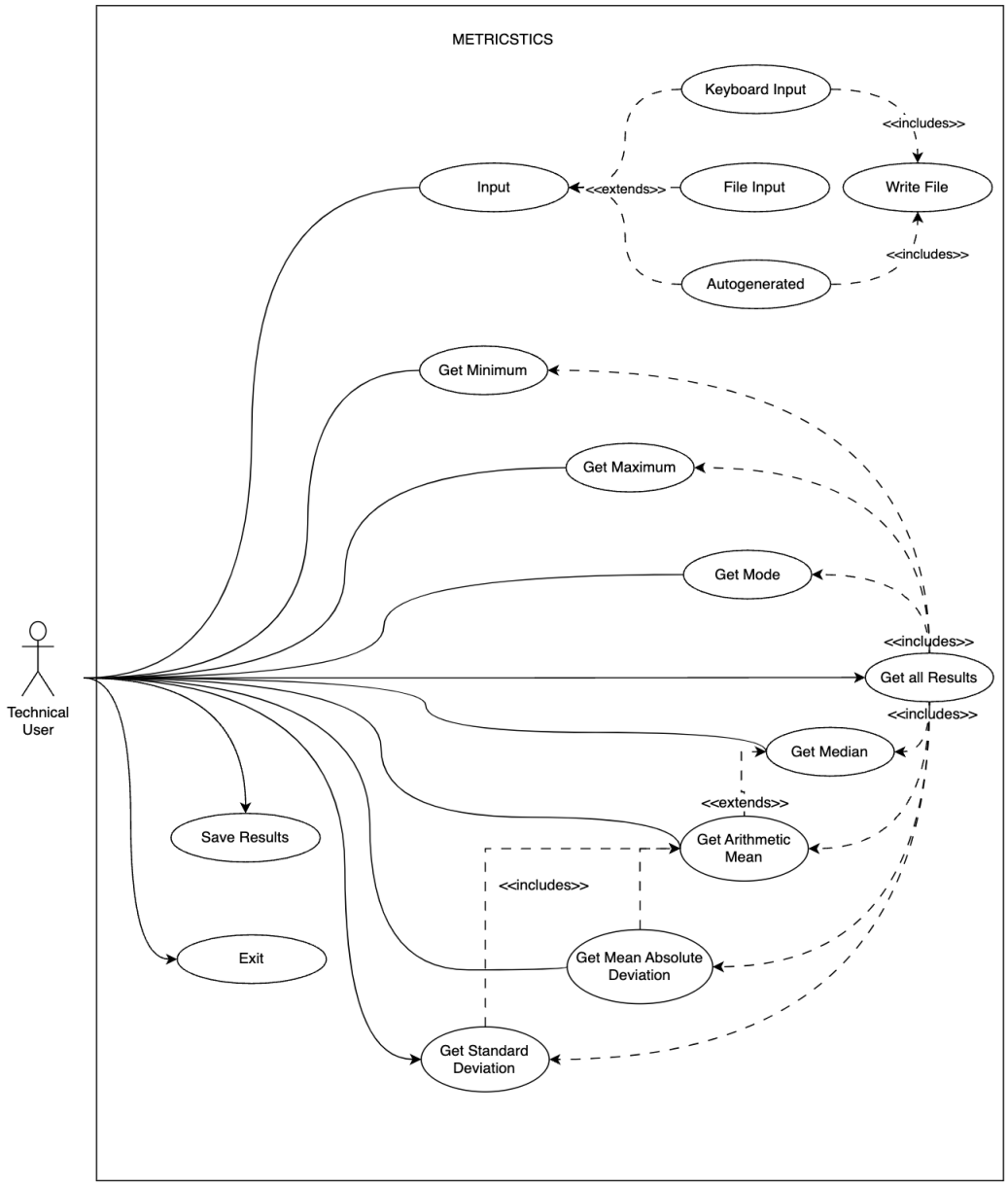


Figure 1: Use case model diagram for METRICSTICS

Actors

The actor is a technical user who can be a researchers, analysts, student or data scientists. They will use the system to obtain descriptive statistics for research, building models or assignment.

2.2 Use Case Model Tables

Use Case 1

Title	**Input data set**
Description	Provide a finite data set x_1, x_2, x_3, ... x_n as input from one of the below two sources: 1. Keyboard, 2. File and 3. Autogenerated.
Primary Actor	Technical User
Preconditions	1. The program must be running 2. The input source must be in proper comma separated values format 3. The input should be numbers 4. There should be at least 1 value in the input file
Success Guarantee	Data set is assigned to a variable
Main Success Scenario	1. User inputs a valid dataset 2. The system sorts the data in an ascending order
Extensions	An error message is showed indicating the data set given was invalid or in the wrong format.

Use Case 2

Title	**Get Minimum**
Description	Calculate the minimum value from the given data set.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The minimum value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system identifies the smallest value in the data set as the minimum. 3. The system displays the calculated minimum value. 4. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without minimum value calculation.

Use Case 3

Title	**Get Maximum**
Description	Calculate the maximum value from the given data set.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The maximum value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system identifies the largest value in the data set as the maximum. 3. The system displays the calculated maximum value. 4. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without maximum value calculation.

Use Case 4

Title	**Get Mode Value**
Description	Calculate the mode value(s) from the given data set which is the most frequently appeared value in the data set (mode value does not have to be unique).
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The mode value(s) is/are successfully displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system analyzes the data set to identify the value(s) that appears most frequently. 3. The identified value(s) is designated as the mode value(s). 4. The system displays the calculated mode value(s). 5. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without mode value(s) calculation.

Use Case 5

Title	**Get Median Value**
Description	Calculate the median value from the given data set. The median is the middle number if the number of data values is odd, and it is the arithmetic mean of the two middle numbers if number of data values is even.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The median value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system checks the number of data values (n) in the dataset. 3. If n is odd: The system identifies the middle value as the median. If n is even: The system calculates the arithmetic mean of the two middle values, which is designated as the median. 4. The system displays the calculated median value. 5. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without median value calculation.

Use Case 6

Title	**Get Arithmetic Mean Value**
Description	Calculate the arithmetic mean value from the given data set. $$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i.$$
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The Arithmetic Mean value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system calculates the arithmetic mean by summing all the values in the data set and dividing by the number of data values. 3. The system displays the calculated arithmetic mean value. 4. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without arithmetic mean value calculation.

Use Case 7

Title	**Get Mean Absolute Deviation Value**
Description	Calculate the mean absolute deviation (MAD) from the given data set. MAD measures the average absolute difference between each data value and the arithmetic mean. $$\mathrm{MAD} = \frac{1}{n}\sum_{i=1}^{n} \lvert x_i - \mu \rvert .$$
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The Mean Absolute Deviation value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system calculates the mean absolute deviation (MAD) using the formula. 3. The system displays the calculated mean absolute deviation value. 4. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is displayed, and the user returns to the main menu without Mean Absolute Deviation value calculation.

Use Case 8

Title	**Get Standard Deviation**
Description	Calculate the standard deviation from the given data set. The standard deviation measures the dispersion or spread of data values around the arithmetic mean. $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The Standard Deviation value is successfully calculated and displayed.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system calculates the standard deviation by using the formula. 3. The system displays the calculated standard deviation value. 4. The user returns to the main menu of operations.
Extensions	If the data set is empty, an error message indicating that the input is empty is, and the user returns to the main menu without Standard Deviation value calculation.

Use Case 9

Title	**Get All Results**
Description	Display all calculated descriptive statistics, i.e., minimum, maximum, mode, median, arithmetic mean, mean absolute deviation, and standard deviation for the given data set.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. A valid data set must have been inputted into the system.
Success Guarantee	The calculated descriptive statistics are successfully computed and displayed to the Technical User.
Main Success Scenario	1. The system retrieves the sorted data set. 2. The system computes and displays the following descriptive statistics to the user: • Minimum • Maximum • Mode • Median • Arithmetic Mean • Mean Absolute Deviation • Standard Deviation 3. The system displays the calculated standard deviation value. 4. The user returns to the main menu of operations.
Extensions	If no valid descriptive statistics are available due to an empty data set or invalid result values, an error message indicating the same is, and the user returns to the main menu without displaying any statistics.

Use Case 10

Title	**Save Session**
Description	Save the input data reference and descriptive statistics to a session file if the user selects the 'save session' option.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. Input file exists.
Success Guarantee	The input data reference and descriptive statistics are saved in the session file.
Main Success Scenario	1. The system prompts the user to provide a session name. 2. The system adds an entry into the session file containing the session name, input data reference and descriptive statistics. 3. The system returns to the main menu.
Extensions	If the session name already exists, an error message indicating that a similar session name already exists is displayed, and the user returns to the main menu without saving the session.

Use Case 11

Title	**Load Session**
Description	Load the input data and descriptive statistics based on the session name selected by the user if the user selects the 'load session' option.
Primary Actor	Technical User
Preconditions	1. The system must be running. 2. Session file and input data file exists.
Success Guarantee	The input data and calculated descriptive statistics are loaded in the system based on user selected session name.
Main Success Scenario	1. The system displays a list of available session names. 2. The user selects the session name they want to load. 3. The system loads the input data along with the descriptive statistics. 4. The system returns to the main menu.
Extensions	None

Use Case 12

Title	**Exit**
Description	Exit the program if the user selects the exit option.
Primary Actor	Technical User
Preconditions	1. The system must be running.
Success Guarantee	The program terminates successfully.
Main Success Scenario	1. The program must end all its processes. 2. The user is no longer able to interact with the program.
Extensions	None

3 UCP and COCOMO 81

(a) Using the use case points (UCP) approach (or one of its extensions), provide an estimate of the effort towards the project.
(b) Using Basic COCOMO 81, provide an estimate of the effort towards the project.
(c) Comment on the difference in estimates using the UCP approach and COCOMO 81, and the actual effort towards the project.
Note: It should b e understood that an estimation should be dependable, but, at the same time, an estimation is not an exactimation.

3.1 Use Case Points (UCP)

To get an estimate for the effort using the UCP approach, like mentioned in lecture [10], we are going to need the following 3 variables:

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

Each of these variables are calculated separately with different equations (which will be described afterwards).

Below is an overview of the steps in the effort estimation based on the UCP approach:

1. Determine and Calculate UUCP.
2. Determine and Calculate TCF.
3. Determine and Calculate ECF.
4. Determine Productivity Factor (PF).
5. Calculate UCP.
6. Calculate the Estimated Number of Person-Hours.

3.1.1 Calculate UUCP

To calculate the UUCP we will need the sum of the **Unadjusted Actor Weight (UAW)** and the **Unadjusted Use Case Weight (UUCW)**.

$$UUCP = UAW + UUCW \ (4)$$

To get the UAW, we have the following table with the type and weight corresponding to each actor:

Actor	Actor Type	Weight
Metricstics System	Simple Actor	1
Technical User	Average Actor	2

Table 13: (UAW) Classification of actors and their weights

After adding all the actor weights we get the following result for our UAW:

$$UAW = (1 * 1) + (1 * 2) = 3$$

To get the UUCW, we have the following table which displays the Use Cases, the use case types and their corresponding weights:

Use Case	Use Case Type	Weight
Input data set	Average Use Case	10
Get Minimum	Simple Use Case	5
Get Maximum	Simple Use Case	5
Get Mode value	Simple Use Case	5
Get Median value	Simple Use Case	5
Get Arithmetic Mean value	Simple Use Case	5
Get Mean Absolute Deviation value	Simple Use Case	5
Get Standard Deviation	Simple Use Case	5
Get All Results	Simple Use Case	5
Save session	Simple Use Case	5
Load session	Simple Use Case	5
Exit	Simple Use Case	5

Table 14: (UUCW) Classification of use cases and their weights

To get the UUCW we add all the weights from the UUCW table. After adding all the weights we get the following result:

$$UUCW = (1 \cdot 10) + (11 \cdot 5) = 65$$

After applying the above equation, we get the following result for UUCP:

$$UUCP = 3 + 65 = 68$$

3.1.2 Calculate TCF

The Technical Complexity Factor (TCF) is used to measure the technical concerns that could impact the project. We can calculate the TCF with the following equation:

$$T_{CF} = C_1 + \left[C_2 \cdot X + \sum_{i=1}^{13} (WT_i \cdot F_i) \right]$$

Where C1 = 0.6 and C2 = 0.01, WTi is the the Technical Complexity Factor Weight, and Fi is the Perceived Impact Factor corresponding to each TCF. The development team will assign a Perceived Impact Factor for each TCF ranging from 0 to 5, depending on team's perception of the project's complexity. A Perceived Impact Factor of 0 means the TCF has no influence, 3 means average influence and 5 means strong influence.

