PA2559
Software Metrics - Project

DhanaLakshmi Ponugubati – *dhpo18@student.bth.se – 9805013563*
Muhammad Waqas – *muaq18@student.bth.se – 9007038251*
Waleed Rasheed – *aara18@student.bth.se – 9301155033*
SriRanganathReddy Sabbella – *srsa18@student.bth.se – 9705230119*

May 18, 2019

Abstract

In this study we evaluated the maintainability of jEdit using an GQM framework based on empirical study. The case study is done in order to evaluate the maintainability. The goal of this study is to identify the jEdit modules that are difficult to maintain. We evaluated this goal using four internal attributes: Code structure, Code complexity, Code understandability, Code size. The metrics were derived for each and every attribute and results are analyzed using different statistical measures and visualization methods.

Index Terms : *–Maintainability, Object Oriented, Metrics, Case Study, GQM paradigm.*

1 Introduction

Object-Oriented has become a great approach in software programming. Object-Oriented programming contains many features such as inheritance, encapsulation, dynamic binding, method binding and polymorphism. These features of object oriented programming mainly focuses on reusability and component based development [1]. The software quality assurances can be obtained by the object oriented metrics. Estimating the quality of a software includes estimating the reliability and maintenance. As the demand for object oriented metrics is been increased there need to be many methods coming into practise [2].

The demand for quality in every field is increasing because of the increases in dependencies in every field and also effect of errors are increasing. The software's that are in toady's society should be error free and also with high quality. Delivering a quality product has become mandatory than advantage. The object oriented system requires all the effort in the early stages that is in the development processes. Hence this early approach with object oriented metrics helps to assess the quality of the classes, design in the early stage [3].

Every design or software may have some problems that are to be solved. These problems should be identified and reduced in order to evaluate the quality of the software [4].

The object-oriented mainly focus on objects. In order to evaluate the quality of the object oriented code there are some traditional and new metrics implemented. All the defined object-oriented metrics are used to evaluate the attributes of software quality. The attributes are maintainability, reusability, efficiency, complexity, understandability. Each of these attributes have a great impact on evaluation of software quality of object oriented code [5].

1.1 Summaries of all other parts

The main focus of this project is achieve the goal of the GQM framework. The goal of the project is mainly stated in the GQM tree and all the requirements for achieving the goal are described. The goal, questions and metrics are stated in section 3. The section 4 gives the solution to each and every question stated in the GQM and also all the findings of the project are stated.The section 5 contains information about related work,reflections and comparison of results.Section 6 contains threats to validity and section 7 contain concluded the project.

The main findings of this project are the modules which are more difficult to maintain. By analyzing the results the modules which are mostly modified, having high coupling, having low cohesion, and also the code structure, code complexity are also evaluated. The code understandability is evaluated by using different metrics related to them.

2 Research Methodology

2.1 Study Type

The suitable study type for our empirical study is case study. There are different types of research methodologies but we have selected the case study for this project because in this field there are already so many research available for this and in fact case study is very useful in field of software engineering. The case study gives the clear picture to the researchers to test not only the complex of the life in which people are associated but also the impact on beliefs and discussion of social interaction [6]. Case study research is a highly legitimate research method appropriate for both qualitative and quantitative research [7].This approach is best suited for our project because we have collected the data in the form of metrics and then we have analyzed it. If we compare the other approach survey which provide the sampling of individual units from population. It provide the important information for all kind of public research field but there are some problems in this such as survey has close ended questions which may have lower validity rate than other question types and our project is totally based on data collection so if we will use the survey then the data error may exist in our project due to the non-respondent questions. That is the reason we have rejected this approach to use in our project. Similarly if we consider the experiment it has also some limitation like the poor design and low rate of accuracy which is not suitable for our project.

RQ1 : Which jEdit modules have high degree of coupling?

RQ2 : Which jEdit modules have relatively poor cohesion?

RQ3 : Which jEdit modules have high complexity?

RQ4 : Which jEdit modules have high degree of understandability?

RQ5 : Which jEdit modules have been modified over the different versions regarding size?

