PROJECT

Devarapalli Naga Jyothi
030822-4864
nade22@student.bth.se

Gujjula Binathy
020404-3541
bigu22@student.bth.se

Medeshetty Nikitha
020604-8381
nime22@student.bth.se

Sri Sai Preetham Poola
010918-0133
srpl22@student.bth.se

***Abstract*—In this paper, we present our attempt to determine the modules that would be more difficult to maintain for Jabref, which is an open-source project. We have considered five releases which are stable versions of Jabref, to get more insights on the modules and evaluate their maintainability. We used the GQM framework to accomplish this, which provides a simple overview of goals, questions, and metrics. With the help of different tools like MetricReloaded and CodeMR, we collected data of different metrics and used visualisation techniques to evaluate and analyse the internal attributes.**

***Index Terms*—Cohesion, Coupling, Complexity, Understandability, Size of code, Object-oriented metrics, Maintainability.**

I. Introduction

Software Metrics are quantitative measures that are used to evaluate different aspects of the Software development process, such as software quality, performance, maintainability, reliability, and productivity.Metrics are a means for attaining more accurate estimations of project milestones,and developing a software system that contains minimal faults [1].
Metrics that measure the quality of Object-Oriented software are known as Object-Oriented Metrics. These metrics provide objective and measurable data that can be used to assess and improve software quality throughout the development lifecycle. Software metrics will reduce the subjectivity of faults during the assessment of software quality and it provides a quantitative basis for making decisions about the software quality [2]. Organisations can gain valuable insights into the quality of their software, identify areas for improvement, and make informed decisions to improve overall software quality by using appropriate software metrics. We concentrated on metrics for software development paradigms, particularly object-oriented metrics, in this report.
Many useful features of object-oriented programming include information hiding, encapsulation, inheritance, polymorphism, and dynamic binding. These object-oriented characteristics make software reuse and component-based development easier [3]. Chidamber and Kemerer proposed a suite of object oriented design metrics based on the Bunge ontology. They analysed the metrics against the principles of Weyuker's measurement theory and provided an empirical sample of these metrics from two commercial systems. Several studies have been carried out in order to validate CK metrics [4]. Lack of Cohesion in Methods (LCOM) is a cohesion metric that assesses the degree of functional relatedness among methods within a class. Higher cohesion indicates higher quality because it implies that methods within a class collaborate to achieve a common goal [5]. Coupling metrics, such as Coupling Between Objects (CBO), quantify the degree to which classes are interdependent. Lower coupling indicates higher quality because it implies loose coupling and greater modularity [1]. Depth of Inheritance Tree (DIT) is an inheritance metric that measures the depth of the class inheritance hierarchy. Excessive depth can indicate potential design flaws as well as decreased maintainability. Cyclomatic Complexity, for example, measures the complexity of control flow within methods or classes. Higher complexity values indicate code that is more difficult to understand, test, and maintain [6].
In this paper, our goal is to evaluate the maintainability of modules and identify the modules that would be more difficult to maintain for Jabref, which is an open-source bibliographic reference manager developed in Java. We have considered five stable releases which are as follows: v5.1, v5.3, v5.5, v5.7, v5.9. We selected different attributes like code size, code complexity, code structure(coupling and cohesion), understandability for better analysis and achieving the goal. To achieve the goal we had developed a "GQM" tree where we formulated a few goal oriented questions and selected relevant metrics to answer these questions. We have used MetricReloaded and CodeMR tools to extract data for these metrics.
The rest of the report is organised as follows. Section 2 is about suitable research methodology, data collection methods. In Section 3 we implemented a GQM tree, Section 4 is about selection of data analysis/statistical techniques and presenting the results. Followed by discussions and reflections on this study in Section 5.

II. Research methodology

A. Studytype and Justification

Study Type: This report presents an empirical study to find out the modules difficult to maintain for the open source system Jabref. The study type chosen is case study.
Justification: In this study we focus on a single software system Jabref where we have least control over the modules of Jabref and their attributes. This study has a low level of replication. In order to provide accurate measurements of code properties including cohesion, coupling, complexity, understandability, and size, this type of study gathers and analyzes numerical data. The results

may be applicable to other software systems because known code measures were used in the study, ensuring repeatability and generalizability.An systematic and impartial strategy for assessing maintainability, allocating modules in order of priority, and improving JabRef's overall quality and maintenance simplicity is provided via a quantitative empirical analysis.

B. Data Collection

MetricReloaded and CodeMR are the two metric tools that we used to collect the quantitative data i.e, the Metric values in order to evaluate the maintainability of the modules

MetricReloaded : MetricReloaded Plugin is an open-source plugin for IntelliJ IDEA and IntelliJ Platform IDEs that can generate automated source code metrics. The metrics CLOC, LOC, CBO, LCOM, WMC, Instability, RFC, NOC, v(G), these extracted metrics are at package level and class level, for class level metrics we took the average of all classes. The data can be extracted to either CSV forms or XML format.

CodeMR : CodeMR Plugin is an open-source software metric tool for IntelliJ for analysing and measuring different aspects of software code. It offers valuable insights into code quality, complexity, maintainability, and other relevant metrics to developers and software teams. The metrics that we had extracted using this tool are LTCC, LCOM, NOM. These metrics were extracted at package and class level, and for class level we considered average of all classes. This tool can generate visual representations such as pie charts and graphs, and metric values can be extracted in CSV or XML format

C. Data analysis

The purpose of this project is to identify modules that are difficult to maintain in Jabref. A combination of metrics reloaded and codeMR tools has been used to gather the data that the metrics need to measure the attributes.
A variety of statistical methods and data visualisation techniques have been applied to the extracted data. It was necessary to convert the class level metrics to package level metrics in order to reach the goal. Several statistical methods were used to convert the class level metrics to package level metrics like mean,median, mode and max value. Data was visualised using scatter plots and histograms. Graphs (histograms) were used to represent the data visually for all modules.

III. GQM TREE

The Goal-Question-Metric (GQM) framework is a method for defining and organising measurement activities in software engineering and other fields. It offers a systematic approach to aligning measurement objectives with specific questions and metrics [10].