TCF	Description	Weight	Factor	Result
1	Distributed System	2	0	0
2	Performance	1	5	5
3	End User Efficiency	1	5	5
4	Complex Internal Processing	1	2	2
5	Re-usability	1	3	3
6	Easy to Install	0.5	2	1
7	Easy to Use	0.5	5	2.5
8	Portability	2	3	6
9	Easy to Change	1	5	5
10	Concurrency	1	0	0
11	Special Security Features	1	3	3
12	Provides Direct Access for Third Parties	1	0	0
13	Special User Training Facilities are Required	1	0	0

Table 15: TCF Calculations

After we apply the above equation, we get the following result for TCF:

$$TCF = 0.6 + [0.01 \quad (0 + 5 + 5 + 2 + 3 + 1 + 2.5 + 6 + 5 + 0 + 3 + 0 + 0)] = 0.925$$

3.1.3 Calculate ECF

The Environment Complexity Factor (TCF) is used to account for th e development's team personal traits, including exience. We can calculate the ECF with the following equation:

$$E_{CF} = C_1 + \left[C_2 \cdot X + \sum_{i=1}^{8} (WE_i \cdot F_i) \right]$$

Where C1 = 1.4 and C2 = 0.03, WEi is the the Environmental Complexity Factor Weight, and Fi is the Perceived Impact Factor corresponding to each ECF. The development team will assign a Perceived Impact Factor for each ECF ranging from 0 to 5.

ECF	Description	Weight	Factor	Result
1	Familiarity with Use Case Domain	1.5	3	4.5
2	Part-Time Workers	-1	3	-3
3	Analyst Capability	0.5	3	1.5
4	Application Experience	0.5	3	1.5
5	Object-Oriented Experience	1	1	1
6	Motivation	0.5	3	1.5
7	Difficult Programming Language	-1	0	0
8	Stable Requirements	2	5	10

Table 16: ECF Calculations

After we apply the given equation, we get the following result for ECF:

$$ECF = 1.4 + [0.03 \quad (4.5 \quad 3 + 1.5 + 1.5 + 1 + 1.5 + 1 + 10)] = 0.89$$

3.1.4 Determine PF

The estimate for a new development team roughly would be around a PF of 20 person-hours per use case point. However, we believe that since this project does not require that much time and effort, therefore having an estimate of 20 hours per use case would be alot. Hence, we decided to change our PF to 5 person-hours per use case point as this seems more realistic and close to the actual work.

3.1.5 Determine PF

Now that we have the required variables (UUCP, TCF and ECF) we can calculate the UCP with the following formula:

$$UCP = UUCP\ X\ TCF\ X\ ECF$$

If we substitute the values in the above equation, we get the following result for UCP:

$$UCP = 68\ X\ 0.925\ X\ 0.89 = 55.981$$

3.1.6 Calculate the Estimated Number of Person-Hours

For the estimated effort, we use the following equation:

$$EffortEstimate = UCP\ X\ PF$$

After substituting the values in the above equation, we get the estimated effort required for this project using the UCP approach:

$$Effort\ Estimate = 55.981\ X\ 5 = 279.905\ person\text{-}hours$$

Using this approach the calculated effort will be 279.905 person-hours.

3.2 COCOMO 81

The COCOMO (Constructive Cost Model) is typically used to estimate the effort and duration of a project based on its size and complexity. COCOMO has three levels:
1. Basic
2. Intermediate
3. Detailed.

In the Basic COCOCMO model, projects are classified into three types
1. Organic
2. Semi-Detached
3. Embedded

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

Basic COCOMO 81 [11, 12] uses the following equation for effort estimation:

$$E = a \times (S)^b$$

E: Total effort required for the project in Person-Months (MM).
a, b: coefficients that depends on the type of software project
S: The size of the code for the project in Kilo lines of code

To get each of the values in the equation we did the following:

- The Project was developed by a small team of 5 members, has been done before, the problem is well understood, number of KLOC does not exceed 50, and the members have experience dealing with similar problems. Therefore, we can consider that the development will be organic. The coefficients **a** and **b** would be, **2.4** and **1.05** respectively.
- The software size (**S**) in KLOC, is **1.43**

– *METRICSTICS/_init_.py*: 0.002 KLOC.

– *METRICSTICS/main.py:* 0.019 KLOC.

– *METRICSTICS/tests/test_main_controller.py:* 0.085 KLOC.

– *METRICSTICS/tests/test_metrics_calculator.py:* 0.059 KLOC.

– *METRICSTICS/tests/_init_.py:* 0.003 KLOC.

– *METRICSTICS/tests/test_session_manager.py:* 0.054 KLOC.

– *METRICSTICS/shared/custom_exceptions.py:* 0.002 KLOC.

– *METRICSTICS/controller/_init_.py:* 0.003 KLOC.

– *METRICSTICS/controller/main_controller_interface.py:* 0.024 KLOC.

– *METRICSTICS/controller/main_controller.py:* 0.034 KLOC.

– *METRICSTICS/model/session_info_parser.py:* 0.030 KLOC.

– *METRICSTICS/model/session_manager.py:* 0.121 KLOC.

– *METRICSTICS/model/_init_.py:* 0.009 KLOC.

– *METRICSTICS/model/metrics_calculator_interface.py:* 0.016 KLOC.

– *METRICSTICS/model/session_manager_interface.py:* 0.022 KLOC.

– *METRICSTICS/model/metrics_calculator.py:* 0.144 KLOC.

– *METRICSTICS/model/session_info_parser_interface.py:* 0.016 KLOC.

– *METRICSTICS/view/metricstics_GUI.py:* 0.784 KLOC.

– *METRICSTICS/view/_init_.py:* 0.003 KLOC.

After we substitute the values for the effort estimation equation we get:

$$E = 2.4(1.43)^{1.05} = 3.4939 \tag{1}$$

If you assume 3 hours of work per day for a month, you would first calculate the total number of person-hours in a month.

$$\textit{Hours in a work month} = \textit{3 hours/day} * \textit{30 days} = 90$$

$$\begin{aligned}\textit{Effort (in person-hours)} &= \textit{Effort (in person-months)} * \textit{Hours in a work month} \\ &= \textit{3.4939}\times\textit{90} \\ &= \textit{314.451}\end{aligned}$$

Therefore, with the assumption of 3 hours of work per day for a month, the effort of 3.4939 person-months would be approximately equivalent to **314.451 person-hours.**

3.3 UCP vs COCOMO 81 vs actual effort

1. Methodology Differences:

- UCP relies on the quantification of functional requirements through use cases, taking into account factors such as complexity and actors. It focuses on the software's functional aspects.
- COCOMO 81 considers lines of code, development time, and project attributes. It is a more traditional approach that is based on historical data.

2. Subjectivity and Assumptions:

- UCP involves subjective assessments of use case complexity and other factors, which can vary between estimators.
- COCOMO 81 relies on certain assumptions about the project characteristics and team productivity, which may not always hold true.

3. Actual Project Dynamics:

- Real-world projects often face unforeseen challenges and changes that estimation models may not fully account for. Team dynamics, changes in requirements, and other external factors can impact actual effort.

The actual effort taken to complete the project was around 300 hours. This effort is close to the effort estimate made by the COCOMO model. Multiple factors may have caused reduced actual effort some of which are listed below:

1. Automated Testing:

- Automated testing involves using tools to automatically test software, identify bugs, and ensure code quality.
- Estimation models may not consider the time saved through automation when predicting project effort.

2. Parallel Development:

- Parallel development involves breaking down a project into smaller components that can be developed simultaneously by different team members.
- This strategy can accelerate the development timeline, but its impact may not be accurately reflected in estimation models that assume a linear development process.

4 DESCRIPTIVE-STATISTICS

(a) Using object-orientation as the programming paradigm and Python as the programming language, implement METRICSTICS from scratch. This means, apart from the functions related to input and output, basic arithmetic, and graphical user interface, an implementation of METRICSTICS must not make use of any reuse mechanism (such as built-in functions, libraries, or APIs) provided natively by the programming language or otherwise. This may lead to recursive implementation of certain primary functions, the result of which would be secondary functions.

(b) Using input data that consists of (at least) 1000 values, randomly distributed between 0 and 1000, test METRICSTICS and provide evidence of tests.

NOTES
It is better to aim for optimal but feasible algorithms. It could be noted that mathematical definitions of certain measures of descriptive statistics are not necessarily optimal for computer programming. The implementation of a graphical user interface for METRICSTICS using Tkinter is required.

4.1 Architecture

To make sure the source code is modular and maintainable we have followed the MVC Pattern where we divided the application into the following 3 tiers:

(a) **Model:**
Contains logic for calculating the statistics and manage user session information.

(b) **View:**
Contains logic of the GUI - Graphical user interface.

(c) **Controller:**
Contains logic to pull, modify, and provide data to the user. Essentially, the controller is the link between the view and model.

4.2 Implementation

4.2.1 Folder Structure

```
METRICSTICS
├── /.gitignore
├── /CODE_OF_CONDUCT.md
├── /CONTRIBUTING.md
├── /LICENSE
├── /README.md
├── /project_structure.txt
├── /requirements.txt
├── /__init__.py
├── /main.py
│   (Connects Model, View, and Controller for METRICSTICS, initializing components and launching
│   the Tkinter-based GUI.)
│
│
└── /[controller]
    ├── /__init__.py
    ├── /main_controller_interface.py
    ├── /main_controller.py
    │   (Contains controller logic)
```

```
├── /[view]
│   ├── /__init__.py
│   ├── /metricstics_GUI.py
│   │   (Contains user interface logic)
├── /[model]
│   ├── /__init__.py
│   ├── /metrics_calculator.py
│   │   (Contains statistics calculation logic)
│   │
│   ├── /metrics_calculator_interface.py
│   ├── /session_info_parser.py
│   │   (Contains JSON parsing logic)
│   │
│   ├── /session_info_parser_interface.py
│   ├── /session_manager.py
│   │   (Contains session management logic)
│   │
│   ├── /session_manager_interface.py
│
├── /[icons]
│   ├── /ai.png
│   ├── /close.png
│   ├── /file.png
│   ├── /icon.ico
│   ├── /keyboard.png
│
├── /[mock] (Contains test data)
│   ├── /invalid_user_input.txt
│   ├── /session_info.json
│   ├── /valid_user_input.txt
│
├── /[screenshots] (Application snapshots)
│   ├── /metricstics_gui_auto.png
│   ├── /metricstics_gui_file.png
│   ├── /metricstics_gui_keyboard.png
│
├── /[sessions] (Location to store session information)
│   ├── /[datasets]
│   │   ├── /23d8c404-3f14-4c7b-9b64-5c9d9f381b6b.txt
│   │   ├── /3f08bc04-7cb9-4c1b-8260-a8f450fa3f5a.txt
│   ├── /session_info.json
│
├── /[shared] (Contains shared components)
│   ├── /custom_exceptions.py
│
├── /[tests] (Contains unit test cases)
│   ├── /__init__.py
│   ├── /test_main_controller.py
│   ├── /test_metrics_calculator.py
│   ├── /test_session_manager.py
```

4.3 Functions

4.3.1 metrics_calculator.py

class MetricsCalculator implements MetricsCalculatorInterface

a) *calculate_metrics()*
Sorts the list passed as parameter, calculates the statistics and returns them as a dictionary.

b) *get_results()*
Returns previous calculated results. Used for session management.

c) *clear_data()*
Clears the calculated result.

d) *min()*
Gets the smallest number in the given data set.

e) *max()*
Gets the largest number in the given data set.

f) *mode()*
Returns the value or the list of values which occurred the maximum number of times in the list.

g) *mean()*
Calculates the Arithmetic mean of the elements of the list from the parameter.

h) *median()*
Gets the middle number if the list length is odd, and is the arithmetic mean of the two middle numbers if the list length is even.

i) *mad()*
Calculates the arithmetic mean of the data set.

j) *standard_deviation()*
Calculates the standard deviation from the data set given a arithmetic mean.

k) *calculate_square_root()*
Helper function to calculate square root of a number.

l) *calculate_absolute_difference()*
Helper function to calculate absolute difference.

m) *calculate_sum()*
Helper function to calculate sum of numbers in a given list.

n) *find_max_frequency()*
Helper function to calculate frequency of numbers in a given list.