2.2 Data Collection

2.2.1 Tool for Metric Extraction

To extract the metrics from the different versions of jEdit we used a tool named as Metrics Reloaded which is a plugin in IntelliJ IDEA and also available free to use it i.e. an open source software.

This plugin extracted different metrics at different levels like Package level, Class level, Module level, Interface level etc,. We can extract the metrics for a selected module or entire package or whole project as the requirement by the user. The Metrics that are extracted from this tools are Chidamber-Kemerer (CK) Metrics, Complexity Metrics, Lines of Code Metrics, MOOD Metrics, Martin Packaging Metrics etc can be calculated. We can obtain the metrics by selecting a module/package and then by right clicking select analyze/calculate metrics we get displayed the above metrics.

The selected metrics are Ca, CBO, CC, Ce, CLOC,Comment Ratio, DIT, I, LCOM, LOC, NOC, NOM, RFC, WMC all the metrics are extracted at package level and these relates to internal attributes of Code Structure, Code Understandability, Code Complexity, Code Size.

Justification : We selected other tools like like Stan tool, Metrics 1.3.8 tool, CodeMR these are all plugins in Eclipse IDE and also open source software. While using the tool Stan we got different values for a selection of package or complete project and some values are not properly extracted and the tool has a limit of 500 classes limit. Similarly, Metrics 1.3.8 tool is a metrics extraction tool but its not working of the package extraction of jEdit. CodeMR is a tool which displays outputs as a pie-chart and for the exact extraction metric tool package level values can not be displayed. So the formation of graph made complex using the tool CodeMR. For all the reasons we used Metrics Reloaded works in IntelliJ IDEA for the metrics extraction.

The tool Metrics Reloaded displays output at the console in the table format with the individual module values and at the last it also shows the complete average values of the package/module. We can also export the data in form of excel sheets and as an XML file from the console.

These CK Metrics CBO, WMC, DIT, LCOM, NOC, RFC values are obtained at class level and the other selected metrics Ca, Ce, I, CLOC, Comment Ratio, LOC, NOM, CC values are obtained package level, as the CK metrics are only used to get on class level. The tool itself gave the final average value of all the individual classes so this final average value is considered as the value for metric measure for package level.

2.3 Data Analysis

We collected the required data from the tool and then analyzed the data using certain methods and combination of methods.

2.3.1 Visualization methods

We analyzed our data using diagrams, figures, graphs and tables. The GQM is designed by using a online tool and made modifications to the template depending upon the requirements.

The extracted data from the tool is analyzed in excel sheets. The output from the analysis is visualized using Bar charts, Line charts, Column graphs and Area graphs.

We analyzed the final results in the form of tables. The data is extracted from the graphs and organized in tables. For every question to answer we used both bar charts/line graphs and graphs. We used these visualization methods because the data analyzed is well understood and easy to extract.

We used line charts for CBO, WMC, Ce, Ca, CLOC. Clustered bar charts for DIT and Clustered column for LCOM, RFC, CC, Comment Ratio, I, LOC, NOC, NOM.

We selected each type of graph based on the type of attribute and the metric. The selection mainly based on easy of understanding and clarity. And also we kept some of the tables in the form of figures in order to provide the report with clear understanding.

2.3.2 Statistical measures

The statistical measures are used to extract the results. The tool we used automatically gave the mean and central tendency values for the metrics. The mean value is taken for each and every metric.

The tool we used gave each individual value for metric and also the central tendency values. We analyzed these values and plotted them over visualization graphs.

We also used range values for analyzing some of the results.Range here is difference between minimum and maximum values.

3 GQM Tree

Goal-Question-Metric (GQM) is mainly used to identify the changes or any updates by defining a particular goal that is related to the process or a project. In GQM the overall goals of an organization are expressed. Then questions are formulated in order to meet the goals. Each question should be answered by analyzing the measurements [8].
This GQM measurement contains three levels : There are Conceptual level, Operational level, Quantitative level[9][10].

- Goal (Conceptual Level) : The main requirement of GQM framework is deriving the goal. In the conceptual level we derived the main Goal. The objects of measurement are products,process,resource.
  GOAL : To identify those jEdit modules that would be more difficult to maintain.