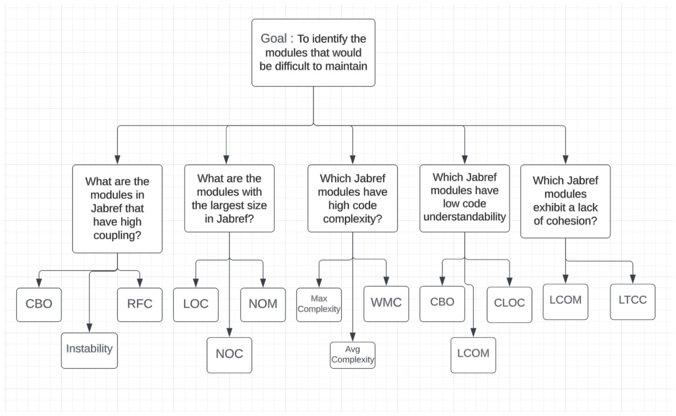
Goal : To identify the modules that would be difficult to maintain
What are the modules in Jabref that have high coupling?
What are the modules with the largest size in Jabref?
Which Jabref modules have high code complexity?
Which Jabref modules have low code understandability
Which Jabref modules exhibit a lack of cohesion?
CBO
RFC
Instability
LOC
NOM
NOC
Max Complexity
WMC
Avg Complexity
CBO
CLOC
LCOM
LCOM
LTCC

Fig. 1. GQM tree

Entity: Jabref
Internal attributes: Code structure(code coupling and code cohesion), code complexity, code size, code understandability.
External Attributes: Maintainability

TABLE I
METRICS

Metric	Description
M1	Instability
M2	Coupling between Objects (CBO)
M3	Response for class (RFC)
M4	Lines of Code (LOC)
M5	Number of Methods (NOM)
M6	Number of Children (NOC)
M7	Maximum Complexity
M8	Weighted Method Per class (WMC)
M9	Average complexity
M10	Lack of Cohesion of methods (LCOM)
M11	Commented lines of code (CLOC)
M12	Lack of tight class cohesion (LTCC)

TABLE II
Q1 GQM

Q1	What are the modules in Jabref that have high coupling?
Entity	Jabref modules
Internal attribute	Coupling
External attribute	Maintainability
Metrics	Instability, RFC, CBO

TABLE III
Q2 GQM

Q2	What are the modules with the large size in Jabref?
Entity	Jabref modules
Internal attribute	Size
External attribute	Maintainablity
Metrics	LOC, NOM, NOC

TABLE IV
Q3 GQM

Q3	Which jabref modules have high code complexity?
Entity	Jabref modules
Internal attribute	Complexity
External attribute	Maintainablity
Metrics	Maximum complexity, WMC, Average complexity

TABLE V
Q4 GQM

Q4	Which jabref modules have low code understandability?
Entity	Jabref modules
Internal attribute	Understandability
External attribute	Maintainability
Metrics	LCOM, CBO, CLOC

A. Description and justification of Metrics

1) LCOM : The degree of cohesiveness of a class is calculated. We employ a Java-specific variant of the LCOM metric developed by Hitz and Montazeri. According to the metric, two methods of the same class are related if they share a variable use or if one method calls another.

LCOM = (number of method pairs with no common instance variables in common) - (number of method pairs with common instance variables)

Justification: In the research paper "Cohesion metric for predicting maintainability of service-oriented software" M. Perepletchikov [5] , According to the findings, higher levels of cohesion have a positive impact on maintainability, indicating that services that are closely related and collaborate effectively are more likely to be maintained. The authors investigate the relationship between cohesion metrics and various maintainability factors like changeability, understandability, and complexity. The high cohesion indicates that the code is easier to understand, reuse, and analyse. Maintainability, portability, and efficiency are all hampered by high cohesion.
Scale: LCOM is a numerical value. So nominal and ordinal scales cannot be used. LCOM has absolute zero or lack of attribute. So Ratio scale is best fit to measure LCOM.

2)Coupling Between Objects(CBO) : Coupling between Objects(CBO) is an object oriented metric which measures the level of interdependence between objects. A high CBO indicates classes are more interdependent on other classes.

TABLE VI
Q5 GQM

Q5	Which jabref modules exhibit a lack of cohesion?
Entity	Jabref Modules
Internal attribute	Cohesion
External attribute	Maintainability
Metrics	LCOM, LTCC

Justification: Coupling is a metric that measures the interdependence of software components or modules, and it is critical for system maintainability. As per the research paper "An empirical study of maintainability in aspect-oriented system evolution using coupling metrics" , by H. Shen, S. Zhang, and J. Zhao [1], The authors' empirical analysis sheds light on the effect of coupling metrics on the maintainability of aspect-oriented systems. According to the findings, higher levels of coupling can have a negative impact on maintainability, implying that aspects that are tightly coupled with other system components may introduce challenges during system evolution.
Scale: CBO is a numerical value. So nominal and ordinal scales cannot be used. We use ratio scale to measure coupling between objects.

3)Response for class : Response for class metric is calculated as the number of methods called by the class or by the methods of another class. RFC says about the coupling between objects which gives insights of code structure
Justification : RFC specifies how many methods can be invoked by an object or by a method in a class in response to a message.The Research paper "Assessment of Maintainability Metrics for Object Oriented Software System" by Sanjay Kumar Dubey and Ajay Rana states that classes with high RFC values are more fault prone resulting less maintainability of system [11].
Scale : RFC is a numerical value So nominal and ordinal scales cannot be used We use ratio scale to measure response for class.

4)Number of Methods(NOM) : NOM metric is a measure of the number of methods that are present in a class or function block. Using this metric, you can determine the size and complexity of your program or module.
Justification : NOM aids in determining the size and complexity of software modules as well as the approximate amount of work needed for testing and maintenance. More methods might mean more complex code that requires more work to maintain and alter.
Scale : NOM is a numerical value. So nominal and ordinal scales cannot be used. NOM has absolute zero value so we use ratio scale to measure NOM

5) Number of Classes(NOC) : The metric NOC is the number of immediate subclasses for each class in the hierarchy [11].
Justification : Basili et al. [12] observed that the larger the NOC, the lower the probability of fault detection. They concluded that Classes with a large number of children are considered to be difficult to modify and usually require more testing, because of the effects on changes on all the children [11].
Scale : NOC is a numerical value. So nominal and ordinal scales cannot be used. NOC has absolute zero value so we use ratio scale to measure NOC.

6)Weighted method per class(WMC) : The WMC (Weighted Method Count) metric is used to determine how complex a class is. The complexity of each of the methods in a class is calculated as a whole. It measures the complexity of an individual class [13].
Justification : The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the greater the impact on children since they inherit all the methods defined in a class [11].
Scale : Weighted Methods per Class (WMC) is a numerical value. So nominal and ordinary scales cannot be used. Ratio scale is used to measure WMC.

7)Average cyclomatic complexity : The complexity of a program indicates the difficulty of understanding, analysing, and maintaining it. Average cyclomatic complexity is measured by calculating the total cyclomatic complexity of the program divided by total number of functions or methods in the program.
Justification : Average cyclomatic complexity is used to measure the complexity of a program.Maintenance tasks are more time consuming for modules with higher average cyclomatic complexity. Modules with high average cyclomatic complexity require high maintenance.
Scale : Average Complexity is a numerical value. So nominal and ordinal scales cannot be used. Average Complexity is measured using ratio scale.