4.3.2 session_manager.py

class SessionManager implements SessionManagerInterface

a) *__init__()*
Constructor, initializes session information directory and file.

b) *save_session()*
Saves user session information (dataset and results).

c) *get_session_by_id()*
Returns the session information for the provided id.

d) *get_all_sessions()*
Returns list all user's saved sessions.

e) *get_session_directory_path()*
Helper function, returns the path to the selected session folder.

f) *get_session_path()*
Helper function, returns the path to the selected session json file.

g) *create_session_item()*
Helper function to create session JSON item.

h) *generate_dataset_filepath()*
Helper function that generates and returns path to store the user's dataset.

i) *save_dataset_file()*
Helper function to save user dataset into a file and save it at the location provided by generate_dataset_filepath().

4.3.3 session_info_parser.py

class SessionInfoParser implements SessionInfoParserInterface

a) *__init__()*
Constructor, initializes the json item properties based on provided dictionary

b) *parse_results()*
Parses into specified format.

c) *__repr__()*
Returns stringified JSON.

4.3.4 main_controller.py

class MainController implements MainControllerInterface

a) *__init__()*
Constructor, assigns the MetricsCalculator instance and SessionManager instance to use

b) *calculate_metrics()*
Requests MetricsCalculator model to calculate the metrics and returns the results.

c) *clear_data()*
Requests MetricsCalculator model to clear its previously calculated results.

d) *get_all_sessions()*
Requests SessionManager to provide list of saved user sessions and returns this list.

e) *save_session()*
Requests MetricsCalculator model to provide the calulated results. If there are results then it requests SessionManager to save the session. Else raises ResultsNotAvailableError error.

f) *load_session()*
Requests SessionManager to provide information of the session based on the given id and returns the session information.

4.3.5 metricstics_GUI.py

a) *__init__()*
Constructor, assigns the MainControllerInterface instance to use

b) *set_dark_theme()*
Set tkinter dark theme

c) *clear_action()*
Clears the UI components and model including dataset text area.

d) *clear_results()*
Clears the UI components and model excluding dataset text area.

e) *get_user_input()*
Updates the label on perform logic based on user selected input mechanism.

f) *open_popup()*
Opens a dialog box to ask user to provide a session name when user clicks on save button

g) *save_action()*
Performs save procedure.

h) *on_text_area_change()*
Ensures that selected user input method is set to keyboard when user presses a key in the input text area

i) *generate_action()*
Requests controller to provide statistics results for user specified input. Checks for input validity. Updates UI Components once the statistics are computed and returned by the controller.

j) *on_item_click()*
Loads the particular clicked session item.

k) *format_timestamp()*
Helper method to format timestamp in dd-mmm-yyyy hh:mm:ss

l) *validate_entry()*
Helper method to make sure dialogue box save button is disabled if session name is not entered by the user.

m) *on_enter()*
Helper method to make sure components background color stays consistent on hover.

4.3.6 main.py

a) *__name__ == "__main__"*
Application entry point. Sets up the components of the application using the MVC pattern: Models for data handling, a controller for logic and data flow, and a view for user interface interactions. The Tkinter main loop keeps the GUI running, enabling user interaction.

4.4 Testing

For testing the **model, controller and shared classes** we wrote unit test cases.
We used coverage.py tool to calculate the code coverage [15]. We achieved code coverage of 88%. We can further improve the code coverage further by refactoring and adding more unit tests.

Installation
pip install coverage

Usage

(a) Run the unit test:
coverage run -m unittest discover -s ¡path_to_tests_directory¿

(b) Generate coverage report:
coverage report -m

(c) Generate more detailed html report
coverage html

(d) Open the generated index.html file

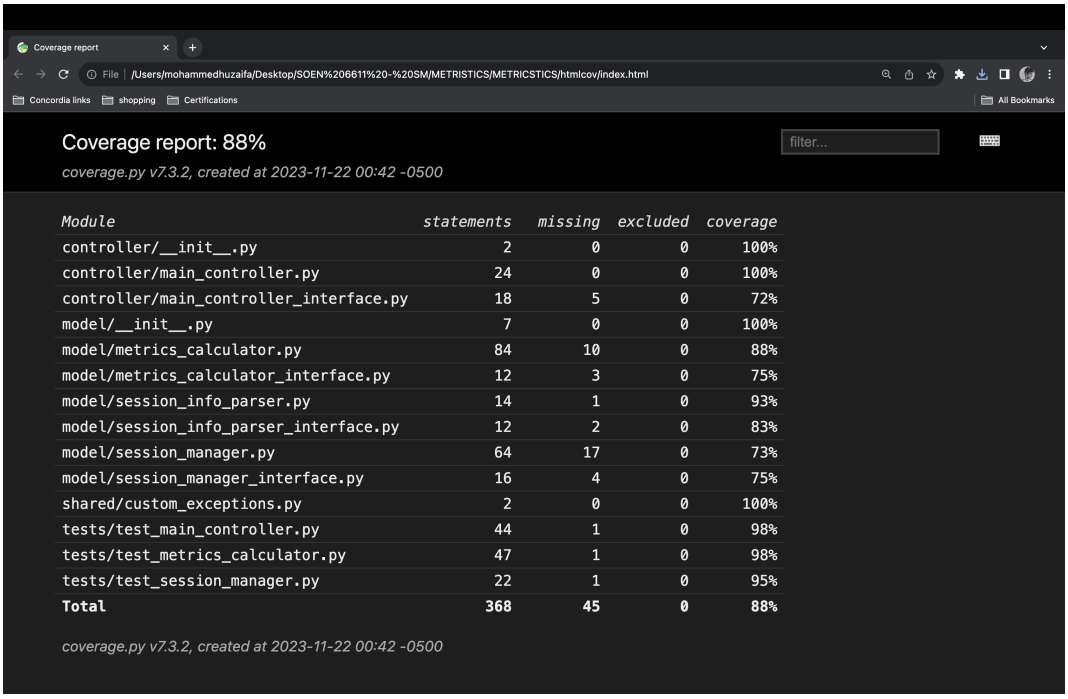


Figure 1: Saving a session.

For testing the **view class** i.e. GUI we performed manual testing.
To review the test evidence, you can access the document 'Test case execution.pdf' located in the 'docs' folder on GitHub. Visit the following link: Test case execution.pdf within the METRICSTICS repository. This document contains details on various executed test cases and their outcomes.

4.5 User Manual

Important: This user manual offers guidance on utilizing the application exclusively through keyboard inputs.

The user will see the following screen when they run the applicaiton.
User can select from 3 different types of inputs:

(a) Keyboard - manually enter using a keyboard

(b) File - read input from a file

(c) Auto - generate random numbers between 0 - 1000 in range 1000 - 10,000

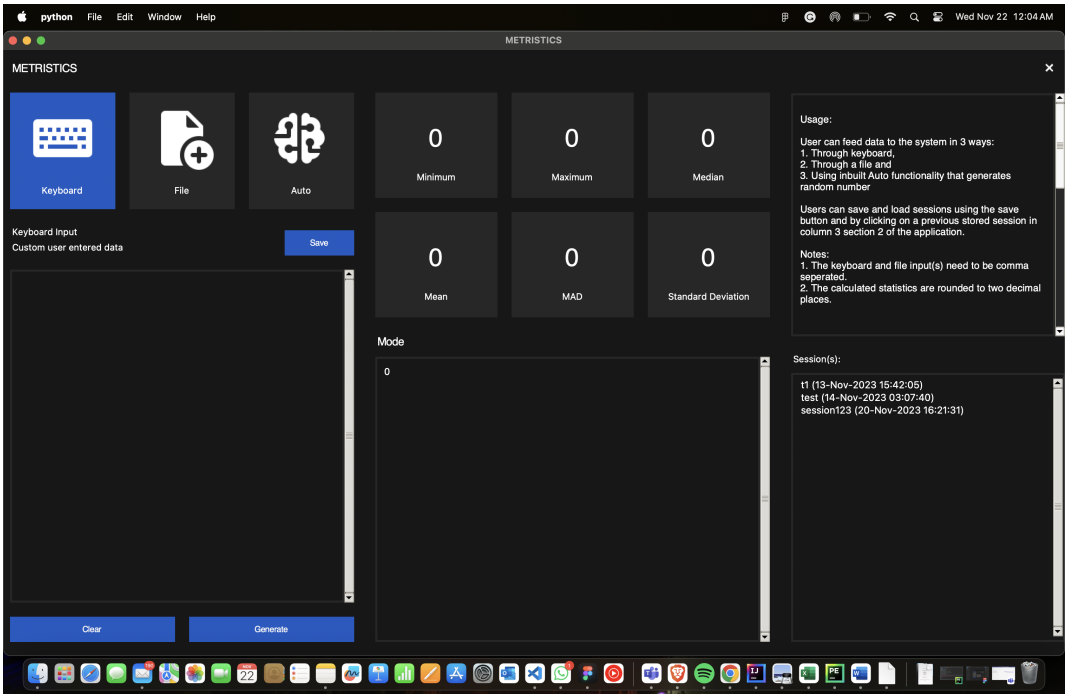


Figure 2: Application user interface

User enters the data

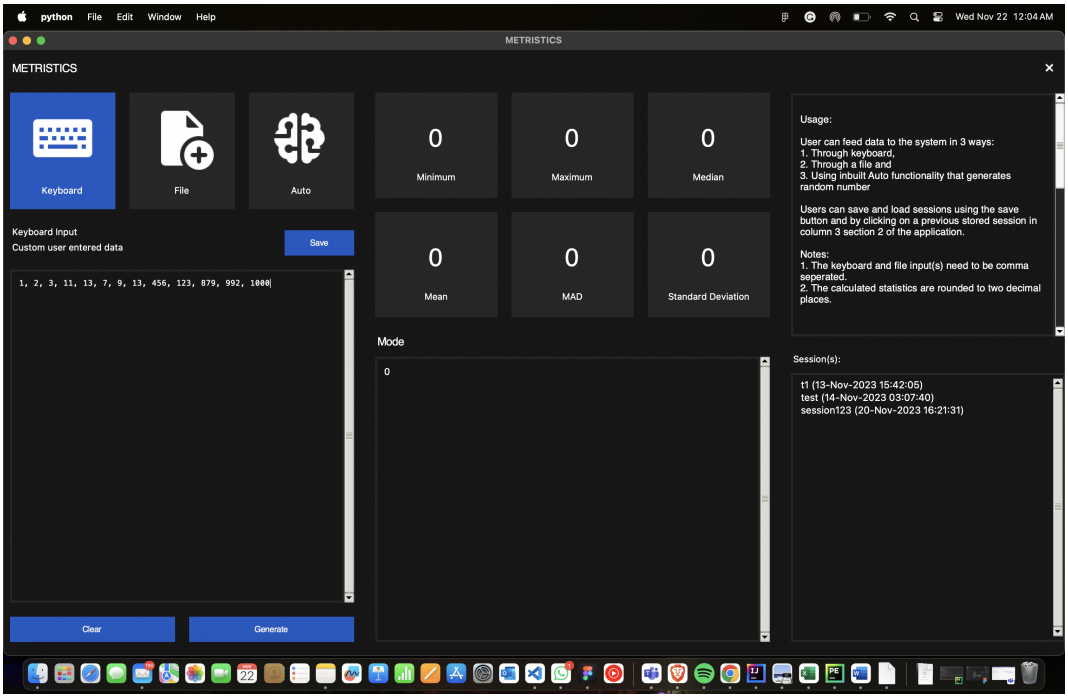


Figure 3: Providing input

User clicks on generate button to generate statistics

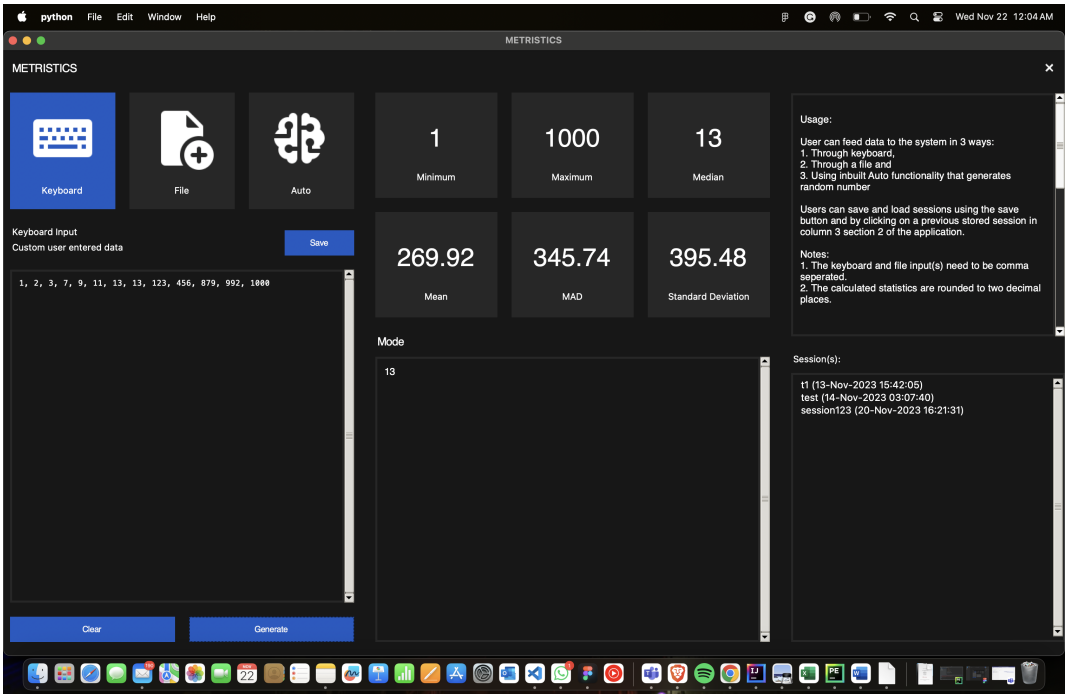


Figure 4: Generating results

User clicks on save button to save a session and saves the session

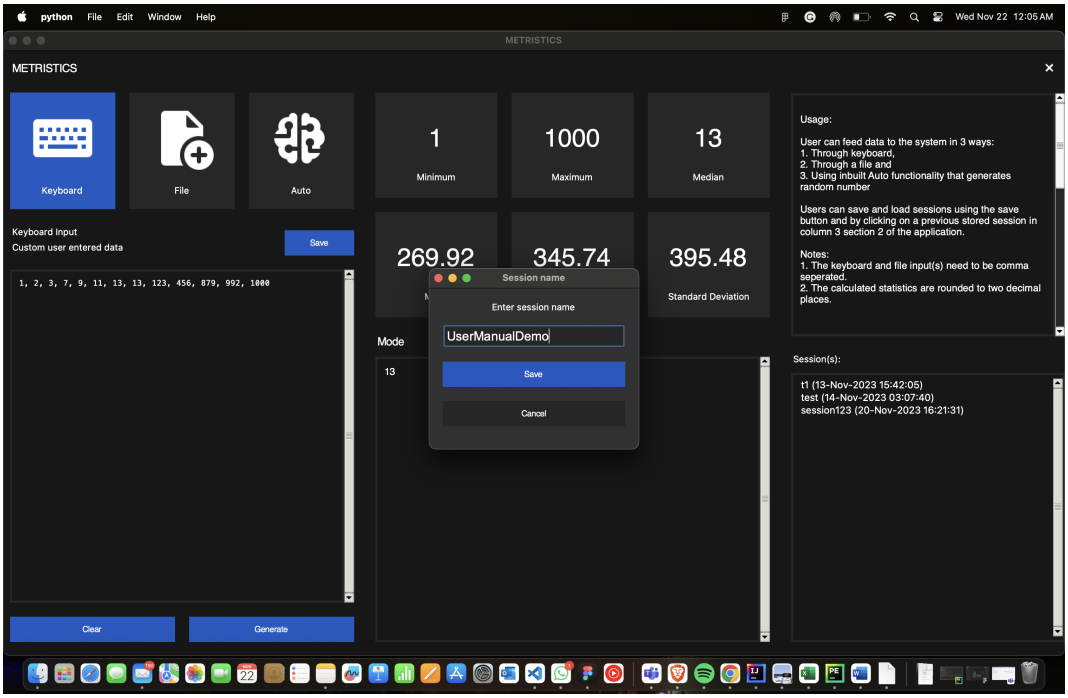


Figure 5: Saving a session.

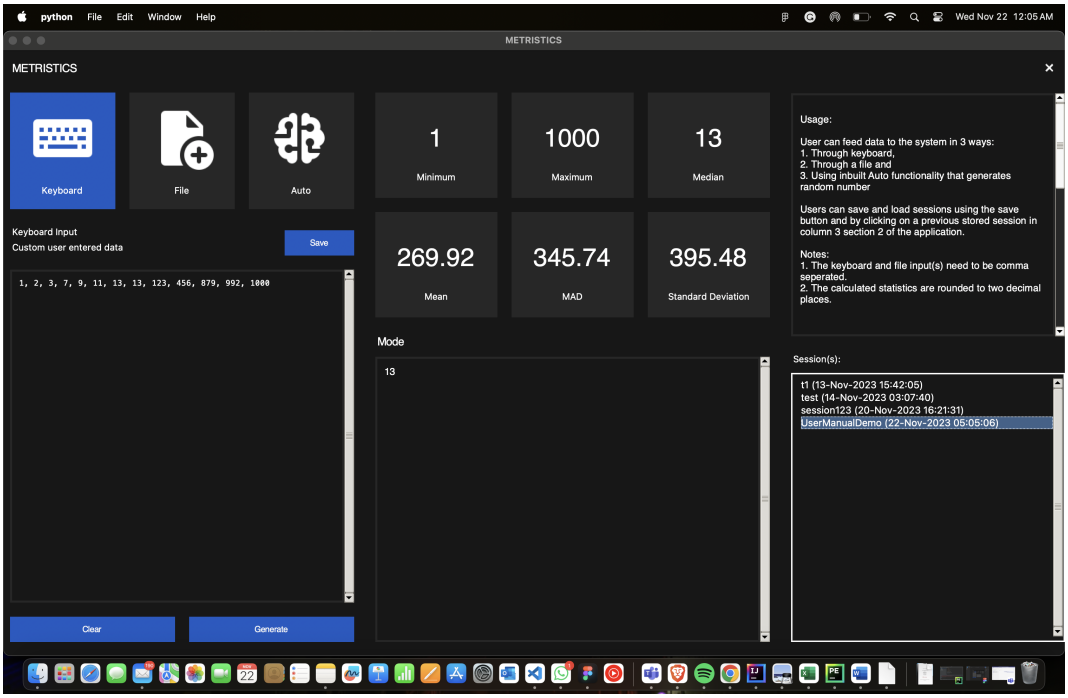


Figure 6: Session saved successfully.