  This is the main goal of the project.Generally the goal can be stated by using a GQM template. The GQM goal template is designed by Basil and Rombach. The template for GQM goal definition is represented as below. The GQM template consists of three parts the Purpose, Perspective and Environment.

  – Purpose : This tells the exact purpose of the study. The purpose can be written as to (evaluate, motivate etc.) the (metric, modules, product etc.) in order to (improve, manage, understand etc.) it.
  – Perspective : This tells us the main reason for the study. The perspective of a study can be written as Examine the (changes, measures, defects etc.) from the side of (developer, engineer, customer etc.)
  – Environment : This tells us main concern of the study. The environment comprises of factors from all viewpoints, tools, constraints etc.

- Question (Operational Level) : In Operational level the questions are formulated which serves as the main step for achieving the goal. The objects of measurement are categorized by the questions formulated refer Figure 2.

- Metric (Quantitative level) : Inorder to answer every question a set of data which is associated with the questions is needed. The data can be either subjective or objective. The metrics are used to answer the questions and derive the results from that.

The GQM tree is designed in top-down approach in which first the goal is set and then the questions are build inorder to achieve the goal and to answer the questions metrics are derived. But,While implementing the GQM tree we follow bottom-up for analyzing the results. There can be many-to-many relationships in GQM tree [11].The GQM tree for our project is represented in Figure 1, 2.

In the GQM tree we represented questions as Q1, Q2, Q3, Q4, Q5.These questions are described in the Figure 2. The metrics are explained in further subsections.

3.1 Description and Justification for the Metrics

3.1.1 Coupling Metrics

- *Coupling Between Objects (CBO)* : CBO counts the number of specific classes to which it is coupled[12]. In CBO the methods of one class access the methods or the variables of the another classes. CBO counts these calls between classes from both directions.
  Justification : High coupling reduces the reusability of the classes which results in decreasing the modularity of the classes. A high coupled class becomes sensible to changes which results in maintenance difficulty between the classes.

- *Response For Class (RFC)* : RFC counts the cardinality of number of methods that are started to execute when an object of that class recieves message[12].
  Justification : High RFC increases in size resulting in declining the maintainability. RFC determines the complexity by evaluating the method calls. More complex the modules are hard to maintain.

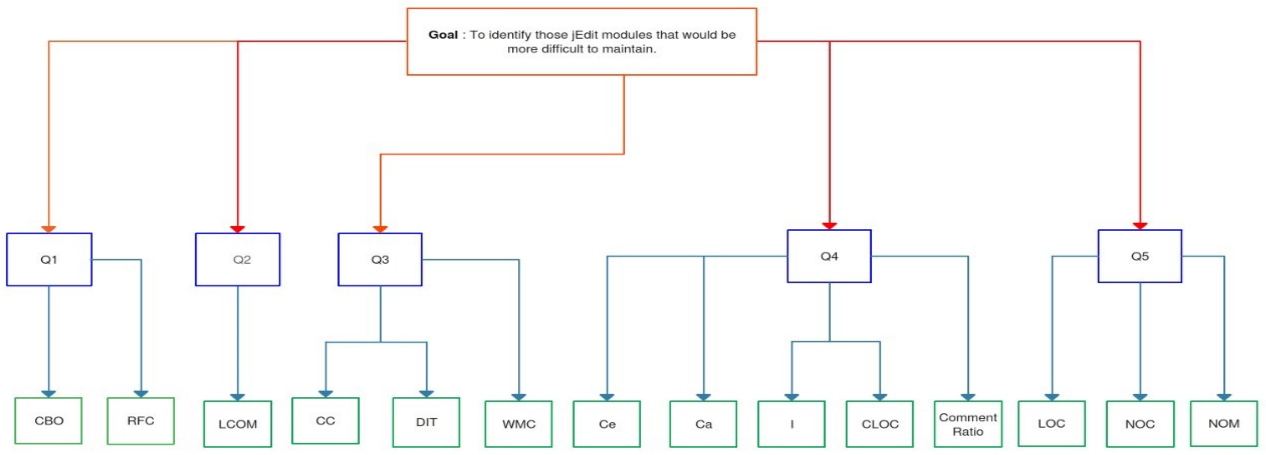
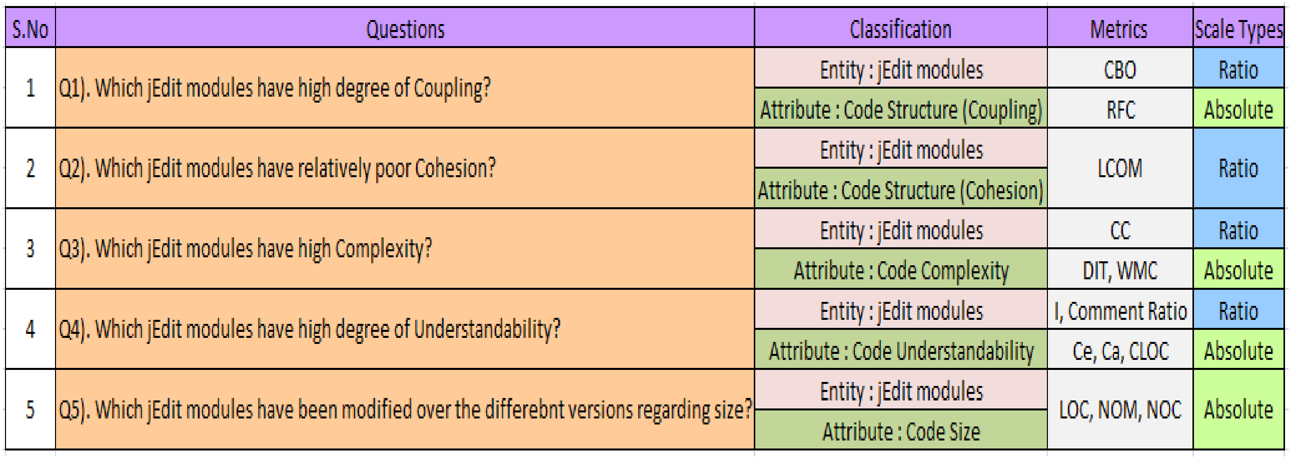