8)Maximum Cyclomatic complexity : Maximum cyclomatic complexity is used to determine the complexity of the program. It is calculated by identifying the function with maximum cyclomatic complexity of the program.
Justification : It indicates the maximum cyclomatic complexity of the module or program.This metric tells us about the level of elaborateness of a code. Higher complexity indicates that the code is difficult to understand
Scale : Max Complexity is a numerical value. So nominal and ordinal scales cannot be used. Ratio scale is used to measure Max Complexity.

9)Instability : Instability of a metric tells us about the relative stability of a module. It gives us information about how a module is affected when other modules of the program are changed.

$$Instability = Ce/Ca + Ce$$

Where Ce represents efferent coupling and Ca represents afferent coupling. The instability metric value ranges from 0 to 1 where 0 indicates high stability and 1 indicates high instability.
Justification : The research paper " using indirect coupling metrics to predict package maintainability and testability " by "saleh almugrin,Waleed Albattah, Austin Melton " [14] conducted experiments and concluded that the instability metric is highly correlated with maintainability [14].Modules with low instability value indicate that it is less affected by changes which means it requires less maintenance and vice versa.
Scale : Instability is a numerical value. SO nominal and ordinal scales cannot be used. Its scale type is interval rather than ratio as in ratio scale 0 indicates absence of the attribute whereas 0 in this metric does not indicate absence of stability rather it indicates the absence of dependencies.

10) Commented lines of code(CLOC) : CLOC represents the number of commented lines in a code rather than the total line of code.comments are lines of text which acts as an explanation for the code to help developers understand the code better.
Justification : A well commented code will decrease the maintenance effort, whenever a developer tries to fix a bug or an error comments help the developer understand the code better and can provide valuable insight into the developer's thought process.if the value of CLOC is high it indicates that there is more clarity provided about the module in the code which makes it easier to understand.
Scale : CLOC is a numerical value. So nominal and ordinal scales cannot be used. Ratio scale is used to measure CLOC.

11) Lines of Code(LOC) : LOC gives us a count of the total line of code in a program or a module.It counts the total number of lines of code,blank spaces,comments.
Justification : Generally if the LOC value is high for a module then it indicates that the codebase is large which means it is difficult to understand,debug and maintain i.e, it is difficult to maintain a large codebase.
Scale : LOC is a numerical value. So nominal and ordinal scales cannot be used. LCO has absolute zero so we use ratio scale to measure LOC.

12)Lack of tight class cohesion(LTCC) : Cohesion refers to the degree to which the methods within the class are related to one another. This metric measures the degree to which members within a class have weak relationships with one another. Software systems usually use this metric to assess class cohesion
Justification : Whenever a class lacks tight cohesion then it becomes difficult to understand the interactions among them which makes it difficult to maintain.so if a module has high value for this metric then it indicates that the module is difficult to maintain.
Scale : LTCC(lack of tight class cohesion) is a numerical value. So nominal and ordinal scales cannot be used. LTCC has absolute zero so we use ratio scale to measure LTCC

IV. RESULTS

A. section 4.1

In this subsection, we will describe the eight modules present in the latest version of JabRef 5.9. The table gives a brief overview of Jabref 5.9 modules in terms of size.

TABLE VII
JABREF 5.9 VERSION

MODULE NAME	LOC	Number of classes
Styletester	563	2
Preferences	16,618	215
Logic	3,781	21
Migrations	54,399	615
Model	765	7
GUI	60,905	775
Architecture	41	5
CLI	1,123	8

TABLE VIII
DESCRIPTIVE STATISTICS FOR SIZE METRICS FOR JABREF 5.9

Descriptive statistics	LOC	Number of Classes
Mean	17274.375	206
Median	2452	14.5
Standard Deviation	23911.443	292.867

The table below is the descriptive statistics values for the Jabref version 5.9 in terms of size. From the table we can say that there is a significant gap between the mean and median which indicates that the graph of the above data is positively skewed. The ratio of standard deviation to mean is high (greater than 1) which indicates that the spread of the data is over a wide range.

The graph in Fig. 1. is a plot of size of various modules of Jabref version 5.9 in terms of LOC and Fig. 2. is a plot of size of various modules of Jabref version 5.9 in terms of Number of classes. The graphs say that there are only two modules with large sizes and all the remaining modules have a size in a similar range.

It is common for main classes to have larger code sizes than other classes because they always contain the core functionality of a module.
The main classes for each module for the most recent Jabref(5.9) version are listed in the TABLE IX.

TABLE IX
MODULE METRICS

Module Name	Main Class	Kloc
Styletester	org.jabref.styletester.StyleTesterView	35
Preferences	org.jabref.preferences.JabRefPreferences	2317
Logic	org.jabref.logic.citationkeypattern.BracketedPattern	1129
Migrations	org.jabref.migrations.PreferencesMigration	434
Model	org.jabref.model.entry.BibEntry	891
GUI	org.jabref.gui.JabRefFrame	1036
Architecture	No class	N/A
CLI	org.jabref.cli.ArgumentProcessor	570

In the TABLE X, we present the average size of Jabref modules across the five versions. In the table above, it can be seen that the average module size has increased progressively with each version

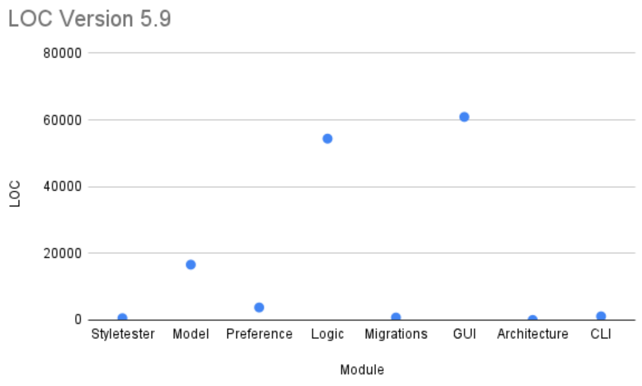

Fig. 2. LOC for different modules of Jabref version 5.9

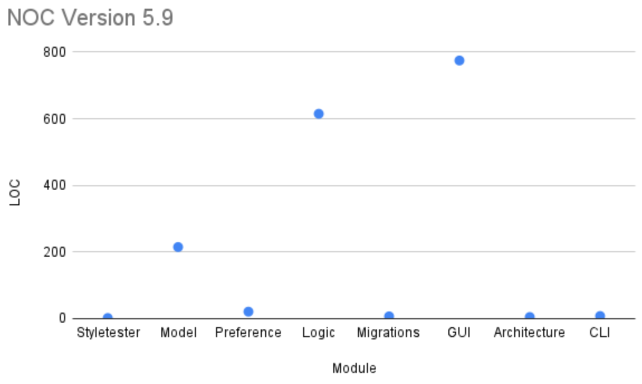

Fig. 3. NOC for different modules of Jabref version 5.9

TABLE X
CODE METRICS

Versions	LOC	NOC
Version 5.1	1796.5	7
Version 5.3	2155	14.5
Version 5.5	2317.5	13.5
Version 5.7	2353.5	13.5
Version 5.9	2452	14.5

