A CASE STUDY ON MAINTAINABILITY OF OPEN SOURCE SOFTWARE SYSTEM JABREF

Denim Deshmukh, Ravi Theja Kataray and Tallari Rohith Girikshith

Blekinge Tekniska Högskola, Sweden

ABSTRACT

Maintainability is a major aspect of any software project; maintainability refers to the ease by which software can adapt to changes. There are various factors that affect the effort required for maintenance, in this paper we conducted a study to observe the extent up to which a metric could affect the maintainability of a software. We have considered various versions of JabRef and studied how maintainability of various packages had changed across versions. This is done using a framework called Goal Question Metric(GQM) which provides a systematic procedure to study various attributes of entities. Data of the attributes are collected using various Object-Oriented code metric tools which provide numerical data to compare the attributes between the versions. The data collected is visualized to answer the questions formulated which indeed tends to achieve the goal to identify the modules that are hard to maintain.

KEYWORDS

Size, Structure, Complexity, Maintainability, Understandability & Goal Question Metric (GQM) approach.

1. INTRODUCTION

This Software quality plays an important role while it comes to the development of the software product as it provides a brief ideology about the software product for the developers or the end users for further changes or modification that would be needed as the technology advances rapidly in the future. Hence, assessing the software quality is done using the measurement units like code size, coupling, maintainability, cohesion and structure of the product. Since software maintenance is critical, prospective developers should consider the maintainability aspect as a high priority during software system development [4]. In this study we have observed various metrics that assess various attributes over the Object-Oriented (OO) system JabRef. The software maintainability being an external attribute and is complicated and vital to calculate to know as it is affected by various factors. Maintainability is an important quality attribute, but it is difficult to estimate, because it involves making predictions about future changes that will occur in a software module, once it has been deployed [3]. As the Object-oriented based software systems are used more widely and often in all the software systems, they are different from the non-OO system due some programming concepts like inheritance, encapsulation. Hence, we cannot apply the well-known software metrics used to predict the non-OO software [3]. Many various studies have been done to provide the OO metrics that are reliable and effective to measure the maintainability of the product. In this paper a detailed analysis of various versions on an object-oriented project namely JabRef is conducted to assess the packages which are high to maintain.

Dhinaharan Nagamalai et al. (Eds): CSEIT, WiMoNe, NCS, CIoT, CMLA, DMSE, NLPD - 2020
pp. 255-273, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101121

1.1. Summery

This paper is divided into various sections Firstly analysis will be carried out by a Goal Question Metric (GQM) framework in which Goals to be achieved will be mentioned and various questions will be formulated questioning about the attributes of the entities and the metrics which will be best suited to answer the question will be mentioned, suitable scales for measuring metrics will be selected, tools which support object oriented metrics will be used to collect data of various versions, visualization methods like bar graphs , flow charts, tables will be uses to analyse the collected data. Later results of the analysis will be discussed along with the change log and related or similar work done on analysis of maintainability on various versions of JabRef.

2. RESEARCH METHODOLOGY

2.1. Study Type

Being an empirical study, Case study would be the suitable methodology. Case study is an empirical method aimed at investigating contemporary phenomena in their context [1]. This study is flexible in nature and could be conducted in real life context which makes it suitable for case study. The objective of this study is to explore for the least maintainable package which requires more effort and focus when compared to others. Such an objective can be achieved by analysing the various factors and metrics in a software.

2.2. System Considered for the Study

In this study we have considered an open source software called JabRef a cross-platform citation and reference management tool. The study is done on 10 recent/stable releases which are 5.0, 4.3, 4.2, 4.1, 4.0, 3.8, 3.7, 3.6, 3.5, and 3.4 versions. JabRef is developed in an object-oriented approach using Java Language. Source code, previous releases and related data was collected from its GitHub repository [7].

2.3. Study Design

Design for the case study and analyses in phases as following steps:

- Define a Goal Question Metric based on the template.
- Define Study type and provide justification for the specific study type.
- Define metrics to measure and justify the use of specific metrics.
- Define Question, Entities, attributes and relevant metric to give answer and result.
- Collect the data with the tool and analyse the data for results.
- Answer the question and provide analysis and overall evaluation.
- Discuss the reflection on the project and related works.

2.4. Data Collection

2.4.1. Quantitative Data

The qualitative data is collected by using three metrics extraction tools which are metrics reloaded, lizard and maintainability index.

Metrics Reloaded: Metrics Reloaded is a Plugin to eclipse IDE and able to extract various metrics such as Ca, Ce, CBO, LOC, LCOM, DIT and WMC form Java source code. This tool can extract

metrics at package level and support various suits such as Martin Suit and CK suit etc. This tool can extract various metrics and have an option to extract metrics in CSV (Comma-Separated Values) or XML format.

Lizard: Lizard is a metric extraction tool written in python. This tool is an open source tool and available in form of a python library but could extract metrics from the source code written in various programming languages. It is used to extract metrics such as Cyclomatic Complexity, Average cyclomatic complexity and Line of Code etc. This tool can be used at package, project as well as class level. It is a command line tool and supports CSV, text file-based output.

Maintainability Index: It is an extension of the lizard tool used to extract Maintainability Index at package, project and class level and support CSV output like the Lizard tool.

2.4.2. Quantitative Data

Change logs are analysed to find major changes in the project and to relate the report [14] with the data extraction.

3. GQM TREE

Goal-Question-Metric (GQM) approach to process metrics provides a framework for deriving measures from organization or business goals. According to Basili's GQM process, the first phase is to develop project goals [5][6].Then in the second phase we develop questions that define the goals as completely as possible in a quantifiable way. Then in the third phase we specify the metrics to be collected to answer the questions we defined. In the fourth phase we define data collection and in the last phase we collect data, validate it and provide feedback for corrective measure [5][6].