Goal : To identify those jEdit modules that would be more difficult to maintain.
Q1
Q2
Q3
Q4
Q5
CBO
RFC
LCOM
CC
DIT
WMC
Ce
Ca
I
CLOC
Comment Ratio
LOC
NOC
NOM

Figure 1: GQM Tree



S.No	Questions	Classification	Metrics	Scale Types
1	Q1). Which jEdit modules have high degree of Coupling?	Entity : jEdit modules	CBO	Ratio
		Attribute : Code Structure (Coupling)	RFC	Absolute
2	Q2). Which jEdit modules have relatively poor Cohesion?	Entity : jEdit modules	LCOM	Ratio
		Attribute : Code Structure (Cohesion)		
3	Q3). Which jEdit modules have high Complexity?	Entity : jEdit modules	CC	Ratio
		Attribute : Code Complexity	DIT, WMC	Absolute
4	Q4). Which jEdit modules have high degree of Understandability?	Entity : jEdit modules	I, Comment Ratio	Ratio
		Attribute : Code Understandability	Ce, Ca, CLOC	Absolute
5	Q5). Which jEdit modules have been modified over the differebnt versions regarding size?	Entity : jEdit modules	LOC, NOM, NOC	Absolute
		Attribute : Code Size		

Figure 2: Formulated Questions for GQM Framework

3.1.2 Cohesion Metrics

- *Lack of Cohesion in Methods (LCOM)* : LCOM measures the responsibilities of classes. It measures the correct relation among the local instance variable of a class and method [13]. LCOM is subtraction of number of similar methods and number of disjoint methods[10]. LCOM value ranges from 0 (totally cohesive) to 1 (non-cohesive).
  Justification : Cohesion between the classes increases the encapsulation. Low cohesion between the methods results in increasing the complexity which makes it error prone and requires high maintainability.

3.1.3 Complexity Metrics

- *Cyclomatic Complexity (CC)* : CC is used to calculate the complexity of a algorithm in a method. All the complexities of the methods can be combined in order to get the CC of the class. The result of this measure gives how complex the code is to understand and modify [14].
  Justification : Higher the code paths increases the CC which resulted increase in the complexity that makes the modules difficult to maintain.