B. section 4.2

Conversion of class level metrics to package level metric:
In this subsection, all the questions of the GQM tree were answered. Among the metrics that are used to answer the questions, there are few class level metrics. These class level metrics must be converted to package level metrics . In order to convert the class level metrics into package level metrics, the class level metrics are extracted and the mean, median, standard deviation and max values are calculated. The values which vary the most among mean, median, standard deviation and max are considered as the package level metrics. Scatter plots were plotted for mean, median standard deviation and max values of class level metric values to find the one that

varies the most.

Q1:What are the modules in Jabref that have high coupling?

To measure the coupling among the classes of Jabref three metrics were considered: **RFC, CBO and instability**.
CBO,RFC metrics are at class level and these metrics are converted into package level.
Data is collected from tools and documented in the form of tables for each metric across various modules for different versions of Jabref.
In order to find out the modules of high coupling we consider three metrics CBO, RFC and Instability. To obtain the modules with high coupling there are two levels of integration involved, one at metric level and another at package level. First we consider one version and for this version, we calculate the average values of CBO, RFC and instability.
For each metric a set of modules are considered that are greater than the average value.Now all the three sets are combined and the frequency of each module is calculated and the Three most frequent modules are considered as the modules with high coupling for version 5.1.
The above process is repeated for the remaining versions now we combine all the modules obtained from each version and the frequency of each module is calculated and the3 most frequent modules are considered as the modules of high coupling.

RFC: RFC(Response for class) metric is calculated as the number of methods called by the class or by the methods of another class. RFC says about the coupling between objects which gives insights of code structure.

TABLE XI
RFC VALUES

Module Name	Version 5.1	Version 5.3	Version 5.5	Version 5.7	Version 5.9
Styletester	5	5	5	4	4
Model	111	113	113	115	119
Preferences	192	266	236	234	226
Logic	58	58	105	95	95
Migrations	19	19	19	19	20
GUI	165	165	173	177	174
Architecture	1	1	1	1	1
CLI	55	61	63	63	63

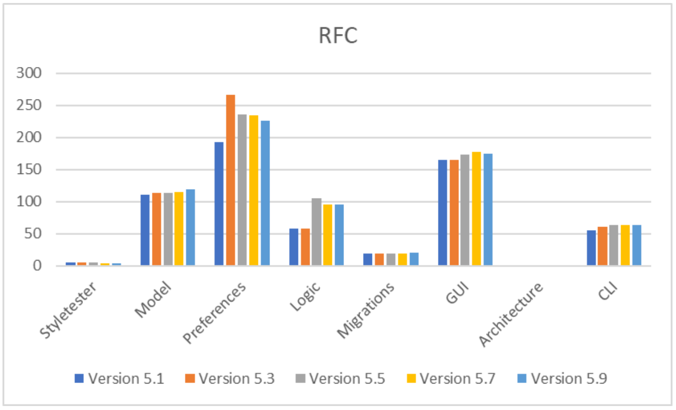

Fig. 4. RFC across differnt versions for every module

CBO : Coupling between Objects(CBO) is an object oriented metric which measures the level of interdependence between objects. A high CBO indicates classes are more interdependent on other classes.

TABLE XII
CBO VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	Version 5.9
Styletester	1	2	2	1	1
Model	69	65	75	7.6	76
Preferences	15	21	19	15	18
Logic	98	103	108	109	107
Migrations	6	7	2	2	2
GUI	111	117	122	133	136
Architecture	0	0	0	0	0
CLI	8	8	5	5	6

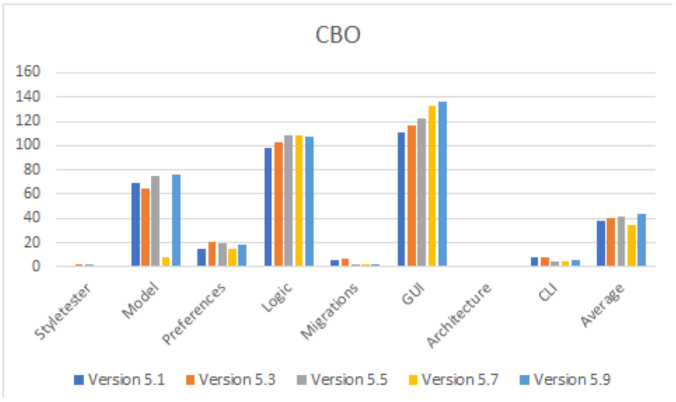

Fig. 5. CBO for different versions across all modules

Instability : Instability of a metric tells us about the relative stability of a module. It gives us information about how a module is affected when other modules of the program are changed.

$$Instability = Ce/Ca + Ce$$

TABLE XIII
INSTABILITY VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
Styletester	1	1	1	1	1
Model	1	0.5	0.5	0.5	0.5
Preferences	0.5	1	1	1	1
Logic	0.49	0.5	0.5	0.5	0.49
Migrations	0.5	1	1	1	1
GUI	0.53	0.53	0.52	0.53	0.52
Architecture	1	1	1	1	1
CLI	1	1	1	1	1

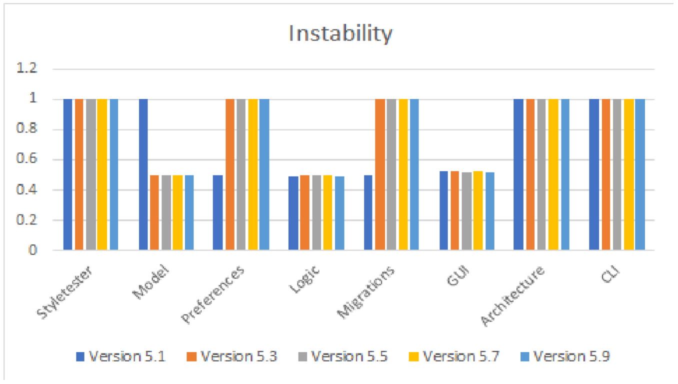

Fig. 6. Instability for different versions across all modules

The most frequently occurring modules are GUI,preference We have therefore concluded that the modules GUI,preference have high coupling.