Table1 GQM Tree

Goal	
Goal 1: To measure maintainability of various packages.	
Q1	How maintainable the packages are with respect to size and structure?
Q2	How does complexity of the packages change the maintainability of the product?
Q3	To what extent does the understandability of each package provide the ease of maintenance of the product?
Q4	How does the cohesive nature of packages influence the maintainability of the product?
Q5	How does the maintainability change with instability in terms of coupling?

Table2 Metrics Table

The metrics that are considered in this study for the evaluation of the JabRef systems are as follows:**S.no**	**Metrics**	**Metrics Full Name**
M1	DIT	Depth of Inheritance Tree
M2	LOC	Line of Code
M3	LCOM	Lack of Cohesion of Method
M4	v(G)	Cyclomatic complexity
M5	Ca	Afferent Coupling
M6	Ce	Efferent coupling
M7	MI	Maintainability Index
M8	CR	Comment Ratio
M9	DIT	Coupling between object classes
M10	LOC	Weighted methods per class

Justification

1) DIT: This metric gives information about the inheritance within a class by measuring the number of nodes between the root node and given node within a class hierarchy, which indirectly describes the structure of the code. This metric was obtained in class level which was aggregated to package level by taking the mean values of all the classes in a specific package.

2) LOC: This is the simplest and most reliable metric to measure the size of a package.

3) LCOM: This metric measures the cohesion between methods of a class. Cohesion is the interdependence of methods in a class, which may affect the maintainability of a package and describes the structure of Source code. Hence this metric is considered to measure size and structure of a package. This metric was obtained in method level which was aggregated to package level by taking the mean values of all the methods of a class in a specific package.

4) Cyclomatic complexity v(G): This metric is essentially used to measure and denote the complexity of code. It measures the number of linearly independent paths in the program. It is also a basis for calculating the maintainability index. This metric can be obtained on class level and averaged on package level.

5) Ca: This metric measures the number of classes in other packages that are dependent on the classes in the package. This metric was obtained on package level. This metric is obtained on package level and shows the dependency of other packages on the package in scope. It is also known as incoming dependency. Afferent coupling will have a significant effect on maintainability.

6) Ce: This metric measures the dependency of the class in scope on the other classes of the package. This metric is obtained on package level and shows the dependency on other

packages in scope. It is also known as outgoing dependency. Efferent coupling has a significant effect on maintainability.

7) MI: This metric measures the maintainability and give value between zero to hundred, zero means very hard to maintain and hundred means easy to maintain. This metric measure maintainability using Cyclomatic complexity, Line of code and Halstead Volume. This provide an insight to the ease of maintenance with respect to complexity

8) CR: This metric defines the ratio of commented lines to lines of code of that particular package which helps to know exactly how many lines of code (LOC) are exactly available per package hence would be useful while estimating the efforts required during the maintenance of that particular package.

9) CBO: Also referred to as coupling between objects usually defines the coupling between the classes for each class to which they are coupled with. This indeed helps us to know the coupling between the packages and helps further while the product is modified or analysed for maintenance.

10) WMC: Also referred to as a weighted method per class usually describes the number of methods present per class which helps to know exactly how much effort will be required to maintain that particular class in maintenance phase which indeed helps to prevent excess cost and time.

3.1. Entities, Attribute and the Metrics

Table 3: Entities, Attribute and Metrics of Question Table

Q1: How maintainable the packages are with respect to size and structure?	
Entity	Package
Attribute1	Size
Attribute2	Structure
Attribute Type	Internal
Metric1	LOC
Metrics2	LCOM, DIT, CBO

Q2: How does complexity of the packages change the maintainability of the product?	
Entity	Package
Attribute	Complexity
Attribute Type	Internal
Metric	CBO, WMC, v(G), DIT

Q3: To what extent does the understandability of each package provide the ease of maintenance of the product?	
Entity	Package
Attribute	Understandability
Attribute Type	Internal
Metric	Ca, Ce, CR, WMC

Q4: How does cohesive nature of packages influence the maintainability of the product?	
Entity	Package
Attribute	Cohesion
attribute Type	Internal
Metric	LCOM

Q5: How does the maintainability change with instability in terms of coupling?	
Entity	Package
Attribute	Coupling
Attribute Type	Internal
Metric	Ca, Ce

Justification: Supportive aspects like why the certain metrics are selected for that internal attribute.

1) Understandability: It is an important attribute in software development process as it plays a vital role in maintaining the software product as understanding the models can help in modifying, analysing the system for later requirements and advancements which indeed save a lot of cost and time for avoiding the implementation errors[19]. Hence understandability is affected with various factors hence out of those many factors Comment ratio (CR), efferent coupling (Ce)and afferent coupling (Ca) are considered [16] in this study.

2) Size and Structure: Size of a package could be measured using lines of code(LOC) and number of functional points(FP), but by using these metrics one cannot draw any conclusion regarding the maintainability of the packages as there are other metrics related to the structure of the code which should be considered to provide a better picture on the overall maintenance of the packages. CK metric suite [13] provides various metrics which could depict the structure of the code like Depth Inheritance Tree(DIT)[11][12], Lack of Cohesion between Methods (LCOM)[10].

3) Complexity: Maintainability is highly dependent on the complexity of the software. Complexity is one of the most important factors affecting the overall maintainability of a software. Software complexity is described as the degree of difficulty in analysing, maintaining and modifying software[3].The cyclomatic complexity is a measurement of independent paths in a program and Average cyclomatic complexity for a package means that every class on average in a program has that much cyclomatic complexity. Similarly, the Weighted method per class provides us with an estimate for complexity for methods [4]. Coupling between object classes is the count of other classes being used which provide help in visualizing dependencies. The depth of inheritance provides a visualization ability to understand the abstraction level in the program and give a prospective of depth of abstraction in the program. Hence v(G), DIT, CBO and WMC are used to calculate complexity.