User clicks on clear button to save a session

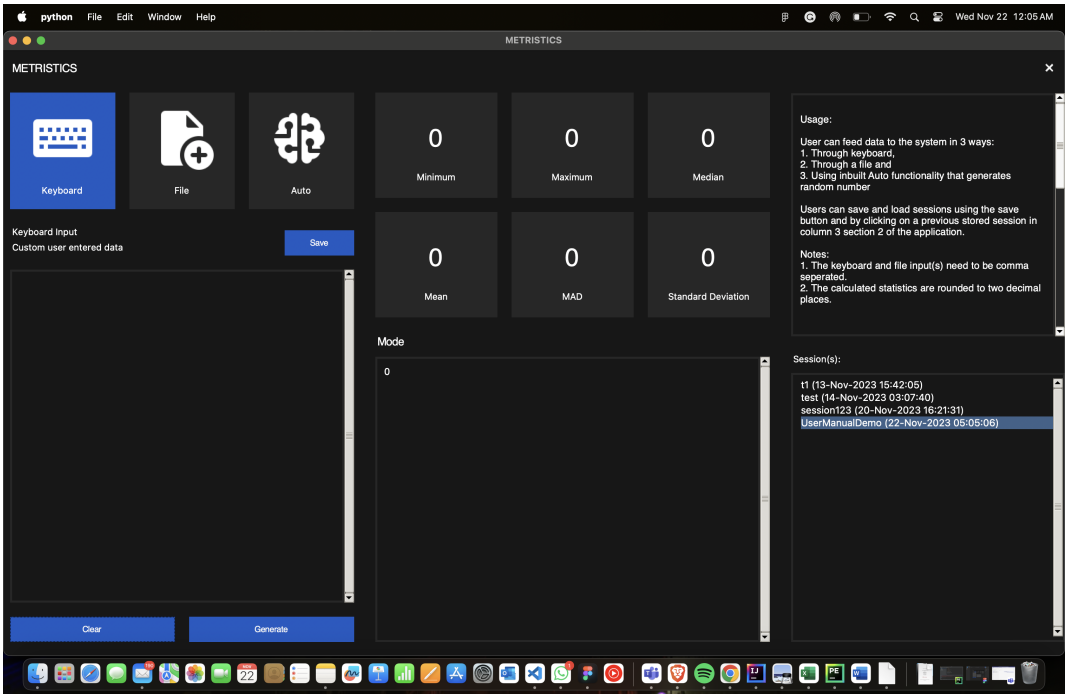


Figure 7: Clearing data and results

User clicks on load button to load a session

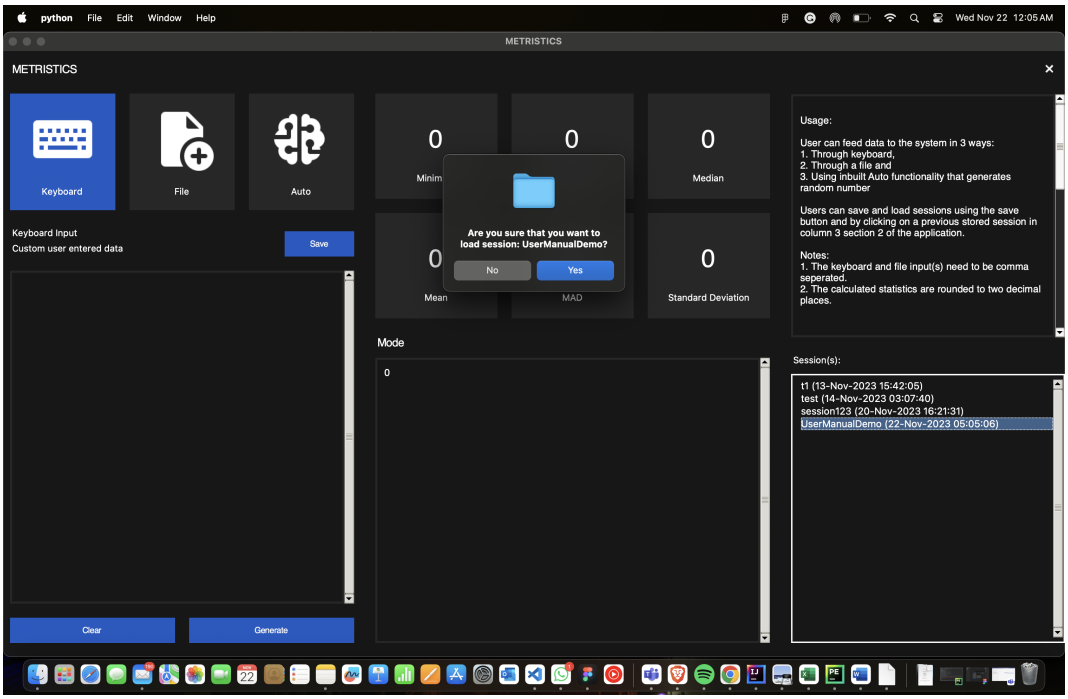


Figure 8: Selecting a session

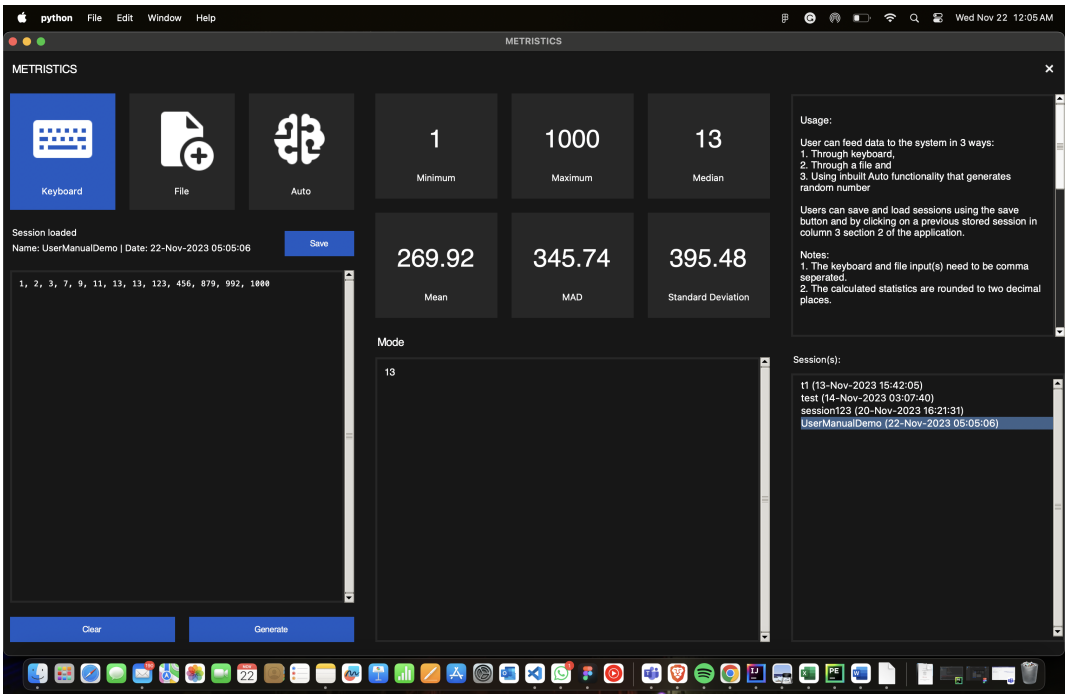


Figure 9: Selected session loaded

4.5.1 Rationale

1. Trello [11] tool was used to manage the work distribution between the team members. (members had prior experience with this tool).
2. GitHub was used for the software development version control [12]. (members had prior experience with this tool).
3. The MVC[13] pattern was used to make the code more modular and maintainable.
4. All the logic for gathering the input data set is managed by MetricsticsGUI class, it should be independent of the statistical algorithms.
5. The MVC pattern divided the application into 3 layers where each layer can be used independently of another. High cohesion and Moderate coupling.
6. We allowed the user to have 3 input methods (keyboard input, file input, auto - random input).
7. To generate a random data set, we used the python's randint method from Random class. It is the quickest way to obtain a pseudo random numbers.
8. To efficiently calculate the different statistics we initially sort the input data set using python's built in sorted method. It uses Timsort[14] to sort the elements. We used this method because it has low time complexity of O(NlogN).
9. After a valid dataset is entered into the system. We calculate all the statistics together as they are a few and the user is more likely to ask for results rather than a specific one.
10. For functions that required some basic computations time to time we created helper functions to prevent unnecessary repetation of code.

5 Cyclomatic Number

(a) Calculate the cyclomatic number (also known as cyclomatic complexity) of DESCRIPTIVE-STATISTICS.

(b) Comment on the qualitative conclusions that can be drawn with respect to the quantitative thresholds of the metric.

Note. It is better to aim for a small cyclomatic number.

5.1 Cyclomatic Complexity

Cyclomatic Complexity, an essential software quality metric conceptualized by Thomas J. McCade, Sr. [15], evaluates a program's quality. It specifically quantifies the quantity of unique paths within a program's source code, directly relating to the code's complexity.

The cyclomatic complexity as a metric can be easily understood, implemented, used, and reported in a short period of time with little effort. Therefore the calculation of Cyclomatic Number has become an interesting subject.[16]

Consequently the more complex the code becomes the more likely the program will be less understandable and the consequences are:

- Harder to read
- Harder to modify
- Harder to maintain
- Harder to analyze
- Harder to test
- More defects

5.2 Perspectives of Cyclomatic Number

Cyclomatic number can be observed from the following different perspectives:

1. Perspective of CONTROL FLOW GRAPHS
2. Perspective of STRONGLY CONNECTED GRAPHS
3. Perspective of DECISION NODES

Problem: The perspectives mentioned above are based on the assumption that the program has only one function and a single exit node.

5.3 Sequential Cyclomatic Number (SCN)

The Sequential Cyclomatic Number (SCN) broadens the traditional CN concept. Unlike CN, which supports a single function, SCN accommodates a sequence of function calls. [17]
For the following equation, CN denotes the cylcomatic number, f is a function, f_{cg} be a calling function, f_{cd} is called function and n be the number of function calls:

$$\mathrm{SCN}(f) = \mathrm{CN}(f), \text{ if } n = 0,$$

and

$$SCN(\mathrm{fcg}) = CN(\mathrm{fcg}) + P_n \cdot \sum_{i=1}^{n} CN(\mathrm{fcd}(i)), \quad \text{if } n > 0.$$

5.4 Maximum Cyclomatic Number (MCN)

The Maximum Cyclomatic Number (MCN) represents the highest cyclomatic number within a class, determined through the following equation:

$$\text{MCN} = \max\{CN(m_1), \ldots, CN(m_n), m_i \in C\}$$

5.5 Calculating the cyclomatic number

To calculate the cyclomatic numbers we used the Radon Python tool [18] which computes various code metrics. This tool is an open source tool that can be used either from the command line or programmatically through its API. It supports calculation of the following metrics:

- Raw metrics: SLOC, comment lines, blank lines and c.
- Cyclomatic Complexity (i.e. McCabe's Complexity)
- Halstead metrics (all of them)
- the Maintainability Index (a Visual Studio metric)

Installation
pip install radon

Usage
radon cc METRICSTICS -a

cc: compute Cyclomatic Complexity
-a: calculate the average complexity at the end

Cyclomatic number of our project:

Number of blocks analyzed: 95 (classes, functions, methods).
Average complexity: A (1.5894736842105264)

```
METRICSTICS/tests/test_main_controller.py
    C 8:0 TestMainController - A (2)
    M 10:4 TestMainController.setUp - A (1)
    M 15:4 TestMainController.test_calculate_metrics - A (1)
    M 30:4 TestMainController.test_clear_data - A (1)
    M 37:4 TestMainController.test_save_session_with_results - A (1)
    M 47:4 TestMainController.test_save_session_without_results - A (1)
    M 55:4 TestMainController.test_get_all_sessions - A (1)
    M 69:4 TestMainController.test_load_session - A (1)
METRICSTICS/tests/test_metrics_calculator.py
    C 4:0 TestMetricsCalculator - A (2)
    M 6:4 TestMetricsCalculator.setUp - A (1)
    M 10:4 TestMetricsCalculator.test_values_data - A (1)
    M 22:4 TestMetricsCalculator.test_even_number_of_data_points - A (1)
    M 34:4 TestMetricsCalculator.test_duplicate_values - A (1)
    M 46:4 TestMetricsCalculator.test_large_dataset - A (1)
METRICSTICS/tests/test_session_manager.py
    C 8:0 TestSessionManager - A (2)
    M 10:4 TestSessionManager.setUp - A (1)
    M 15:4 TestSessionManager.test_save_session - A (1)
METRICSTICS/shared/custom_exceptions.py
    C 1:0 ResultsNotAvailableError - A (1)
```

```
METRICSTICS/controller/main_controller_interface.py
    C 5:0 MainControllerInterface - A (2)
    M 7:4 MainControllerInterface.calculate_metrics - A (1)
    M 11:4 MainControllerInterface.clear_data - A (1)
    M 15:4 MainControllerInterface.get_all_sessions - A (1)
    M 19:4 MainControllerInterface.save_session - A (1)
    M 23:4 MainControllerInterface.load_session - A (1)
METRICSTICS/controller/main_controller.py
    C 8:0 MainController - A (2)
    M 25:4 MainController.save_session - A (2)
    M 9:4 MainController.__init__ - A (1)
    M 15:4 MainController.calculate_metrics - A (1)
    M 19:4 MainController.clear_data - A (1)
    M 22:4 MainController.get_all_sessions - A (1)
    M 33:4 MainController.load_session - A (1)
METRICSTICS/model/session_info_parser.py
    C 5:0 SessionInfoParser - A (2)
    M 6:4 SessionInfoParser.__init__ - A (1)
    M 14:4 SessionInfoParser.parse_results - A (1)
    M 25:4 SessionInfoParser.__repr__ - A (1)
METRICSTICS/model/session_manager.py
    M 95:4 SessionManager.get_session_by_id - B (6)
    M 28:4 SessionManager.save_session - A (5)
    M 108:4 SessionManager.get_all_sessions - A (5)
    C 10:0 SessionManager - A (3)
    M 12:4 SessionManager.__init__ - A (1)
    M 18:4 SessionManager.get_session_directory_path - A (1)
    M 23:4 SessionManager.get_session_path - A (1)
    M 49:4 SessionManager.create_session_item - A (1)
    M 73:4 SessionManager.generate_dataset_filepath - A (1)
    M 87:4 SessionManager.save_dataset_file - A (1)
METRICSTICS/model/metrics_calculator_interface.py
    C 5:0 MetricsCalculatorInterface - A (2)
    M 7:4 MetricsCalculatorInterface.calculate_metrics - A (1)
    M 11:4 MetricsCalculatorInterface.clear_data - A (1)
    M 15:4 MetricsCalculatorInterface.get_results - A (1)
METRICSTICS/model/session_manager_interface.py
    C 7:0 SessionManagerInterface - A (2)
    M 9:4 SessionManagerInterface.get_session_path - A (1)
    M 13:4 SessionManagerInterface.save_session - A (1)
    M 17:4 SessionManagerInterface.get_session_by_id - A (1)
    M 21:4 SessionManagerInterface.get_all_sessions - A (1)
METRICSTICS/model/metrics_calculator.py
    M 69:4 MetricsCalculator.mode - B (7)
    M 119:4 MetricsCalculator.find_max_frequency - A (4)
    C 5:0 MetricsCalculator - A (3)
    M 61:4 MetricsCalculator.median - A (3)
    M 97:4 MetricsCalculator.standard_deviation - A (3)
    M 93:4 MetricsCalculator.mad - A (2)
    M 102:4 MetricsCalculator.calculate_square_root - A (2)
    M 109:4 MetricsCalculator.calculate_absolute_difference - A (2)
    M 113:4 MetricsCalculator.calculate_sum - A (2)
    M 9:4 MetricsCalculator.__init__ - A (1)
    M 14:4 MetricsCalculator.get_results - A (1)
    M 18:4 MetricsCalculator.calculate_metrics - A (1)
    M 52:4 MetricsCalculator.min - A (1)
    M 55:4 MetricsCalculator.max - A (1)
    M 58:4 MetricsCalculator.mean - A (1)
    M 135:4 MetricsCalculator.clear_data - A (1)
```

```
METRICSTICS/model/session_info_parser_interface.py
    C 5:0 SessionInfoParserInterface - A (2)
    M 7:4 SessionInfoParserInterface.__init__ - A (1)
    M 11:4 SessionInfoParserInterface.parse_results - A (1)
    M 15:4 SessionInfoParserInterface.__repr__ - A (1)
METRICSTICS/view/metricstics_GUI.py
    C 64:0 ScrollableLabelFrame - A (3)
    C 236:0 MetricsticsGUI - A (3)
    M 65:4 ScrollableLabelFrame.__init__ - A (2)
    C 104:0 InfoCard - A (2)
    M 124:4 InfoCard.update_number - A (2)
    C 135:0 IconButton - A (2)
    M 176:4 IconButton.on_leave - A (2)
    C 190:0 PrimaryButton - A (2)
    C 213:0 SecondaryButton - A (2)
    M 246:4 MetricsticsGUI.__init__ - A (2)
    M 96:4 ScrollableLabelFrame.setContent - A (1)
    M 105:4 InfoCard.__init__ - A (1)
    M 121:4 InfoCard.grid - A (1)
    M 136:4 IconButton.__init__ - A (1)
    M 168:4 IconButton.on_button_click - A (1)
    M 172:4 IconButton.on_enter - A (1)
    M 181:4 IconButton.select - A (1)
    M 185:4 IconButton.deselect - A (1)
    M 191:4 PrimaryButton.__init__ - A (1)
    M 214:4 SecondaryButton.__init__ - A (1)
    M 783:4 MetricsticsGUI.run - A (1)
```