Q2: What are the modules with the large size in Jabref?

To answer this question we have used three metrics **LOC(line of code),NOM(number of objects),NOC(number of classes)**.
All the values for these metrics have been extracted from Metrics Reloaded Plugin.
In order to find out the modules of large size we consider three metrics LOC, NOM and NOC. To obtain the modules with large size there are two levels of integration involved, one at metric level and another at package level. First we consider one version and for this version, we calculate the average values of LOC,NOM and NOM.
For each metric a set of modules are considered that are greater than the average value.Now all the three sets are combined and the frequency of each module is calculated and the three most frequent modules are considered as the modules with large size for version 5.1.
The above process is repeated for the remaining versions now we combine all the modules obtained from each version and the frequency of each module is calculated and the three most frequent modules are considered as the modules of large size.

Lines of Code(LOC): LOC(line of code) is a metric which represents the number of lines of code including comments and blank lines.

TABLE XIV
LOC VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
Styletester	561	559	563	563	563
Model	12947	13399	16199	16472	16618
Preferences	2700	3314	3572	3639	3781
Logic	41693	46184	51966	53121	54399
Migrations	634	696	696	696	765
Gui	49553	53410	56114	57685	60905
Architecture	9	25	25	33	41
Cli	893	996	1063	1068	1123

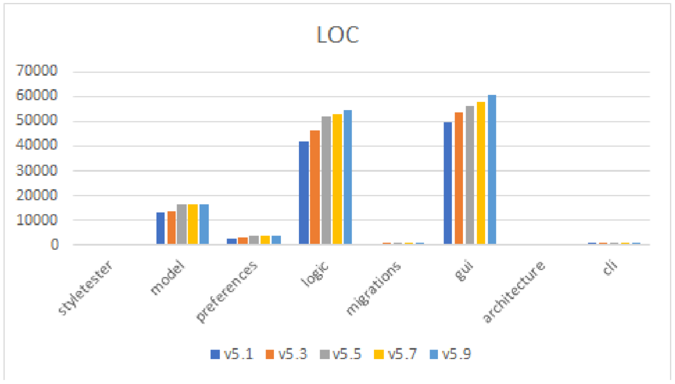

Fig. 7. LOC for different versions across all modules

Number of Methods(NOM): The number of methods or functions in a program is measured using the metric "NOM" in software engineering. It aids in determining the size and complexity of software modules as well as the approximate amount of work needed for testing and maintenance.

TABLE XV
NOM VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
styletester	6	6	6	6	6
model	1176	1250	1465	1502	1514
preferences	315	451	448	444	449
logic	2845	2863	3223	3381	3453
migrations	36	40	40	40	41
gui	3571	3789	3955	4095	4371
architecture	1	3	3	4	5
cli	66	72	74	74	77

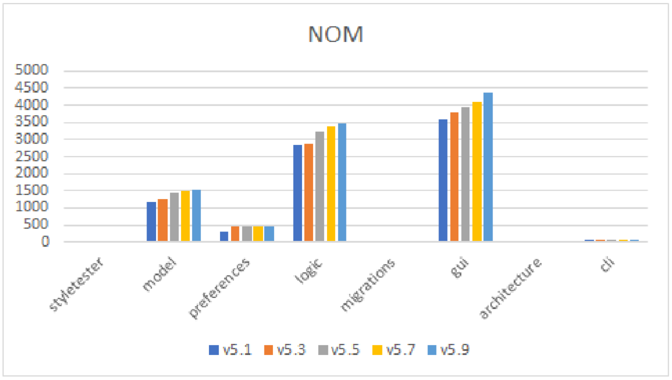

Fig. 8. NOM for different versions across all modules

Number of classes(NOC): NOC refers to the number of distinct classes present in a program.NOC varies with change in size and complexity of the program.

TABLE XVI
NOC VALUES

Module Name	Version 5.1	Version 5.3	Version 5.5	Version 5.7	Version 5.9
styletester	2	2	2	2	2
model	15	163	209	215	215
preferences	15	21	19	19	21
logic	495	556	601	612	615
migrations	6	7	7	7	7
gui	1	657	690	707	775
architecture	1	3	3	4	5
cli	8	8	8	8	8

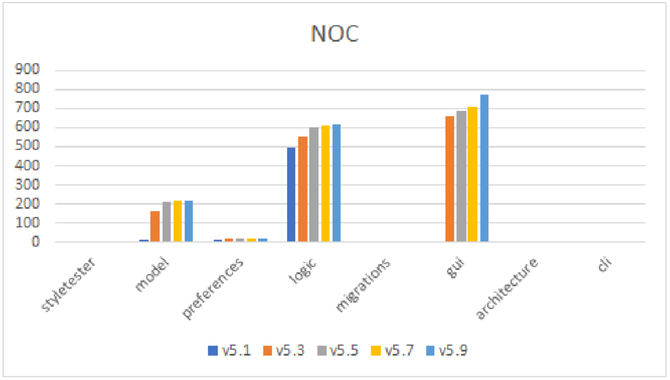

Fig. 9. NOC for different versions across all modules

The most frequently occurring modules are GUI,logic and Model.
We have therefore concluded that the modules GUI, Logic, and Model have high size.
Maintaining large software will be more difficult and cumbersome than maintaining small size software [8].As GUI,Logic,Model modules have high size we can say that these modules require high maintainability

Q3:Which jabref modules have high code complexity?

To answer this question we have used three metrics - **Max complexity, Weighted method per class, Average complexity,Max complexity,Weighted methods per class**.

Using the metrics reloaded plugin, we collected the values of these metrics. As the metrics Maximum complexity, Weighted method per class, Average complexity are class level metric, we have converted them to package level metrics.

In order to find out the modules of high complexity we consider three metrics Max complexity, Weighted method per class, Average complexity.
To obtain the modules with high complexity there are two levels of integration involved, one at metric level and another at package level. First we consider one version and for this version, we calculate the average values of Max complexity, Weighted method per class, Average complexity.

For each metric a set of modules are considered that are greater than the average value.Now all the three sets are combined and the frequency of each module is calculated and the three most frequent modules are considered as the modules with high complexity for version 5.1.
The above process is repeated for the remaining versions now we combine all the modules obtained from each version and the frequency of each module is calculated and the 3 most frequent modules are considered as the modules of high complexity.

TABLE XVII
MAX COMPLEXITY VALUES

Module Name	Version 5.1	Version 5.3	Version 5.5	Version 5.7	Version 5.9
styletester	1	1	1	1	1
model	37	22	22	22	22
preferences	8	8	8	8	8
logic	72	72	72	72	72
migrations	7	7	7	7	7
gui	96	96	96	96	96
architecture	0	0	0	0	0
cli	6	6	6	6	4

Weighted method per class(WMC): Weighted Methods per Class metric used to measure the complexity of a class or module in object-oriented programming. By examining the number of methods and their complexity weights, WMC quantifies the complexity of a class's methods. Higher the value of WMC , the more complex the module is considered to be.