3.2. Scale Type

This section briefly discusses the scale type and its utilization in this study and in-detail justification is provided for selecting the scale type for the suitable metrics. Two different types of scales were considered in this paper.

Ratio: As this scale is the best suitable for the ratio values and the measurement of the metrics like CBO and comment ratio are taken under this scale type as their measurement values start from 0(zero) and increase gradually at equal intervals.

Absolute: This scale is usually associated with the number of the entity which is in the scope to measure. Hence, the measurement of the metrics like WMC, Ca and Ce are taken under this scale type as their measurement values are exact counted entities.

Justification

CBO: The best scale for measuring CBO is the absolute scale as it is the measure of the number of classes coupled to a class.

CR: The ratio scale is selected to represent the comment ratio as this is the best suitable scale to represent the ratio values better than any other scale types.

WMC: Ratio scale is taken into consideration for the weighted method per class as it is the best suitable for the arithmetic analysis and most of the measurement values have been ratio of complexity and number of elements which are indeed taken for ratio scale type.

DIT: According to the definition of DIT it is the count of number of nodes between root node to leaf node which could be measured using absolute scale.

LOC: Absolute scale is the best suitable scale to measure LOC as an exact count of the source line of code can be calculated at package and class level.

LCOM: Ratio scale is the best suitable scale to measure this metric as representing a link between methods and local variables is done for more than one class.

v(G): Cyclomatic complexity measures the number of linearly independent paths in a program. For calculating absolute values and representing on a scale Absolute scale is best to be used in this case.

Ca: The best scale fit for Afferent coupling is absolute since it represents the absolute numerical value of the number of packages on whom the package in scope is dependent.

Ce: The best scale fit for Efferent coupling is absolute since it represents the absolute numerical value of the number of packages who are dependent on the package in scope.

4. RESULT

The Study was formed according to the Goal Question Metrics tree and analysed to find the answer to the question for the goal. All 10 versions of JabRef were analysed and compared to form the results. An overview of JabRefprojects according to size with respect to number of classes and Line of code for every version is given in the table below.

Table 4. Overview Table

Version	LOC	Number of Children(NOC)
3.4	101246	183
3.5	102112	184
3.6	106703	239
3.7	113227	245
3.8	115359	250
4.0	123062	334
4.1	125653	340
4.2	126238	345
4.3	126563	373
4.5	119085	331

Based on the goal and analysing the software there were 10 main packages or called modules were found in the main directory and the analysis is done on these packages. These packages were significant and persistent throughout all the versions.

- net.sf.jabref.cli as cli
- net.sf.jabref.logic as logic
- net.sf.jabref.migration as migration
- net.sf.jabref.model as model
- net.sf.jabref.preferences as preferences
- net.sf.jabref.pdfimport as pdfimport
- net.sf.jabref.gui as gui
- net.sf.jabref.collab as collab
- net.sf.jabref.shared as shared

Some packages were there in the project which were either very small or merged in some other packages. There packages are listed below

- net.sf.jabref.architecture as architecture
- net.sf.jabref.external as external
- net.sf.jabref.sql as sql
- net.sf.jabref.util as util
- net.sf.jabref.exporter as exporter
- net.sf.jabref.specialfields as specialfields
- net.sf.jabref.bst as bst

4.1. Analysis of Size and Structure

In this section Q1 of GQM is answered providing a detailed analysis on the effect of size and structure on the maintainability of the product using DIT, LOC, LCOM metrics. The data is collected using Metrics Reloaded plugin installed to IntelliJ IDEA

DIT metric values obtained are depicted below in a tabular form:

Table 5. DIT table

DIT										
	V 3.4	**V3.5**	**V3.6**	**V 3.7**	**V 3.8**	**V 4**	**V 4.1**	**V 4.2**	**V 4.3**	**V 5**
Architecture						0	0	0	0	0
Cli	1	1	1	1	1	1	1	1	1	1
Logic	1.1	1.1	1.11	1.15	1.14	1.15	1.15	1.16	1.16	1.15
Migration	1	1	1	1	1	1	1	1	1	1
Model	1.13	1.12		1.16	1.17	1.14	1.14	1.16	1.15	1.17
Preferences			1.43	1.3	1.3	1.27	1.27	1.27	1.27	
Styletester										1
Pdfimport	2.25	2.25	2.25	2.25	2.25	1	1	1	1	
Gui	2.56	2.57	2.48	2.46	2.47	2.12	2.11	2.08	2.08	1.29
Collab	1.58	1.58	1.58	1.61	1.61	1.33				
Shared			1.46	1.53	1.53	1.53	1.53			

Maintenance of a product or code is directly proportional to DIT value i.e., maintainability decreases with increase in DIT value from our observations package “model” has a gradually increasing DIT value and could be classified as high to maintain. “GUI” package has gradually decreasing values over versions, which indicates low maintenance and frequently updated versions. “pdfimport” package had a major update from V3.8 to V4.0 decreasing DIT values indicates low maintenance of the package. Rest of the packages are classified into moderate level maintenance and low maintenance categories depending on their mean values across the versions which is discussed in later sections of the paper.