Block Type	**Letter**
Function	F
Method	M
Class	C

Table 17: Block Type and Letter mapping

CC	**Rank**	**Risk**
1 - 5	A	low - simple block
6 - 10	B	low - well structured and stable block
11 - 20	C	moderate - slightly complex block
21 - 30	D	more than moderate - more complex block
31 - 40	E	high - complex block, alarming
41+	F	very high - error-prone, unstable block

Table 18: Rank, cyclomatic complexity score and risk mapping

Based on the Radon tool's cyclomatic number calculation, Table 18 and Table 19, we observe that the cyclomatic complexity metric for almost all the classes and methods is low (i.e. between 1-5).

1. Rank A represents a cyclomatic number that ranges between 1-5 which has a low risk with simple block of code assessment.

2. Rank B represents a cyclomatic number that ranges between 6-10 which has a low risk with well structured and stable block of code assessment.

3. Rank C represents a cyclomatic number that ranges between 11-20 which has a moderate with slightly complex block of code assessment.

4. Rank D represents a cyclomatic number that ranges between 21-30 which has a more than moderate with more complex block of code assessment.

5. Rank E represents a cyclomatic number that ranges between 31-40 which has a high risk with complex block, alarming assessment.

6. Rank F represents a cyclomatic number that ranges between 41+ which has a very high with error-prone, unstable block of code assessment.

Lower cylcomatic number represents less complexity in the classes. Therefore, our aim should be to have a better cyclomatic rank. Better the rank, less is the cyclomatic number.

As discussed in Subsection 5.4, to calculate the cyclomatic number of a class, we get the MCN of the class. Therefore we get the following results:

Class Name	**Cyclomatic Number**
TestMainController	2
TestMetricsCalculator	2
TestSessionManager	2
ResultsNotAvailableError	1
MainController	2
SessionInfoParser	2
SessionManager	6
MetricsCalculator	7
MetricsticsGUI	3

Table 19: Cyclomatic Number of major Classes

As shown in Table 20, we can see that MetricsCalculator class has a more complex control flow compared to other classes. This is due to the MetricsCalculator mode method which has a high cyclomatic number.

5.6 Evaluation of the tool

Given that tools can potentially introduce issues and yield incorrect results, we opted to manually calculate the cyclomatic number for classes. Below presents an instance contrasting the manually computed result against that generated by the tool. Our manual counting approach aligns with the principles outlined in Theorem 4 [19]

$$v(G)=d+1$$

Considering the mode method code from MetricsCalculator class below:

```
def mode(self, sorted_data):
    frequency_map = {}

    # Count the frequency of each element in the sorted data
    for num in sorted_data:
        if num in frequency_map:
            frequency_map[num] += 1
        else:
            frequency_map[num] = 1

    # Find the maximum frequency without using max function
    max_frequency = 0
    for freq in frequency_map.values():
        if freq > max_frequency:
            max_frequency = freq

    # Find the mode values without using list comprehension
    mode_values = []
    for num, freq in frequency_map.items():
        if freq == max_frequency:
            mode_values.append(num)

    return mode_values
```

We can see that there are 7 decisions (3 for loops, 3 if statements and a return statement) in the code given above. If we apply equation 5.6 the result turns out to be 7. The results of the tool is as shown below:

```
METRICSTICS/model/metrics_calculator.py
    M 69:4 MetricsCalculator.mode - B (7)
```

As evident from the obtained result of 7, which aligns with our manual calculation, we can confidently conclude that the tool accurately calculates the cyclomatic number.

5.7 Qualitative conclusions

The analysis of cyclomatic complexity using the Radon tool revealed that the majority of classes and methods exhibit low cyclomatic numbers, typically falling within the 1-5 range. However, it's apparent that the cyclomatic number alone does not account for code quality. Striving for enhanced code quality remains pivotal, emphasizing the aim for lower cyclomatic numbers to reduce complexity. Notably, the MetricsCalculator class shows higher complexity in its control flow, notably with the mode method registering a cyclomatic number of 7. To ensure precision, manual verification of cyclomatic numbers was conducted alongside tool-based calculations, validating the tool's accuracy in determining complexity metrics.

6 Calculate Object Oriented Metrics

(a) Calculate the object-oriented metrics, WMC, CF, and LCOM*, for each of the classes of DESCRIPTIVE STATISTICS. For WMC, assume that the weights are not normalized. Show your calculations in detail, manually or automatically (using a tool), as applicable.

(b) Comment on the qualitative conclusions that can be drawn with respect to the quantitative thresholds of the respective metrics.

Note: It is better to aim for values that are within the respective thresholds allowed.

6.1 Weighted Method Per Class(WMC)

The Weighted Method Count (WMC) metric, originally defined in the "Metrics Suite for Object Oriented Design" by Chidamber and Kemerer, measures the complexity of a class by summing up the complexities of all its methods. This metric serves as an indicator of the effort required for maintaining and developing a class. The WMC is calculated by adding up the complexity values of each method in the class, denoted as ci(Mi) for the ith method.[20]

$$WMC = \sum_{i=1}^{n} (c_i \cdot M_i)$$

where n is the total number of methods and $c_i(M_i)$ is the complexity of the method. The above mentioned complexity is not something that is generally agreed upon. To make WMC general in its applicability, there exists two possibilities for the value of c_i:

1. **Normalization**
   If the complexity values (c_i) are normalized to one, WMC equates to the total number of methods in the class, making it a straightforward count of methods

2. **Non-Normalization**
   In this approach, c_i represents the actual cyclomatic complexity of each method. The WMC is then the sum of these cyclomatic complexities for all methods in the class. For this discussion, we consider the non-normalization approach, where weights are not normalized, meaning WMC represents the combined cyclomatic complexity of all methods.

WMC is relevant to several key attributes of a class, including its analyzability, modifiability, and testability. The WMC values provided for each class are based on the cyclomatic complexity of their methods.

The attributes relevant to Weighted Method per Class are **analyzability, modifiability, and testability.**

Below we have the WMC for each class based on the cyclomatic number of the methods using the formula mentioned above:

Class Name	WMC
MainController	9 (2+2+1+1+1+1+1)
MetricsCalculator	35 (7+4+3+3+3+2+2+2+2+1+1+1+1+1+1)
SessionInfoParser	5 (2+1+1+1)
SessionManager	25 (6+5+5+3+1+1+1+1+1+1)
TestMetricsCalculator	7 (2+1+1+1+1+1)
TestSessionManager	10 (3+3+2+1+1)
ScrollableLabelFrame	6 (3+2+1)
MetricsticsGUI	6 (2+2+1+1)
InfoCard	6 (2+2+1+1)
IconButton	9 (1+1+1+1+1+2+2)
PrimaryButton	3 (1+2)
SecondaryButton	3 (1+2)

Table 20: WMC of major Classes

Qualitative conclusion(s)

In Object-Oriented Design (OOD) metrics, specific thresholds for evaluation are not set, allowing us to gauge the complexity of classes relative to each other. This relative assessment helps in understanding the time and effort needed for maintenance. The total WMC value for our application came out to be 124. From the calculated metrics, it's evident that class MetricsCalculator and class SessionManager are more complex compared to others, indicating they will require more effort in terms of maintenance. This complexity also suggests a higher likelihood of encountering bugs in these classes, and they will necessitate more time for developing thorough test cases. In the context of an isolated project, the complexity levels can be considered quite manageable.

6.2 Coupling Factor(CF)

The Coupling Factor (CF) is used to quantify the average level of inter-class coupling in Object-Oriented Design (OOD), excluding any coupling that arises from inheritance. It's considered an external attribute, focusing on the interdependence between different software modules. Coupling, in essence, measures the interconnectedness of modules or routines, highlighting the strength of their relationships. This concept is often contrasted with cohesion, with a preference for a low coupling factor to achieve better quality results.

In an OOD, classes can be represented as C1, ..., Cn, with 'n' being the total number of classes. The CF measures relationships between classes where IsClient(Ci, Cj) = 1 indicates a relationship between class Ci and Cj, such as method calls, references, or attribute access, provided it's not due to inheritance. The formula for CF is as follows:

CF = Total Actual Couplings (excluding inheritance)/Maximum Possible Couplings in OOD

Here, the numerator represents the actual number of couplings (excluding those from inheritance), and the denominator represents the maximum number of possible couplings in an OOD. The coupling factor can be

found with the following equation[21]:

$$CF = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \text{IsClient}(C_i, C_j) \right) / (n^2 - n)$$

Class Name	**CF**
MainController	3
MainControllerInterface	0
MetricsCalculator	8
MetricsCalculatorInterface	0
SessionInfoParser	4
SessionInfoParserInterface	0
SessionManager	16
SessionManagerInterface	0
ScrollableLabelFrame	5
InfoCard	5
IconButton	2
PrimaryButton	3
SecondaryButton	3
MetricsticsGUI	73

Table 21: CF (COUPLING FACTOR) of classes

CF calculation:

$$CF = \frac{(3+0+8+0+4+0+16+0+5+5+2+3+3+73)}{14^2 - 14} = 0.73$$

Qualitative conclusion(s)

Based on the complexity of the application we selected CF threshold to be 0.5 which is indicative of moderate coupling.

Analyzability, modifiability, and testability are characteristics that are pertinent to coupling factor. The degree of interdependence between software modules is known as coupling, and a low coupling factor means that classes are relatively independent of one another. This has a favourable effect on the previously mentioned traits and increases the maintainability of the codebase. In this case, the coupling factor is nearly 0.7 which is a bit higher than the threshold. This can be further improved by performing code refactoring.

6.3 Lack of Cohesion in methods (LCOM*)

In Object-Oriented Design, the Lack of Cohesion in Methods (LCOM*) metric is used to evaluate the cohesion within a class. It's based on the interaction between methods and attributes of the class. Higher cohesion is indicated by lower LCOM* values. Here's an explanation along with an example of how LCOM* is calculated [19]:

Method-Attribute Relationship: Consider a class with m methods (M1, M2,..., Mm) and a attributes (A1, A2, ..., Aa). Let $\mu(A_k)$ be the count of methods that access attribute Ak.

LCOM Calculation*: The formula for LCOM* is based on the relationship between methods and attributes. It's calculated as [22]:

$$LCOM^* = \frac{\left(\frac{1}{a}\sum_{i=1}^{a} A_i\right) - m}{1 - m}$$

This formula sums the number of methods accessing each attribute, normalizes it by the number of attributes, and then compares it to the total number of possible method-attribute accesses.