Average complexity: Average Cyclomatic Complexity provides an average measure of the complexity of control flow within the software system. It can be useful in identifying modules or functions that have a higher degree of decision-making and potential complexity.

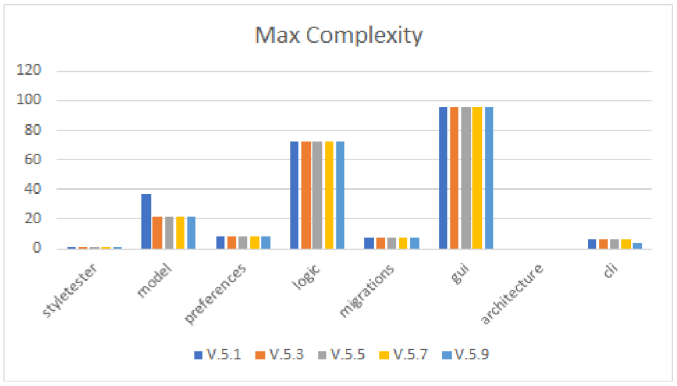

Fig. 10. Max complexity for different versions across all modules

TABLE XVIII
WMC VALUES

Module Name	Version 5.1	Version 5.3	Version 5.5	Version 5.7	Version 5.9
styletester	3	3	3	1	3
model	199.833	144	144	146	151
preferences	242	274	272	270	253
logic	214	176	179	177	177
migrations	49	49	49	49	55
gui	169	176	176	124	132
architecture	0	0	0	0	0
cli	76	95	97	97	97

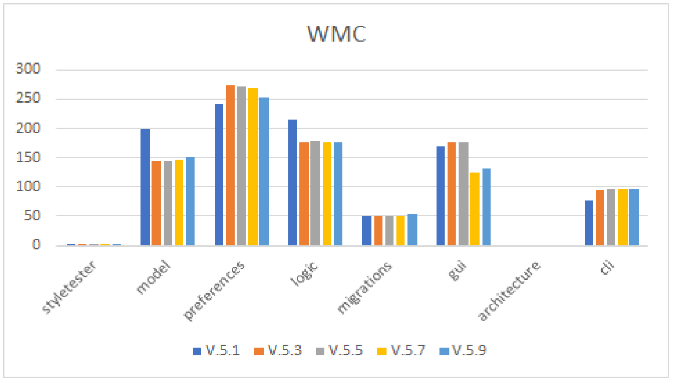

Fig. 11. WMC for different versions across all modules

TABLE XIX
AVERAGE COMPLEXITY VALUES

Module Name	Version 5.1	Version 5.3	Version 5.5	Version 5.7	Version 5.9
Styletester	1	1	1	1	1
Model	14.4	4.5	5.5	5.5	5.5
Preference	4	4	1.72	1.78	1.85
Logic	25	25	25	25	25
Migrations	4	4	4	4	4
GUI	49	49	49	49	49
Architecture	0	1	0	0	0
CLI	3.62	3.8	0	3.88	0

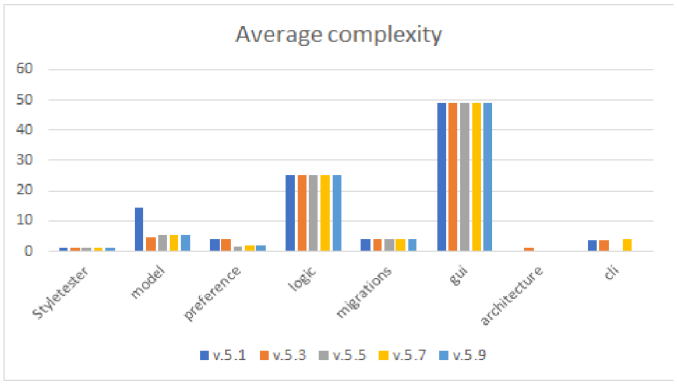

Fig. 12. Average complexity for different versions across all modules

The most frequently occurring modules are GUI,logic.
We have therefore concluded that the modules GUI, Logic have high complexity.
GUI.LOGIC modules have more complexity. The research paper [7] says that,the results of our findings show that there is a direct relationship between software complexity and maintenance costs. That is, as lines of code increase, the software becomes more complex and more bugs may be introduced, and hence the cost of maintaining such software increases [7]. Hence we can say that GUI,LOGIC modules are difficult to maintain.

Q4: Which jabref modules have low code understandability?

To answer this question we have used **LCOM,CBO,CLOC** metrics.
Using the metrics reloaded plugin, we collected the values of these metrics. As LCOM and CBO are class level metrics , we have converted them to package level metrics.
In order to find out the modules of high coupling we consider three metrics LCOM , CBO , CLOC. To obtain the modules with low understandability there are two levels of integration involved, one at metric level and another at package level. First we consider one version and for this version, we calculate the average values of LCOM , CBO , CLOC. For each metric a set of modules are considered that are greater than the average value for LCOM and CBO and values less than the average for CLOC. Now all the three sets are combined and the frequency of each module is calculated and the three most frequent modules are considered as the module with low understandability for version 5.1.
The above process is repeated for the remaining versions now we combine all the modules obtained from each version and the frequency of each module is calculated and the three most frequent modules are considered as the modules of low understandability.

LCOM: LCOM (Lack of Cohesion of Methods) is a metric used to measure the cohesion within a class or module. It is a measure of the level of interconnection between methods within a class. The higher the LCOM, the more difficult it will be to understand a class's purpose and behaviour.

TABLE XX
LCOM VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
Styletester	3	3	3	1	1
Model	30	31	31	32	32
Preference	8	9	9	11	11
Logic	9	10	11	11	11
Migrations	2	3	3	3	3
GUI	15	15	15	15	15
Architecture	0	0	0	0	0
CLI	5	5	5	5	5

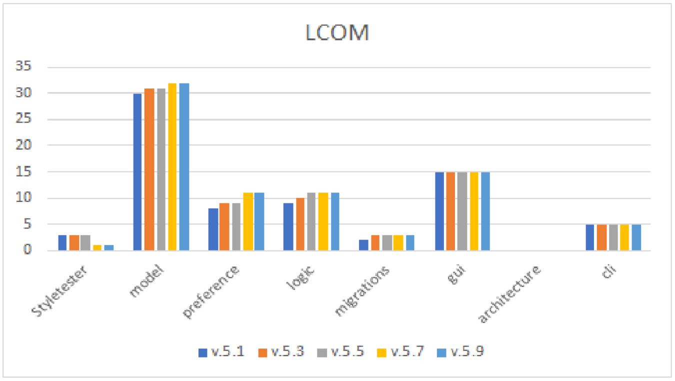

Fig. 13. LCOM for different versions across all modules

CBO: CBO measures the degree of coupling between classes or objects within a software system. An object's direct coupling to other classes or objects is quantified. High value of CBO indicates high coupling between classes or modules which means increase in complexity, context switching , difficulty to maintain and to understand.