- *Depth of Inheritance Tree (DIT)* : DIT metric measure the depth of the tree that is the length which calculates it from node to root of the tree. It calculates the level of class in the inheritance hierarchy [15].
  Justification : High value of DIT indicates that there are many methods inherited and it makes it difficult to calculate the individual classes behaviour. It makes the hard to understand the system. High value of DIT also include high reusability.

- *Weighted Methods per Class (WMC)* : WMC is the summation of all the complexities of the class local methods [12]. It also counts the number of methods implemented in a class [10]. The larger number of methods in a class has its impact on its children, since all the methods in a class are inherited by the child [16].

Justification : As the number of methods in a class increase that is WMC value increases the complexity. By measuring the WMC the effort spent on a class can be evaluated.Hence an increase in WMC value increases the effort thereby which makes it difficult to maintain.

3.1.4 Understandability Metrics

- *Efferent Coupling (Ce)* : Ce is defined as the number of other packages that the classes in the package depend upon. It is used to measure outgoing dependencies(Fan-out). Reusability becomes harder as many classes and packages are related to each other [13].
  Justification : Since with increase in Ce increases the number of relations between classes which makes it unstable. The instability results in maintainability reduction.

- *Afferent Coupling (Ca)* : Ca is defined as the number of other packages depending upon the classes present within this package. It measures incoming dependencies(Fan-in)[13].
  Justification : Since with increase in Ce increases the number of relations between classes which makes it unstable. The instability results in maintainability reduction.

- *Instability (I)* : The instability metric is defines as the ratio between efferent coupling to total coupling. Total coupling is calculated by adding the Efferent and Afferent coupling.
  Justification : Higher the instability it is more prone to changes. Hence it makes understandability difficult which results in difficulty in maintenance.

- *Comment Ratio* : Comment Ratio measures the density of comments. It is defined as the ratio between Commented lines of code(CLOC) and lines of code(LOC).
  Justification : Maintainability becomes easy with higher comment ratio value. Commenting the code makes it easy to understand and also code readable.

- *Commented Lines of Code (CLOC)* : CLOC counts the effective lines of code along with the comments in the code. Comments helps when there is need to alter the code. It increases the understanbility of the code.
  Justification : More CLOC value indicates that the maintainability is more because the number of comments increases the size of the code.

3.1.5 Size Metrics

- *Lines of Code (LOC)* : LOC measures the number of lines the source code contains. It is measured inorder to evaluate the understandability of the code. LOC also includes brackets and comments [10].
  Justification : As LOC increases the size of the code increases which may results in increases in errors. Hence it become more error prone and the requires high maintainability.

- *Number of Children (NOC)* : NOC measures the number of sub-classes inherit the methods of the main class. Resusability can also be measured by this metric [15]. If a class has more NOC then the testing for methods is needed [16].
  Justification : Higher the value of NOC the reuseability decreases and hence it becomes difficult to maintain.

- *Number of Methods (NOM)* : NOM measures the total number of operations performed in a class. IT only the class operations related to the methods [10].
  Justification : More NOM values states more operations and method calls in a module. The size increases with increase in NOM thereby it becomes difficult to maintain.

3.2 Scale types and Justification for Metrics

A scale is defined as an tool for measuring the attributes of an entity. Metrics are the measurements of the attributes. These measurements should be categorizes depending upon their behaviour. Hence these scale types are used for categorization. In general there are five types of scales. They are Nominal, Ordinal, Ratio, Interval, Absolute. For categorizing the metrics we used only absolute and ratio as they suits for these measurement values.

3.2.1 Ratio

Ratio scale is the common possible scale where operations such as equity, rank-order, equality of intervals, and equality of ratios can be measured [17]. It also has an absolute zero which is essential for measuring the metrics. All types of statistical measures are applicable between numbers on ratio scale. The size of intervals between entities and ratios between entities are also considered for measuring the metrics. We chose these metrics which are utilizing ratio scale CBO, LCOM, CC, I, Comment Ratio [18].