Maintenance is directly proportional to the product size in terms of Line of Code(LOC). The metric value obtained are depicted below in tabular form:

Table 6. LOC table

LOC										
VERSION	**V 3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V 4**	**V4.1**	**V4.2**	**V 4.3**	**V 5**
Architecture						9	9	9	9	9
Cli	714	727	1050	1060	1062	1073	1073	946	946	885
Logic	25861	25717	35308	34295	34625	36205	37066	39369	39429	40609
Migration	324	324	344	343	412	488	530	650	669	564
Model	5463	5567	6108	11946	11749	12374	12346	12665	12678	12843
Preferences			1435	1750	1750	1953	1961	2088	2088	2336
Styletester										561
Pdfimport	485	485	450	474	474	478	478	473	473	
Gui	41282	42265	48735	52236	53165	56494	58590	58365	58452	47818
Collab	1784	1784	1574	1493	1482	56594				
Shared			1208	1659	1661	1662	1681			

In “Model” package LOC has increases drastically from V3.6 to V3.7 and has a considerable change in LOC on further packages, increasing the maintenance of the package as LOC is directly proportional to maintenance of the product.

Cohesion is the inter-relatedness among class members and methods of a class. Cohesion has a negative effect on the complexity of the code, increasing the maintenance of the product. The metric to measure cohesion is Lack of Cohesion between Methods (LCOM). Greater LCOM values indicate very poor cohesion between methods of a class. LCOM metric is inversely proportional to maintenance of the code. The following table depicts LCOM values obtained by using Metric reloaded tool.

Table7. LCOM Table

LCOM										
VERSION	**V 3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V 4**	**V 4.1**	**V 4.2**	**V4.3**	**V 5**
Architecture						0	0	0	0	0
Cli	2.25	2.25	1.56	1.56	1.56	1.56	1.56	1.62	1.62	1.86
Logic	1.77	1.78	2.03	1.94	1.94	1.99	2.04	2.07	2.09	2.13
Migration	2	2	2	2	1.67	2.33	2.33	2	1.2	1.2
Model	1.91	1.94	1.8	2.22	2.03	2.25	2.25	2.28	2.31	2.49
Preferences			2	Y	2.36	2.33	2.33	2.33	2.33	2.35
Styletester										2.5
Pdfimport	1.75	1.75	1.75	2	2	2	2	2	2	
Gui	1.79	1.77	1.81	1.77	1.78	1.79	1.78	1.81	1.81	1.98
Collab	1.84	1.84	1.84	1.89	1.89	1.89				
Shared			2.14	1.95	1.95	1.95	1.95			

“cli” package is frequently updated over versions and has no pattern which depicts instability of the package. ”Migration” package had a constant value from V3.4 to V3.7 then the value was gradually increasing up to 4.2 and a sudden fall in LCOM value was noticed from V4.2 to V5.0 indicating low maintenance required to handle the package.

4.2. Analysis on the Understandability

Here with these metrics we have successfully discussed and found the suitable answers for the Q3 and the following data to prove those results was collected from the MetricsReloaded plugin installed in the IntelliJ IDE.

The results collected from the tools for the Ca metrics is shown in the table below:

Table8. Ca Table

Ca										
VERSIONS	**V 3.4**	**V3.5**	**V3.6**	**V 3.7**	**V3.8**	**V 4**	**V 4.1**	**V 4.2**	**V 4.3**	**V 5**
Architecture						0	0	0	0	0
Cli	5	5	5	5	5	5	5	5	5	6
Logic	23	10	12	12	7	6	6	6	6	3
Migration	2	2	2	2	2	2	2	5	5	5
Model	17	32	31	43	54	55	54	59	59	31
Preferences			64	63	65	69	70	65	65	79
Styletester										0
Pdfimport	6	6	6	6	6	6	6	6	6	
Gui	282	280	271	267	272	351	358	352	352	202
Collab	13	13	13	13	13	13				
Shared			6	11	11	11	11			

Afferent coupling is measured and shown in the table. With increase in Afferent coupling the understandability decreases, since the dependency of the class on other packages is measured by afferent coupling which means increase in value of afferent coupling means more dependency on other packages. This dependency on other packages makes the program hard to understand. From the table three packages “model”, “preferences” and “gui” are having high value of afferent coupling. The model package shows an increase of afferent coupling value from version 3.4 to version 3.8 and then approximate constant value till version 4.3 and then a decrease to a relatively low value which show the scope for maintainability. The package preferences have also

relatively high value from version 3.6 to version 5.0. The package logic stands different and has very high afferent coupling value relative to all other packages, it shows a significant increment from version 3.8 to version 4.0 and also a drop from version from 4.3 to version 5.0 but still the value is relatively very high. This shows the package "gui" is hard to understand thus hard to maintain.

Efferent coupling is measured and shown in table with increase in efferent coupling the understandability decreases. The dependency of other packages on the class in scope is measured by efferent coupling. This increase in dependency of other packages on this class in scope makes understandability low. The increase in efferent coupling reduces the ease of maintenance. As we can observe the graph for the package GUI the initial versions had high maintenance as the Ce values of the package was large and later versions the Ce values have reduced, and the maintenance of the package reduced. In the package cli package we observe that the initial versions have exceptionally low Ce values and had low maintenance and in the later versions has slightly increased and resulted in the high maintenance of the package. In the package migration we also observe that the initial versions have exceptionally low values of Ce and had low maintenance of the package and in further versions the values have slightly increased making the package high maintenance when compared to the initial versions.