TABLE XXI
CBO VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	Version 5.9
Styletester	1	2	2	1	1
Model	69	65	75	7.6	76
Preferences	15	21	19	15	18
Logic	98	103	108	109	107
Migrations	6	7	2	2	2
GUI	111	117	122	133	136
Architecture	0	0	0	0	0
CLI	8	8	5	5	6
Average	38.5	40.375	41.625	34.075	43.65

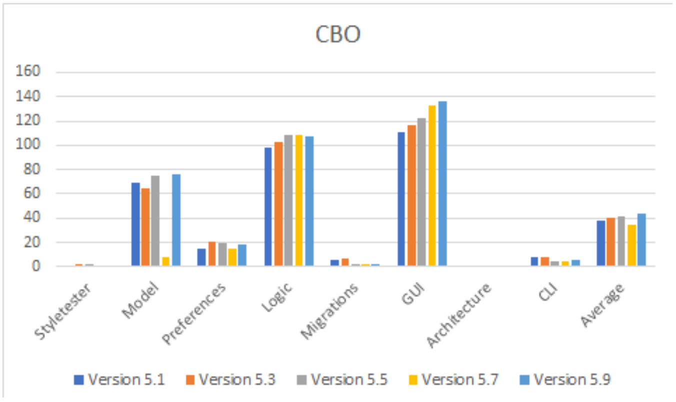

Fig. 14. CBO for different versions across all modules

CLOC: CLOC (Commented Lines of Code) is a software metric that simply measures the number of lines of code in a software system. It is a straightforward and commonly used metric to gauge the size and complexity of a codebase. This metric provides indication of overall volume of the code.

TABLE XXII
CLOC VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	Version 5.9
Styletester	0	0	3	0	0
Model	0	3171	5750	0	0
Preference	0	0	307	0	0
Logic	0	0	10394	0	0
Migrations	0	0	96	0	0
GUI	0	0	35750	0	0
Architecture	0	0	0	0	0
CLI	0	0	50	0	0

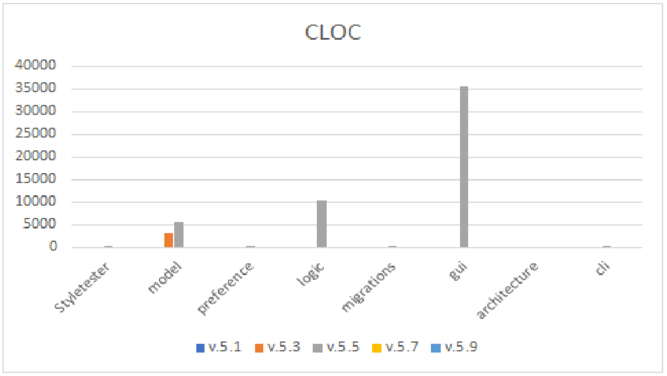

Fig. 15. CLOC for different versions across all modules

The most frequently occurring modules are GUI,logic. GUI and LOGIC are the two modules that have least understandability.

Q5: which jabref modules exhibit a lack of cohesion?

To answer this question we have used three metrics- **LCOM(Lack of cohesion of methods),LTCC(number of classes with high lack of tight cohesion)**.

All the values for these metrics have been extracted from Metrics Reloaded Plugin. LCOM and LTCC metrics are at class level and we have converted them into package level metrics.
In order to find out the modules of low cohesion we consider three metrics LCOM,LTCC. To obtain the modules with low cohesion there are two levels of integration involved, one at metric level and another at package level. First we consider one version and for this version, we calculate the average values of LCOM,LTCC.
For each metric a set of modules are considered that are greater than the average value.now all the three sets are combined and the frequency of each module is calculated and the three most frequent modules are considered as the module with low cohesion for version 5.1.
The above process is repeated for the remaining versions now we combine all the modules obtained from each version and the frequency of each module is calculated and the three most frequent modules are considered as the modules of low cohesion.

Lack of cohesion of methods(LCOM): LCOM (Lack of Cohesion of Methods) is a metric used to measure the cohesion within a class or module. It is a measure of the level of interconnection between methods within a class. The higher the LCOM, the more difficult it will be to understand a class's purpose and behaviour.

TABLE XXIII
LCOM VALUES

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
Styletester	3	3	3	1	1
Model	30	31	31	32	32
Preference	8	9	9	11	11
Logic	9	10	11	11	11
Migrations	2	3	3	3	3
GUI	15	15	15	15	15
Architecture	0	0	0	0	0
CLI	5	5	5	5	5

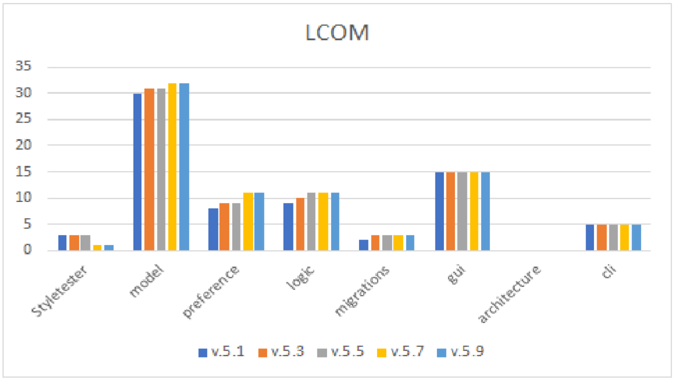

Fig. 16. LCOM for different versions across all modules

Lack of tight class cohesion(LTCC): Cohesion refers to the degree to which the methods within the class are related to one another.This metric measures the degree to which members within a class have weak relationships with one another.