Justification

- CBO : It counts the number of classes as a class is to be coupled with. The ratio scale is best suites as involves the counting of number of classes that are coupled. We ignored the remaining metrics because it is best suited

- LCOM : It indicates the lack of cohesion between the methods. The ratio scale is best suited for this metric because it indicates the amount of cohesion lack present between the methods. Hence we neglected the remaining scales.
- CC : It calculates the complexity of the modules. The ratio scale is selected because the complexity can also be a zero value. This is not indicated in remaining scale types so we rejected them.
- I : It is ratio of Ce to total coupling(Ce+Ca). As this metric involves the ratio among the Ce and total coupling we selected ratio scale and not taken any other scale type. Comment Ratio : It is ratio of CLOC and LOC. As this metric itself is the ration among CLOC and LOC the ratio scale type is best suites than other scale types.

3.2.2 Absolute

Absolute scale states that there is only one possible measurement mapping that is the actual count. The system of measurement that begins at a minimum or zero point and progresses in one direction. We chose these metrics which are utilizing absolute scale RFC, DIT, WMC, Ce, Ca, CLOC, LOC, NOC, NOM [18].

Justification

- RFC : It is the number of methods in the set of all methods that can be invoked in response to a message. As it counts the total number of methods invoked the absolute is the best over other because some of the scales doesn't involve arithmetic operations.
- DIT : It measures the number of ancestors of a class. As it measures the number of ancestors of a class absolute is the best over other because some of the scales doesn't involve arithmetic operations.
- WMC : It is the sum of weights of all the class methods. As this metric calculates the individual weights and make sum of them the absolute scale is best suited. As it involves arithmetic operations remaining scales are not well suited.
- Ce : It counts the number of unique outgoing dependencies. As this metric involves counting of dependencies absolute scale is the best option where remaining scales may not include arithmetic's in them.
- Ca : It counts the number of unique incoming dependencies. As this metric involves counting of dependencies absolute scale is the best option where remaining scales may not include arithmetic's in them.
- CLOC : It counts the commented lines of code.This metric is involved in counting the comments in a code. This metric determines the understandability of the code. The absolute scale is well suited for this because of its counting we ignored remaining scales as some of them don't support arithmetic.
- LOC : It counts the lines of code. As this metric involves counting of lineso code the absolute scale is best option.We neneglectedemaining scales as they are not well suited for this metric.
- NOC : It is the total number of children of a class.As this mtmetricnvolves counting of the chchildrenf a class the absolute is best suites when compared to others.
- NOM : It is the total number of methods.As this metric involves counting of methods the absolute scale is best suited than any other scale.

4 Results

4.1 Overview

JEdit is an open source software used for editing text. JEdit is developed using java language. SVN repository is used to report the bugs in jEdit [19]. The eleven versions of jEdit analyzed in this project are 5.5.0, 5.4.0, 5.3.0, 5.2.0, 5.1.0, 5.0.0, 4.3, 4.2, 4.1, 4.0, 3.0. Certain modules are selected from these versions of jEdit. Only the important modules are selected and remaining are neglected in order evaluate clear results and understanding of the overall modules. For each release of jEdit the developers behind made several changes. The change include change in code, adding extra features, improving the existing ones. In order to improve the functionalities of jEdit certain changes are made and newer versions are released. Each release is analyzed and the changes are marked down.

Jedit has wide range of world wide developers. It supports wide range of 211 programming languages in its latest version. Many plugins and macros are being added and updated continuously by jedit developers using subversion tool. Extensive range of plugins are used and are easier to install using plugin manger within the jedit. Many builtin macros are available in jedit for making the reuse of code.

The modules we selected in this project are *installer, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.buffer, org.gjt.sp.jedit.msg, org.gjt.sp.jedit.print, org.gjt.sp.jedit.search, org.gjt.sp.jedit.pluginmgr, org.gjt.sp.jedit.browser, org.gjt.sp.jedit.syntax, org.gjt.sp.jedit.io, org.gjt.sp.jedit. options, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea, org.gjt.sp.util* refer Figure 3.

We selected only these modules because of their priority. If we take all the modules we may miss some of the important points due to the overflow of data. So in order to gain better results we limited the number of modules by selecting only that has more effect on maintainability of the system.