Interpreting LCOM* Values:

$0 \leq \text{LCOM}^* \leq 1$: A value of 0 indicates high cohesion, while a value closer to 1 indicates low cohesion.
Desired Outcome: Lower LCOM* values are preferable as they signify higher cohesion within the class. So, it means that **low** values of LCOM* is preferred in order to have high cohesion.

Considering the attributes, methods and their usage in different method we calculated the LCOM* values. The calculation are shown below:

1. **SessionInfoParser**
   a = 5
   m = 3
   μ(self.id) = 1 (accessed by 1 method)
   μ(self.name) = 1 (accessed by 1 method)
   μ(datasetFilePath) = 1 (accessed by 1 method)
   μ(self.timestamp) = 1 (accessed by 1 method)
   μ(self.result) = 2 (accessed by 2 method)

$$LCOM^* = \frac{\left(\frac{1}{5} \cdot (1 + 1 + 1 + 1 + 2) - 3\right)}{1 - 3} = 0.9$$

2. **SessionManager**
   a = 2
   m = 13
   μ(self.SESSIONS_FOLDER) = 4 (accessed by 4 method)
   μ(self.SESSION_FILE_NAME) = 2 (accessed by 2 method)

$$LCOM^* = \frac{\left(\frac{1}{2} * \ (4 + 2) - 13\right)}{1 - 13} = 0.83$$

3. **MetricsCalculator** a = 2
   m = 15
   μ(self.sorted_data) = 9 (accessed by 9 method)
   μ(self.results) = 1 (accessed by 1 method)

$$LCOM^* = \frac{\left(\frac{1}{2} * \ (9+1) - 15\right)}{1-15} = 0.71$$

4. **MainController**
   a = 2
   m = 6
   μ(self.metrics) = 3 (accessed by 3 method)
   μ(self.sessions) = 3 (accessed by 3 method)

$$LCOM^* = \frac{\left(\frac{1}{2} * \ (3+3) - 6\right)}{1-6} = 0.6$$

5. **MetricsticsGUI**
   a = 8
   m = 30
   μ(self.root) = 21 (accessed by 21 method)
   μ(self.controller) = 27 (accessed by 27 method)
   μ(self.SessionName) = 7 (accessed by 7 method)
   μ(self.textInput) = 10 (accessed by 10 method)
   μ((self.MinimumCard),(self.MaximumCard),(self.MedianCard),(self.MeanCard),(self.MadCard),
   (self.tandardDeviationCard),(self.ModeCard)) = 28 (accessed by 28 method)

$$LCOM^* = \frac{\left(\frac{1}{8} * \ (21+27+7+10+28) - 30\right)}{1-30} = 0.63$$

LCOM* for some of the UI components are listed below:

6. **ScrollableLabelFrame**
   a = 3
   m = 2
   μ(self.master) = 1 (accessed by 1 method)
   μ(self.heading) = 1 (accessed by 1 method)
   μ(self.content) = 2 (accessed by 2 method)

$$LCOM^* = \frac{\left(\frac{1}{3} * \ (1+1+2) - 2\right)}{1-2} = 0.67$$

7. **InfoCard**
   a = 3
   m = 3
   μ(self.master) = 1 (accessed by 1 method)
   μ(self.number) = 2 (accessed by 1 method)
   μ(self.title) = 1 (accessed by 2 method)

$$LCOM^* = \frac{\left(\frac{1}{3} * \ (1+2+1) - 3\right)}{1-3} = 0.83$$

8. **IconButton**
   a = 3
   m = 6
   μ(self.master) = 1 (accessed by 1 method)
   μ(self.bg_color) = 5 (accessed by 1 method)
   μ(self.Input_callback) = 1 (accessed by 2 method)

$$LCOM^* = \frac{\left(\frac{1}{3} * \ (1+5+1) - 6\right)}{1-6} = 0.73$$

Qualitative conclusion(s)

High LCOM* (Indicative of Low Cohesion)

SessionInfoParser: LCOM* = 0.9
SessionManager: LCOM* = 0.83
InfoCard: LCOM* = 0.83

These classes have LCOM* values greater than or equal to 0.8, which is a strong indicator of low cohesion. Such low cohesion implies that the methods within these classes are not well related in terms of shared attributes or functionality. This scenario often leads to the "Multifaceted Abstraction", where a class may be trying to handle too many responsibilities, negatively affecting maintainability aspects like analyzability, modifiability, and testability.

Moderate LCOM* (Some Cohesion Concerns)

IconButton: LCOM* = 0.73
MetricsCalculator: LCOM* = 0.71
ScrollableFrame: LCOM* = 0.67
MetricsticsGUI: LCOM* = 0.63
MainController: LCOM* = 0.6
These classes have moderate LCOM* values, indicating some level of cohesion but also highlighting areas that could benefit from improvement. While not as critical as higher values, these classes might still possess some attributes or methods that are not closely related.

In summary, focusing on reducing the LCOM* values, particularly for those classes with values nearing or exceeding 0.8 and further imporing the classes near 0.7 LCOM* values, can lead to an overall improvement in the design quality and maintainability of the software.

7 Calculating the Physical SLOC and Logical SLOC

(a) Calculate the Physical SLOC and Logical SLOC for METRICSTICS. State your counting scheme, and show your calculations, manually or using a tool, as applicable.

(b) Comment on the qualitative conclusions that can be drawn with respect to the quantitative thresholds of the respective metrics.

7.1 LOC Metrics

LOC metrics, or Lines of Code metrics, are quantitative measures used to assess the size and complexity of a software program based on the number of lines of code written. These metrics serve as indicators of software size, productivity, and complexity. They are commonly used in software engineering to estimate effort, maintainability, and understand the scale of a codebase.

The Radon python tool was used for calculating the LOC. It analyzes the given Python modules in order to compute raw metrics which includes [23]:

(a) LOC: The total number of lines of code

(b) LLOC: The number of logical lines of code

(c) SLOC: The number of source lines of code - not necessarily corresponding to the LLOC

(d) Comments: The number of Python comment lines (includes comment on the same line as code)

(e) Single line comment (SLC): The number of Python comment lines (i.e. only single-line comments)

(f) Multi: The number of lines representing multi-line strings

(g) Blank: The number of blank lines (or whitespace-only ones)

The equation:

$$\text{LOC} = \text{SLOC} + \text{Multi} + \text{Single Comments} + \text{Blank}$$

should always hold. [27]

7.2 Calculations

Symbol	**Count**	**Definition**
Source Files	19	Source Files
LOC	1430	Lines of Code
LLOC	822	The number of logical lines of code
SLOC	929	The number of source lines of code
Comments	213	The number of Python comment lines
Single comments	176	Single-line comments
Multi	0	The number of lines representing multi-line strings
Blank	325	The number of blank lines (or whitespace-only ones)

Table 22: Total LOC

File	LOC	LLOC	SLOC	Comments	SLC	Blank
__init__.py	2	1	1	1	1	0
main.py	19	9	9	6	6	4
tests/test_main_controller.py	85	50	44	19	19	22
tests/test_metrics_calculator.py	59	47	47	7	5	7
tests/__init__.py	3	2	2	1	1	0
tests/test_session_manager.py	54	24	45	1	1	8
shared/custom_exceptions.py	2	2	2	0	0	0
controller/__init__.py	3	2	2	1	1	0
controller/main_controller_interface.py	24	18	18	1	1	5
controller/main_controller.py	34	25	27	1	1	6
model/session_info_parser.py	30	20	26	0	0	4
model/session_manager.py	121	65	78	18	13	30
model/__init__.py	9	7	7	1	1	1
model/metrics_calculator_interface.py	16	12	12	1	1	3
model/session_manager_interface.py	22	16	16	1	1	5
model/metrics_calculator.py	144	90	96	16	16	32
model/session_info_parser_interface.py	16	12	12	1	1	3
view/metricstics_GUI.py	784	418	483	136	106	195
view/__init__.py	3	2	2	1	1	0

Table 23: Total LOC

7.3 Thresholds

Considering the context of the METRICSTICS project and the need for maintaining code quality, readability, and maintainability, the established thresholds based on the Physical SLOC, Logical SLOC, comments, and blank lines are as follows:

(a) **Physical LOC Thresholds:** Considering the combined complexity of statistical calculations and GUI development, a range of 12,000 to 18,000 Physical LOC might be appropriate.

(b) **Logical LOC Thresholds:** For the logical lines excluding comments and blank lines, a range of 8,000 to 12,000 Logical LOC would align with the complexity of the statistical algorithms and Tkinter-based GUI implementation.

(c) **SLOC Thresholds:** Specifically for the code representing the functionality of statistical computations and the Tkinter GUI, a range of 10,000 to 14,000 Source LOC might be applicable.

(d) **Comments Thresholds:** Given the emphasis on readability and documentation, aiming for a range of 15% to 25% of total Physical LOC for comments, approximately 1,800 to 4,500 lines of comments could be reasonable.

(e) **Blank Lines Thresholds:** allocating around 20% to 30% of the total Physical LOC for blank lines, roughly 2,400 to 5,400 lines, can help ensure a well-structured codebase without an excessive number of unnecessary blank lines

7.4 Qualitative conclusions

The tool reported that the total Lines of Code (LOC) stands at 1430, which falls within the suggested range of 12,000 to 18,000 Physical LOC. This suggests a substantially smaller codebase than the threshold, indicating a more concise implementation.
The tool reported Logical Lines of Code (LLOC) to be 822, falling within the proposed range of 8,000 to 12,000 Logical LOC. This signifies a notably smaller and more efficient codebase in terms of executable lines, aligned with the expectations.
The Source Lines of Code (SLOC) reported by the tool was 929 which is within the range of 10,000 to 14,000 Source LOC.
The reported number of comments stands at 213 (18%). This percentage falls below the lower end of the proposed comment threshold range (15% to 25% of total Physical LOC), potentially suggesting a slightly lower emphasis on documentation or comments within the codebase.
In summary, the data indicates a compact and efficient codebase, reflecting a focused and streamlined implementation of the statistical computations and Tkinter-based GUI, with a relatively moderate emphasis on comments and documentation within the code.

8 Correlation between Logical SLOC and WMC

(a) Using Scatter Plot, carry out an analysis of the correlations between the data for Logical SLOC and WMC obtained from DESCRIPTIVE-STATISTICS.

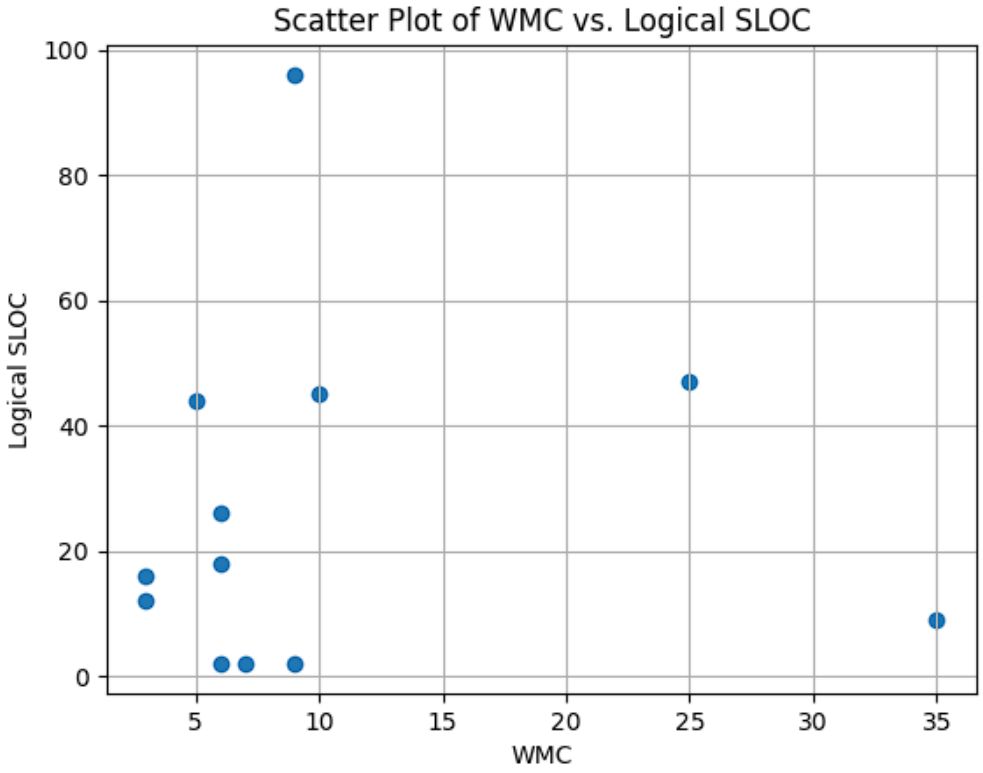
(b) Using a correlation coefficient, carry out an analysis of the correlations between the data for Logical SLOC and WMC obtained from DESCRIPTIVE-STATISTICS.