Table9. Ce Table

Ce										
VERSIONS	**V3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V 4**	**V4.1**	**V4.2**	**V4.3**	**V5**
Architecture						0	0	0	0	0
Cli	24	24	47	50	50	50	50	46	46	43
Logic	9	8	8	8	4	4	4	4	4	3
Migration	12	12	17	17	24	25	25	37	37	33
Model	6	7	7	8	8	6	6	6	6	2
Preferences			25	38	38	44	44	47	47	70
Styletester										4
Pdfimport	20	20	22	27	27	28	28	26	26	
Gui	459	457	421	437	411	442	446	432	432	229
Collab	85	85	89	85	85	87				
Shared			14	22	22	22	22			

It was also observed for the CR metrics that the maintenance is inversely proportional to the metrics CR. The results that were collected for the CR metrics for the packages of 10 different versions are as follows:

Table10 CR Table

Comment Ratio										
VERSIONS	**V 3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V4**	**V4.1**	**V4.2**	**V4.3**	**V5**
Cli	0.067	0.083	0.08	0.08	0.08	0.08	0.08	0.054	0.054	0.055
Logic	0.286	0.291	0.202	0.191	0.191	0.185	0.187	0.184	0.183	0.176
Migration	0.256	0.256	0.209	0.209	0.199	0.178	0.181	0.153	0.153	0.125
Model	0.3	0.299	0.265	0.256	0.252	0.246	0.247	0.244	0.244	0.237
Preferences			0.132	0.11	0.106	0.103	0.105	0.099	0.099	0.096
Pdfimport	0.169	0.169	0.095	0.09	0.09	0.089	0.089	0.088	0.088	
Gui	0.177	0.178	0.127	0.125	0.124	0.119	0.117	0.115	0.115	0.089
Collab	0.237	0.237	0.128	0.138	0.137	0.138				
Shared			0.236	0.203	0.203	0.203	0.201			

According to the above results generated and the conclusions drawn from "logic" package the outputs that had been collected and observed the variations keenly in this package declares us that the Code to comment ratio in the version 3.4 and 3.5 we can observe a negligible increase of the

comment ratio and later it had an exceptional decrease in the 3.6 version and had gradually decreased in the further versions but an except case where a negligible increase of the ratio between 4.0 and 4.1 versions. Hence, the decrease in the values are only in the initial versions and thus 3.6 version the package is more difficult to maintain than the 3.4 and 3.5 versions and 4.1 versions is easy to maintain than the 4.0 version. According to the above results generated and the conclusions drawn from ”GUI” package the outputs that had been collected and observed the variations keenly in this package declares us that the Code to comment ratio in the versions 3.4 and 3.5 it has negligibly increased and has remarkably decreased in the next version i.e. 3.6 version and has gradually decreased for further versions hence 4.0 version has a decreased value than the difficult to maintain than the previous version 3.8 and the further versions are also equally maintained for this package. According to the above results generated and the conclusions drawn from ”pdfimport” package the outputs that had been collected and observed the variations keenly in this package declares us that the Code to comment ratio in the versions 3.4 and 3.5 versions is constant and has exceptionally decreased in 3.6 version and since has slightly decreased in further versions. Hence there is a decrease in the values in 3.5 and 3.6 versions thus 3.6 is more difficult to maintain than the initial versions and the further versions have not shown any noticeable change and thus are considered to be equally maintained versions for these packages.

4.3. Analysis on Complexity

The Q2 of GQM is discussed in this part where the relation of how the complexity effect the maintainability of the software project. Complexity is a crucial factor in determining the ease of maintenance of the software project. Ease of maintaining of software depend on various factor but complexity is one of the important factors in determining the maintainability. Metrics as v(G), CBO and WMC are used to estimate complexity and maintainability relation.

Table11. v(G) Table

COMPLEXITY										
VERSION	**V 3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V4**	**V4.1**	**V4.2**	**V4.3**	**V5**
Architecture						0	0	0	0	0
Cli	2.6	2.6	2.8	2.8	2.8	2.7	2.7	2.5	2.5	2.4
Logic	3.2	3	3.2	3.4	3.3	3.2	3.2	3	3	2.9
Migration		4.7	4.6	4.6	4.3	3.6	3.6	3	2.9	2.6
model	2.3	2.2	2.1	2.1	2.1	2	2	1.9	1.9	2
Preferences			1.7	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Styletester										1
Pdfimport	2.7	2.7	2.7	2.7	2.7	2.7	2.7	2.8	2.8	
Gui	2.5	2.5	2.5	2.5	2.5	2.3	2.3	2.2	2.2	1.8
Collab	2.7	2.7	2.7	2.8	2.8	2.8				
Shared			1.9	1.8	1.8	1.8	1.8			

Cyclomatic complexity is measured with the Lizard tool and collected values are shown in the table. With increase in complexity maintenance also increases. The module “cli”,“pdfimport”, migration and logic are the modules which are showing relative high complexity. Here the logic module is having highest complexity among other modules and migration also have high complexity in initial versions but have as shown in table migration module have steady drop in complexity from first version 3.4 to last version 5.0 which show the scope of maintainability. The module logic does not show any significant drop in complexity and maintain high complexity value thought all versions which is from version 3.4 to 5.0 which show that module is hard to maintain. The module “cli” and “pdfimport” are also showing steady relatively high complexity and are also belong to the group which show these are hard to maintain based on their complexity nature.

The maintainability index which is calculated based on cyclomatic complexity and number of lines of code (LOC) and Halstead volume has can be used to see a relation between the complexity and maintenance and to validate the result. The maintainability index shows that the lower the value of maintainability index the harder is to maintain the project and higher the value means highly maintainable. There is a very similar patter between the maintainability index and the complexity, and both shows the similar result of relation between the complexity and maintainability. The result observed is that cyclomatic complexity is inversely proportional to maintainability of packages.

Table12. MI Table

Maintainability Index										
VERSIONS	**V3.4**	**V3.5**	**V3.6**	**V3.7**	**V 3.8**	**V4**	**V4.1**	**V4.2**	**V4.3**	**V5**
architecture						0	0	0	0	0
Cli	43.83	43.83	43.41	43.37	42.98	44.49	44.49	44.96	44.96	44.34
Logic	45.04	45.33	45.1	44.83	44.96	45.62	45.8	46.03	46.1	46.21
Migration	27.7	27.7	27.09	26.73	27.93	33.9	32.73	37	37.11	38.03
Model	42.93	43.36	44.39	45.91	45.89	46.8	46.79	47.25	47.23	48.76
preferences			40.55	41.47	41.28	39.5	39.48	38.44	38.44	36.7
Styletester										53.5
Pdfimport	39.57	39.57	39.57	38.91	38.91	36.55	36.55	36.64	36.64	
Gui	37.75	37.6	37.91	37.89	37.8	41.4	41.67	41.88	41.85	47.65
Collab	42.23	42.23	42.28	41.99	42.12	42.11				
Shared			48.31	48.01	49.33	49.32	49.22			

Maintenance is directly proportional to the WMC metrics i.e. as the WMC increase it becomes easy to maintain the package.