The jEdit modules we used in this study are mentioned in the table 3. The modules having highest LOC and NOC are highlighted in the Figure 3. From the last eleven version the minimum and maximum values are gathered and tabled in order to analyze the range and size of the modules.The size of the modules are mentioned in terms of LOC and NOC

4.2 Answers

Q1) Which jEdit modules have high degree of coupling?

Answer : In order to determine the coupling we selected two metrics, Coupling Between Objects (CBO) and Response for Class (RFC). High coupling results in high maintainability. By analyzing the values of various versions of jEdit we came to know the changes in different modules.

For these modules of jEdit *org.gjt.sp.util, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.io, org.gjt.sp.jedit. textarea, org.gjt. sp. jedit.buffer* the CBO values are high and the highest values for these modules are 19.46, 12.83, 11.48, 10.57, 10.7 thus with the increase in CBO values maintenance increases.

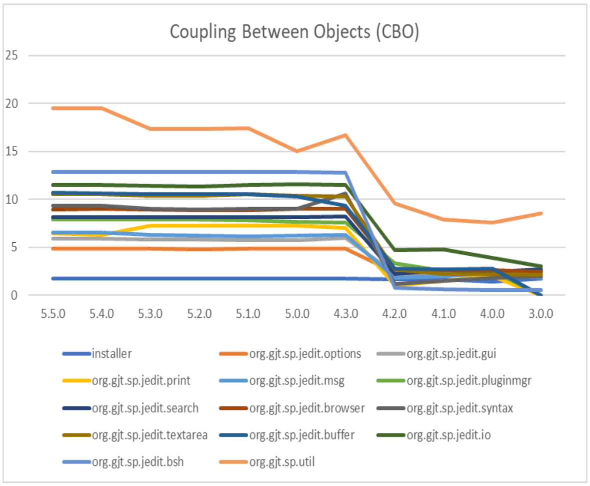


Figure 4: CBO metrics for jEdit versions

From the analysis it is found that *org.gjt.sp.util* module has drastic changes over the different versions and there seemed to be more changes occurred from 4.3.0 on wards. The *org.gjt.sp.jedit.bsh* module is very much updated from 4.3.0. Hence the CBO drastically increase over all the versions which results in high maintenance refer in Figure 5.