TABLE XXIV
MODULE DATA

Module Name	**Version 5.1**	**Version 5.3**	**Version 5.5**	**Version 5.7**	**Version 5.9**
Styletester	3	3	3	1	3
Model	30	30	30	32	30
Preference	8	8	8	11	8
Logic	9	9	9	11	9
Migrations	2	2	2	3	2
GUI	15	15	15	15	15
Architecture	0	0	0	0	0
CLI	5	5	5	5	5

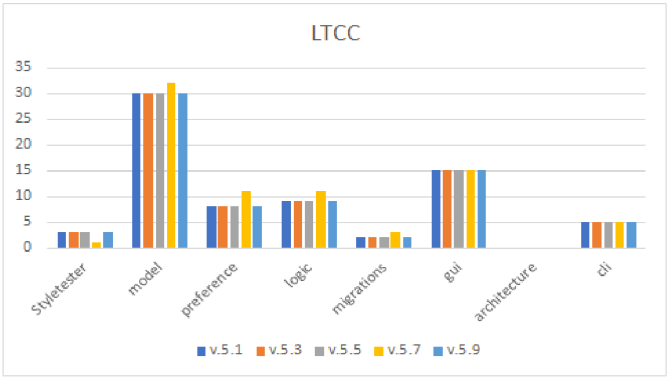

Fig. 17. LTCC for different versions across all modules

The most frequently occurring modules are GUI,Model. We have therefore concluded that the modules GUI, Model have low cohesion.

Cohesion is highly desired by the developers because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand[9].As GUI,Model has low cohesion we can say that GUI,Model are difficult to maintain.

C. Section 4.3

This study follows a GQM based approach to find the modules that are difficult to maintain in Jabref. In this approach we try to explore a relationship between the internal attributes of the code(code understandability, code complexity and code structure) and the maintainability of the code. This is done in Section 4.2 by finding the modules of high coupling, modules with low cohesion, modules of large size, modules with high complexity and modules with low code understandability.

In the previous section, the following questions were answered:
1. What are the modules in Jabref that have high coupling?
2. What are the modules with the large size in Jabref?
3. Which jabref modules have high code complexity?
4. Which jabref modules have low code understandability
5. Which jabref modules exhibit a lack of cohesion?

In order to determine modules with high maintainability, all the modules are aggregated from the above questions that has high coupling,high complexity,low cohesion,low understandability and the frequency of each module is calculated. Modules with high frequency are considered as modules that require high maintenance.
GUI, Logic are the modules of Jabref which require high maintenance.
GQM tree,data collection, data analysis and visualization for all the questions is performed in the excel sheet the link is provided below.
Data Collection and Analysis-Excel sheet.

V. Discussion

A. Section 5.1

The main focus of this project is to identify the modules of Jabref that require high maintenance. We have considered eight modules: Styletester, Model, Logic, Architecture, GUI, Preferences, Migrations, Cli. We have focused on 5 different versions v5.1,v5.3,v5.5,v5.7,5.9.
To measure the maintainability, we have considered the following attributes: size, complexity, cohesion, coupling, understandability.
We found a relation between size and complexity, size and understandability.
By measuring the LOC, NOM, NOC We found that GUI, Logic, Model modules have high size and by measuring the Max complexity,Weighted methods per class,average complexity. The research paper[7] says that, there is a direct relationship between software complexity and maintenance. That is, as lines of code increase, the software becomes more complex and more bugs may be introduced, and hence the cost of maintaining such software increases[7].
From this, we found that GUI,Logic have high complexity. Similarly by measuring CLOC, CBO, LCOM we found that GUI,Logic modules have high understandability. From this we can say that as size increases understandability will increase.

B. Section 5.2

This project's main focus is to identify the Jabref modules which are difficult to maintain. five different versions of Jabref are considered and it contains eight different modules. According to our observations, module sizes, complexity, and coupling vary from one version to another.
All the metrics data is extracted from code MR and metrics Reloaded. We have observed that the size and complexity metrics are interrelated.
We found that GUI and Logic are significant modules
From this project we have learnt how to practically apply a GQM tree and how to extract relevant data and measure the attributes.
We learnt how to analyse the extracted data and reach the goal.This project has given us a deeper level of understanding about object oriented metrics and about which metric is used to measure a specific attribute.
The GQM framework consists of five steps; the steps that we found most challenging to implement are step 2 and step 5 i.e, specifying the relevant metrics needed to answer the questions and validating and analysing the data.
We gained a clear knowledge of the GQM Tree, what kinds of questions should be included, and what kinds of metrics should be utilised to measure the attributes in order to answer those questions as a result of the group work that was conducted on April 6.

References

[1] Boehm, B. W., "Improving Software Productivity," IEEE Computer , pp. 43-57, September 1987.

[2] Belachew, Ermiyas Birihanu. "Analysis of Software Quality Using Software Metrics." International Journal on Computational Science & Applications 8.4/5 (2018): 11–20. Web.

[3] Mei-Huei Tang, Ming-Hung Kao and Mei-Hwa Chen, "An empirical study on object-oriented metrics," Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), Boca Raton, FL, USA, 1999, pp. 242-249, doi: 10.1109/METRIC.1999.809745.

[4] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," in IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994, doi: 10.1109/32.295895.

[5] M. Perepletchikov, C. Ryan, and K. Frampton, "Cohesion metrics for predicting maintainability of service-oriented software," in Seventh International Conference on Quality Software (QSIC 2007). IEEE, 2007, pp. 328–335.

[6] H. Liu, X. Gong, L. Liao and B. Li, "Evaluate How Cyclomatic Complexity Changes in the Context of Software Evolution," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 2018, pp. 756-761, doi: 10.1109/COMPSAC.2018.10332.

[7] Ogheneovo, E. (2014) On the Relationship between Software Complexity and Maintenance Costs. Journal of Computer and Communications, 2, 1-16. doi: 10.4236/jcc.2014.214001.

[8] Ogheneovo, E.E. (2013) Software Maintenance and Evolution: The Implication for Software Development. West Africa Journal of Industrial and Academic Research, 7, 34-42.

[9] Sharma, Shweta, and S. Srinivasan. ”A review of Coupling and Cohesion metrics in Object Oriented Environment.” International Journal of Computer Science Engineering Technology (IJCSET) 4.8 (2013): 1105-1111.

[10] ”The Goal Question Metric Approach” by Victor R. Basili and H. Dieter Rombach (1988).

[11] Sanjay Kumar Dubey and Ajay Rana. 2011. Assessment of maintainability metrics for object-oriented software system. SIGSOFT Softw. Eng. Notes 36, 5 (September 2011), 1–7. https://doi.org/10.1145/2020976.2020983

[12] Basili, V.R., L. C. Briand and W. L Melo. (1996): A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE Transactions on Software Engineering, 22(10), 1996, pp. 751-761.

[13] Chidamber, S. R. and C. F. Kemerer (1994): A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20(6), 1994, pp. 476–493.

[14] Saleh Almugrin, Waleed Albattah, Austin Melton, Using indirect coupling metrics to predict package maintainability and testability, Journal of Systems and Software, Volume 121, 2016, Pages 298-310, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2016.02.024.