Table13. WMC Table

WMC										
VERSIONS	**V 3.4**	**V3.5**	**V3.6**	**V 3.7**	**V 3.8**	**V 4**	**V 4.1**	**V 4.2**	**V 4.3**	**V 5**
Architecture						0	0	0	0	0
Cli	113	113	148	148	149	152	152	136	139	18.43
Logic	3278	3194	5265	5153	5187	5299	5416	5677	5686	5844
Migration	41	41	44	44	51	62	69	91	91	76
Model	742	759	760	1593	1570	1786	1784	1827	1831	1836
Preferences			123	187	191	218	218	236	236	291
Styletester										6
Pdfimport	55	55	54	56	56	57	57	58	58	
Gui	4723	4830	5770	6356	6463	6851	7048	6979	6985	
Collab	199	199	199	183	183	183				
Shared			137	200	200	200	201			

According to the results generated and observations we here conclude that the package like ”GUI” the WMC metrics values have gradually decreased over the versions 3.4 to 5.0 of the JabRef system which makes it a low maintenance of the package. For the package ”Migration” the outputs that had been collected and observed the variations keenly in this package declares us that the WMC in the versions 3.4-3.5 and 3.6-3.7 remains constant and increase between v3.5-v3.6 and later there is an exceptional decrease between 3.7 and 3.8 and later it increases from version 3.8 to 4.0. And again, decreases between 4.1 and 4.2 and remains constant between 4.2 and 4.3 and again decreases in the version 5 making the migration package low maintenance. For the package ”Model” the data collected and observed shows that the values were decreasing overall from the version 3.4 and 5.0 but had an exceptional decrease in the versions 3.6 and 3.7 and

hence that makes the package model low maintenance. For the package ”Logic” the outputs that had been collected and observed the variations keenly in this package declares us that the WMC in the versions 3.4 and 3.5 there is a gradual decrease and then from the versions 3.5 and 3.6 there has been a remarkable increase and for later version it has been increasing since then making the package high maintenance package.

It was also observed for the CBO metrics that the maintenance is directly proportional to the metrics CBO.

The results that were collected for the CBO metrics for the packages of 10 different versions are as follows:

Table14. CBO Table

CBO										
VERSIONS	**V 3.4**	**V3.5**	**V3.6**	**V3.7**	**V3.8**	**V 4**	**V41**	**V 4.2**	**V4.3**	**V 5**
Architecture						0	0	0	0	
Cli	0.75	0.75	0.44	0.44	0.44	0.44	0.44	0.38	0.38	0.43
Logic	2.62	2.51	3.18	3.25	3.31	3.4	3.44	3.38	3.39	3.53
Migration		0	0	0	0	0.33	0.33	0.2	0.2	0.2
Model	4.49	1.81	2.08	3.06	2.84	2.9	2.91	2.86	2.84	3.55
Preferences	6.36		0.75	0.73	0.73	0.67	0.67	0.67	0.67	
Styletester										0
Pdfimport	8.23	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
Gui	10.1	4.44	4.62	5.19	5.15	4.99	5	4.92	4.93	4.84
Collab		0.37	0.37	0.28	0.28	0.28				
Shared	11.97		1.14	1.6	1.6	1.6	1.6			

According to the above results generated and the conclusions drawn from it were that for “cli” package the outputs that had been collected and observed the variations keenly in this package declares us that the CBO in the versions 3.4 and 3.5 have the same constant outputs whereas, the later versions from the 3.6 to 4.1 had a massive decrease in the outputs values but were constant in all the versions further which was equal to initial versions i.e. 3.4 and 3.5 respectively. Hence as the CBO metrics value decreased from versions 3.4-5.0 it is concluded as a low maintenance package for the latest version when compared to the previous versions. According to the above results generated and the conclusions drawn from the package “gui” the outputs that had been collected and observed the variations keenly in this package declares us that the CBO for all the version is initially increasing from the version 3.4 to 3.7 but negligible decrease between 3.4 and 3.5 and has further decreased from version 3.7 to 5 but negligible increase between the 4.0-4.1 and 4.2-4.3 respectively as the values is frequently changing and the has noticeably increased from 3.4 version to 5.0 version hence it is high maintenance package. According to the above results generated and the conclusions drawn from the ”migration” package the outputs that had been collected and observed the variations keenly in this package declares us that for most of the versions the CBO has been 0 and the has rose to 0.33 from 4 version and then has been decreased to 0.2 from version 4.2 and further. Hence, considering the frequent change and considering the initial version and final version and values had increased and thus it is considered a highmaintenance package.

4.4. Analysis on Cohesion

In this part we are answering Q4 of GQM, how does cohesion effect maintainability. LCOM of CK metric suite is considered for analysing cohesion and maintainability relation. The data is collected using Metric-Reloaded Plugin with IntelliJ IDE. Observation and Reflection Cohesion is directly proportional to maintainability of a package i.e. with increase with cohesion package

are easy to maintain. LCOM shows the lack of cohesion between methods of a class. The greater value of LCOM means packages are hard to maintain.

package is frequently updated over versions and has no specific pattern which shows instability of package. Cohesion in the migration package has constant value from version 3.4 to version3.7. In the later versions the package becomes a bit unstable as we go from version 3.7 to version 4.2 and suddenly fall in LCOM value from version 4.2 to 5.0 which shows low maintenance.