Note: It is important that the conclusions are sensible.

8.1 Scatter Plot

1. **Introduction:** The scatter plot illustrates the correlation between two essential software metrics—Weighted Methods per Class (WMC) and Source Lines of Code (SLOC). Each point on the plot represents a specific software component, with its position determined by its corresponding WMC and SLOC values.

   The scatter plot shows the relationship between Weighted Methods per Class (WMC) and Source Lines of Code (SLOC) for 17 classes in the METRICSTICS project. WMC measures the complexity of a class, while SLOC measures the amount of code in a class



2. **Axes:** The x-axis represents the WMC values, indicating the complexity and size of each software component in terms of its weighted methods. The y-axis represents the SLOC values, reflecting the size of the codebase for each component in terms of the number of source lines.

3. **Data Points:** Each data point on the plot corresponds to a unique software component, identified by its name. The horizontal position of a point signifies the WMC value for the respective component. The vertical position indicates the SLOC value for the same component.

8.2 Correlation Coefficient

1. **Spearman's Rank Correlation Coefficient:** Spearman's Rank Correlation Coefficient is a statistical measure that assesses the strength and direction of the monotonic relationship between two variables. It is based on the ranks of the data points rather than the actual values, making it robust to outliers and suitable for variables with a nonlinear association.

   Let the data in each set of Logical SLOC and WMC be ranked separately.
   Let D = rank(xi) - rank(yi).

File	SLOC (Xi)	Rank (Xi)	WMC (Yi)	Rank (Yi)	D
tests/test_metrics_calculator.py	47	4	7	3	1
tests/test_session_manager.py	45	3	10	5	-2
controller/main_controller.py	27	2	9	4	-2
model/session_info_parser.py	26	1	5	1	0
model/session_manager.py	78	5	25	6	-1
model/metrics_calculator.py	96	6	35	7	1
view/metricstics_GUI.py	483	7	6	2	5

Table 24: Coefficient Table

The Spearman's Rank Correlation Coefficient is given by:

$$r_s = 1 - \frac{6\sum D^2}{n\left(n^2 - 1\right)}$$

Thus $r_s = 0.35$, therefore we can conclude that there is a weak positive correlation between Logical SLOC and WMC.

2. **Correlation Coefficient Interpretation:** The scatter plot generated for the METRICSTICS project visually represents the relationship between Weighted Methods per Class (WMC) and Source Lines of Code (SLOC). The calculated correlation coefficient of 0.35 indicates a weak positive correlation. This means that there is a subtle tendency for an increase in WMC as SLOC increases, but the correlation is not pronounced.

3. **Potential Explanations for the Weak Positive Correlation**

   (a) **Functionality-Driven Complexity:** One plausible interpretation is that larger classes in the METRICSTICS project tend to encompass more functionality, leading to a higher WMC and, concurrently, more code (SLOC). This suggests that the project's larger classes may be designed to handle a greater level of complexity and perform more diverse tasks.

   (b) **General Trend in Software Development:** Another perspective is that the observed correlation reflects a broader trend in software development. It is not uncommon for codebases to become more complex and verbose over time, potentially contributing to the weak positive correlation between WMC and SLOC.

4. **Implications for Design and Implementation:** The weak positive correlation implies a degree of flexibility in designing and implementing classes within the METRICSTICS project. Classes can vary widely in terms of WMC and SLOC without imposing a significant impact on the overall maintainability or performance of the project.

5. **Considerations for Future Development:** While the weak positive correlation suggests flexibility in design, it also raises questions about whether certain design patterns or practices contribute to this correlation. Further analysis and exploration may be warranted to identify specific factors influencing the observed relationship.

6. **Project Maintainability and Performance:** The weak positive correlation indicates that developers working on the METRICSTICS project have the latitude to make design choices based on project-specific needs without being overly constrained by a rigid relationship between WMC and SLOC. This can positively influence the project's maintainability and performance.
   Future iterations of the METRICSTICS project may benefit from periodic reevaluation of the correlation between WMC and SLOC, especially as development practices evolve and the project's codebase continues to mature.

9 References

[1] Kamthan, Pankaj. - INTRODUCTION TO FRAMEWORKS FOR SOFTWARE MEASUREMENT. Retrieved from Concordia ENCS: http://users.encs.concordia.ca/~kamthan/courses/soen-6611/software_measurement_frameworks_introduction.pdf

[2] Lowe, S. A. (n.d.). 9 metrics that can make a difference to today's software development teams. Retrieved from TechBeacon: https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams

[3] Bottoni, Alex. (2012, December 6). What are the metrics for software documentation? Retrieved from StackExchange: https://pm.stackexchange.com/questions/8089/what-are-the-metrics-for-software-documentation

[4] Microsoft Learn. (2022, October 10). Code metrics values. Retrieved from: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022

[5] Stackify. (2017, September 16). What Are Software Metrics and How Can You Track Them? Retrieved from Stackify: https://stackify.com/track-software-metrics/

[6] G¨ohler, S. M., Eifler, T., & Howard, T. J. (2016). Robustness Metrics: Consolidating the multiple approaches to quantify Robustness. Retrieved from DTU: https://backend.orbit.dtu.dk/ws/files/125649168/Robustness_Metrics_Consolidating_the_multiple_approaches_to_%20quantify_Robustness_WebReady.pdf

[7] Try QA. (n.d.). What is Portability testing in software? Retrieved from: https://tryqa.com/what-is-portability-testing-in-software/

[8] Template Lab (n.d.). 40 Use Case Templates & Examples (Word,PDF). Retrieved from Template Lab: https://templatelab.com/use-case-templates/

[9] Paradigm, V. (n.d.). What is a Use Case Diagram? Retrieved from Visual Paradigm: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

[10] Dureja, Monika. (2014). Performance Evaluation of COCOMO Vs UCP. INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY https://www.javatpoint.com/cocomo-model

[11] Trello https://trello.com/

[12] GitHub: https://github.com/

[13] MVC Tkinter: https://www.pythontutorial.net/tkinter/tkinter-mvc/

[14] TimSort: https://en.wikipedia.org/wiki/Timsort

[15] Wikipedia Contributors, W. (n.d.). Coupling(Cyclomatic Complexity). : https://en.wikipedia.org/wiki/Cyclomatic_complexity

[16] Kamthan, Pankaj. - CONTROL FLOW STRUCTURE OF SOURCE CODE. Retrieved from section 14.1. ADVANTAGES OF THE CYCLOMATIC NUMBER : https://users.encs.concordia.ca/~kamthan/courses/soen-6611/source_code_control_flow_structure.pdf

[17] Kamthan, Pankaj. - CONTROL FLOW STRUCTURE OF SOURCE CODE. Retrieved from section 14.1.3.1. SEQUENTIAL CYCLOMATIC NUMBER : https://users.encs.concordia.ca/~kamthan/courses/soen-6611/source_code_control_flow_structure.pdf

[18] Radon Python Tool - Cyclomatic complexty: https://radon.readthedocs.io/en/latest/intro.html#cyclomatic-complexity

[19] Kamthan, Pankaj. - CONTROL FLOW STRUCTURE OF SOURCE CODE. Retrieved from section CYCLOMATIC NUMBER FROM THE PERSPECTIVE OF DECISION NODES : http://users.encs.concordia.ca/~kamthan/courses/soen-6611/software_measurement_frameworks_introduction.pdf

[20] Kamthan, Pankaj. - METRICS FOR CLASSES IN OBJECT-ORIENTED DESIGN. Retrieved from section 8. WEIGHTED METHOD PER CLASS (WMC): https://users.encs.concordia.ca/~kamthan/courses/soen-6611/ood_metrics_classes.pdf

[21] Kamthan, Pankaj. - METRICS FOR CLASSES IN OBJECT-ORIENTED DESIGN. Retrieved from section 12. COUPLING FACTOR (CF): https://users.encs.concordia.ca/~kamthan/courses/soen-6611/ood_metrics_classes.pdf

[22] Kamthan, Pankaj. - METRICS FOR CLASSES IN OBJECT-ORIENTED DESIGN. Retrieved from section 13. LACK OF COHESION IN METHODS (LCOM*) (CF): https://users.encs.concordia.ca/~kamthan/courses/soen-6611/ood_metrics_classes.pdf

[23] Radon Python Tool - RAW(LOC): https://radon.readthedocs.io/en/latest/commandline.html#the-raw-command

Contributions

Course: SOEN 6611 - Software Measurement
Team: E
Project: METRICSTICS
Professor: Pankaj Kamthan
TA: Iymen Abdella and Hamed Jafarpour
Department of Computer Science and Software Engineering
Projects GitHub Link: https://github.com/huzaifafcrit/METRICSTICS

Student ID	Name	Contribution
40242080	Mohammed Huzaifa	• Scheduled meetings • Tracked progress • Distributed tasks • Defined Goal/SMART goal • Came up with 4 questions and metrics associated with them • Created Latex document for GQM • Integrated all the parts from each team member into a Word file • Reviewed Use Cases and their descriptions • Setuped GitHub Repo • Created Graphical User Interface using tkinter • Implemented MVC Pattern • Integrated Model View and Controller • Developed custom UI components • Answered and documented question 4 - OOD • Answered and documented question 5 - Cyclomatic complexity • Answered and documented question 7 - LOC • Reviewed latex document and peers work such as WMC, CF, and LCOM*
40224578	Anagha Harinath	• Came up with 2 questions and metrics associated with them • Created Use Case Diagram Description tables • Created Use Case Descriptions page of latex document • Reviewed the GQM and Use Case Diagram • Asked questions to the TA on behalf of the team • Upload/Submit deliverable 1 on the portal on behalf of the team • Ensured all the things were included as per the emails received • Created trello board and cards • Worked on the test cases • Worked on fixing median bug for even number of data points • Performed manual testing on the GUI • Worked on Problem 3 of deliverable to calculate effort estimate using COCOMO model and comparison of UCP and COCOMO

Student ID	Name	Contribution
40204785	Sameer Kamble	• Came up with 2 questions and metrics associated with them • Create Use Case Diagram • Created Use Case Diagram page for the latex document • Reviewed the GQM and Use Case Descriptions • Implemented Basic Arithmetic Functions • Implemented Descriptive Statistics Calculations • Worked on the bug which was not refreshing the saved session • Contributed in the latex documentation • Calculated the values for WCM, CF, LCOM* • Code review with the TA
40216270	Madiah Itrat	• Came up with 2 questions and metrics associated with them • Created Cover page and index page for the latex document • Created header and footer for the latex document • Reviewed the whole solution • Increased input range to 10k random values • Created latex document • Performed UCP calculations including UUCP, TCF, ECF, PF, UAW, UUCW, estimated number of person hours • Reviewed the document • Document sections review with the TA
40230004	Srikar Hasthi	• Came up with 2 questions and metrics associated with them • Added references • Create references page for the latex document • Review the whole solution • Contributed in session storage implementation • Fixed bug where input number should not be greater than 1000 • Code Refactor of alert box • Fixed bug of invalid timestamp • Plotted a Scatter Plot showing the relation between Logical SLOC and WMC • Analyzed the correlations between the Logical SLOC and WMC using Spearman's Correlation Coefficient • Tested the Metrictics application and made a document demonstrating different scenarios

https://docs.google.com/document/d/1rRepmLwkqzTn-p66eA0qnV5Yr3qtT0x55YTpZvz9_jo/edit?usp=sharing