4.5. Overall Evaluation

In this part we are answering Q4 of GQM, how does cohesion effect maintainability. LCOM of for overall evaluation we have processed the collected data and summarized it for more possible evaluation. In this section for every package all the values of the metrics are processed by mean and the mean is done over time to find a single value for all packages and the values are normalized to find comparative results. The data is processed according to the direct proportionality or inverse proportionality as required by values and as the result have been found in the above subsections.

Table15. Normalised Table

	NORMALIZED METRIC VALUES									
PACKAGE	**DIT**	**LCOM**	**LOC**	**CBO**	**v(G)**	**Ca**	**Ce**	**MI**	**CR**	**WMC**
Cli	0	0	0.0095	0.0998	0.5927	0.017	0.0945	0.3076	1	0.974
Logic	0.1121	0.3131	0.6705	0.6535	0.7734	0.0304	0.0038	0.2538	0.094	0.687
Migration	0	0.175	0	0.0514	1	0.0097	0.0482	1	0.131	1
Model	0.1218	0.5368	0.1932	0.599	0.3831	0.1456	0.0053	0.2384	0.221	0.851
Preferences	0.2466	0.7872	0.0283	0.1426	0.1897	0.2259	0.0972	0.5153	0.54	0.948
Styletester	0	1	0.0018	0	0	0	0	0	0	0
Pdfimport	0.5682	0.2324	0.0001	0.102	0.6224	0.02	0.0506	0.5846	0.536	0.712
Gui	1	0.0907	1	1	0.4807	1	1	0.4692	0.2	0.576
Collab	0.4487	0.1644	0.2012	0.0645	0.6325	0.0435	0.1987	0.3923	0.2	0.523
Shared	0.4222	0.3263	0.2163	0.3078	0.2963	0.0334	0.0397	0.1384	0.0826	0.505

All the packages are classified into three difficulty levels on ordinal scale as high maintainability, moderate maintainability, low maintainability packages. Mean values of a metric for a specific package is calculated and normalized according to the dependency of the metric on maintainability. The above table depicts the normalized values of various metrics. Depending upon these values packages are classified into 3 categories, then the mode represents the difficulty of maintenance as form difficulty level of high to low if the mode have sufficientvalue to be in high difficulty level it is placed in high else it is considered for medium level of difficulty and still if not fit in the difficulty level it is finally in low difficulty level.

Table16. Maintainability Level Table

PACKAGE	HIGH	MEDIUM	LOW	FINAL MAINTENANCE LEVEL
Cli	2	1	7	MEDIUM
Logic	3	0	7	HIGH
Migration	3	0	7	HIGH
Model	1	3	6	MEDIUM
Preferences	2	2	6	MEDIUM
Styletester	1	0	9	LOW
Pdfimport	1	4	5	MEDIUM
gui	5	3	2	HIGH
Collab	0	4	6	MEDIUM
Shared	0	2	8	LOW

Table17. Packages Maintainability Classification Table

Package	Maintenance	Reason
cli	Medium	-
Logic	High	LOC,v(G),WMC,CBO
migration	High	v(G),WMC
Model	Medium	LCOM,CBO,WMC
preferences	Medium	-
Styletester	Low	-
pdfimport	Medium	DIT,CR,v(G)
gui	High	DIT,LOC,CBO,Ca,Ce,WMC
Collab	Medium	v(G)
Shared	Low	-

4.6. Change log and Timeline

The timeline for how the JabRef versions is released and duration between them is given in the figure below.

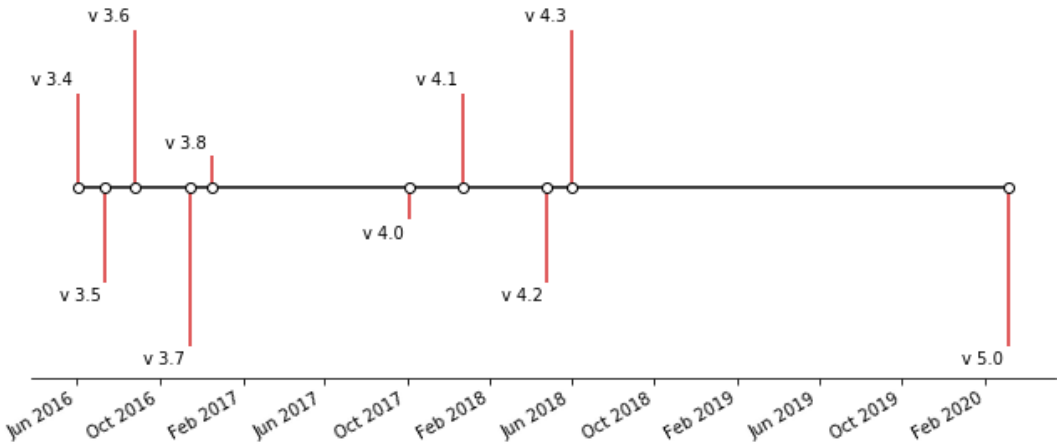

Fig 1. Version Release Timeline

Table18. Change log Table

Version	Changes	Fixed	Removed
V 3.4	18	31	6
V 3.5	8	15	0
V 3.6	33	44	7
V 3.7	48	45	5
V 3.8	17	15	0
V 4.0	11	22	1
V 4.1	26	28	0
V 4.2	28	17	1
V 4.3	9	6	1
V 5.0	8	25	2

We have observed all the 10 versions of the JabRef system namely 3.4, 3.5, 3.6, 3.7, 3.8, 4.0, 4.1, 4.2, 4.3, 5.0 that initially in the 3.4 version there many packages like “Cli”, “logic”, ” migration”, “model”, “pdfimport”, “gui”, “collab”, “specialfields”, “event”, “external”, “bst”, exporter, “sql”, “util”, “importer” and as the new versions were introduced these packages were either removed or were merged in the other packages i.e. the “bst” and “specialfields” packages were removed and the packages like importer, exporter, external, collab were merged into the GUI package and che packages namely “shared”, “event”, “util”, “sql” were merged into “model” package. Hence as these small packages were present all in few versions and were again merged in other packages, we have neglected these packages during the analysis done by the metrics. From change log the extracted data for the issue changes in version and issue fixed and issue removed are shown in table below.

4.7. Conclusion of Results and Future Work

Proportionality of an object-oriented metric to the maintainability of a product. The relation of the metric and maintainability is discussed in the above sub-section of analysis and the inverse or direct proportionality is also discussed. We could use the study to obtain the maintainability level for various Object oriented software system by considering the goal and the questions covering our goal and the metrics which are helpful in answering the question in scope to fulfil the goal and a similar approach can be applied to obtain the result to be found.

As in this paper we majorly focused on factors such as complexity, size, structure and understandability to determine the Maintainability. The Future work for the study is to do a study on various Object-Oriented open source system independent of the language and to analyse the result to while increasing the scope of attribute and similarly increasing the scope of metrics.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] Robson, Colin. (2002). Real World Research : A Resource for Social Scientists and Practitioner-Researchers / C. Robson.

[2] Laing, Victor Coleman, Charles: Principal Components of Orthogonal Object-Oriented Metrics. White Paper Analyzing Results of NASA Object-Oriented Data. SATC, NASA, 2001.

[3] Horst Zuse. 1991. Software complexity: measures and methods. Walter de Gruyter Co., USA.

[4] Bansal, M., Agrawal, C.P., 2014. Critical Analysis of Object Oriented Metrics in Software Development, in: 2014 Fourth International Conference on Advanced Computing Communication Technologies. IEEE, pp. 197–201. doi:10.1109/ACCT.2014.106.

[5] V.R. Basili and D. Weiss (1984), A Methodology for Collecting Valid Software Engineering Data, IEEE Trans. Software Engineering, vol. 10, pp.728-738.

[6] V.R. Basili and H.D. Rombach (1988), The Tame Project: Towards Improvement-Oriented Software Environments, IEEE Trans. Software Engineering, vol.14, pp.758-773.

[7] https://github.com/JabRef/jabref

[8] Yadav, A. and Khan, R.A., 2011, September. Class cohesion complexity metric (C 3 M). In 2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011) (pp. 363-366). IEEE.

[9] Kaur, K. and Singh, H., 2011, February. Towards a Valid Metric for Class Cohesion at Design Level. In 2011 Second International Conference on Emerging Applications of Information Technology pp. 351-354. IEEE.

[10] Al Dallal, J., 2012. Theoretical analysis for the impact of including special methods in lack-of-cohesion computation. Procedia Technology, 1, pp.167-171.

[11] Dubey, S.K., Rana, A., 2011. Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Softw. Eng. Notes 36, pp. 1–7. doi:10.1145/2,020,976.2020983.

[12] Sandesh Ganjare, Koustubh Kulkarni, Dinesh B Hanchate et al. Measuring Structural Code Quality Using Metrics. Inventi Rapid: Soft Engineering, 2015(3):1-7, 2015.

[13] S R Chidamber and C F Kemerer. A Metrics Suite for Object Oriented Design. IEEE International Conference on Data Mining, 150-159, 2008.

[14] Simon, Martin, Linus W. Dietz, Tobias Diez and Oliver Kopp. "Analyzing the Importance of JabRef Features from the User Perspective." ZEUS (2019).

[15] Al-Jamimi, H.A., Ahmed, M., 2012. Prediction of software maintainability using fuzzy logic, in: 2012 IEEE International Conference on Computer Science and Automation Engineering. IEEE, pp. 702–705. doi:10.1109/ICSESS.2012.6269563.

[16] Wirotyakun, A., Netisopakul, P., 2012. Improving software maintenance size metrics A case study: Automated report generation system for particle monitoring in Hard Disk Drive Industry, in: 2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE). IEEE, pp. 334–339. doi:10.1109/JCSSE.2012.6261975.

[17] Kaur, A., Kaur, K., Pathak, K., 2014. Software maintainability prediction by data mining of software code metrics, in: 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC). IEEE, pp. 1–6. doi:10.1109/ICDMIC.2014.6954262.

[18] Dubey, S.K., Rana, A., 2011. Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Softw. Eng. Notes 36, pp. 1–7. doi:10.1145/2,020,976.2020983.

[19] Saifan, Ahmad Alsghaier, Hiba Khateeb, Khaled. (2017). Evaluating the Understandability of Android Applications. International Journal of Software Innovation. 6. 10.4018/IJSI.2018010104.

[20] Jehad Al Dallal,Object-oriented class maintainability prediction using internal quality attributes,Information and Software Technology,Volume 55, Issue 11,2013,Pages 2028-2048,ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2013.07.005.

AUTHORS

Denim Deshmukh
Denim Deshmukh is currently completing his master's degree in Software Engineering from Blekinge TekniskaHögskola, Sweden. He is looking forward for contributing to the field of Software Engineering.

Ravi ThejaKataray
Ravi ThejaKataray currently pursuing a Master of Science in the field of Software Engineering from Blekinge TekniskaHögskola, Sweden. Striving hard to work in a team for a cause of gradually developing software methodology.

Rohith Girikshith
Rohith Girikshith is currently pursuing a Master of Science in the field of Software Engineering from Blekinge TekniskaHögskola, Sweden. He is doing his research work in the field of Software Engineering.