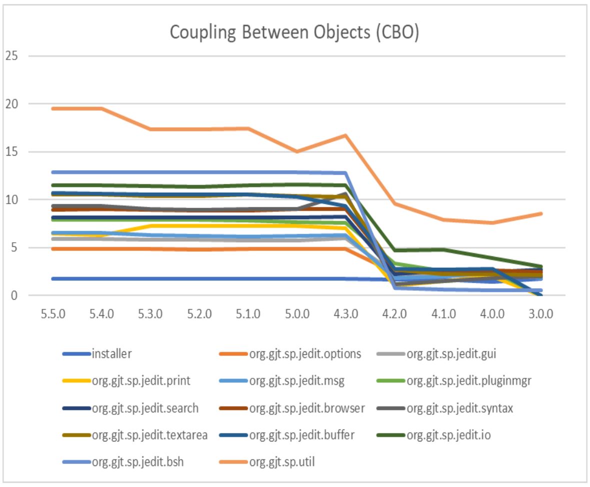


Figure 5: CBO metrics for jEdit versions

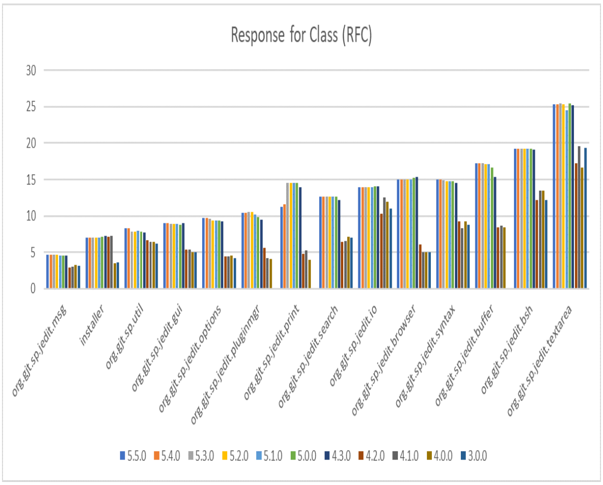


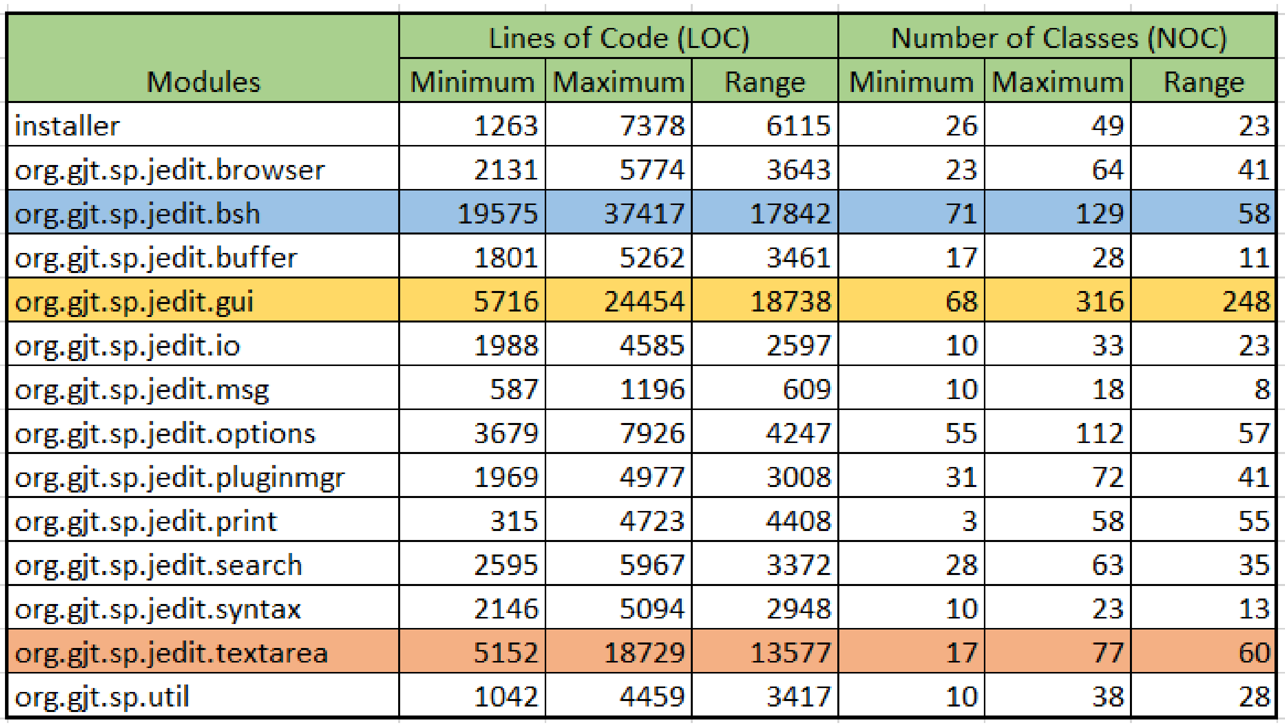
Figure 6: RFC metrics for jEdit versions

For these modules of jEdit *org.gjt.sp.jedit. textarea, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.buffer, org.gjt.sp. jedit.browser, org.gjt.sp.jedit.syntax* the RFC values are high and the highest values for these modules are 25.37, 19.2, 17.23, 15.33, 15.05 thus with the increase in RFC values the maintenance increases refer in Figure 6. The RFC increased over the versions but there seemed to be a large different from version 4.3.0.The module *org.gjt.sp.jedit.textarea* has been greatly modified overall the versions.

These modules *org.gjt.sp.util, org.gjt.sp. jedit.bsh, org.gjt.sp.jedit.io, org.gjt.sp.jedit.textarea, org.gjt.sp. jedit.buffer* have high coupling refer in Figure 7. Hence high coupling leads to poor maintainability.

Q2) Which jEdit modules have relatively poor cohesion?

Answer : In order to determine the cohesion we selected one metric, Lack of Cohesion in Methods (LCOM).Low cohesion results in high mainatainability. When moving forward to latest version



Modules	Lines of Code (LOC)			Number of Classes (NOC)		
	Minimum	Maximum	Range	Minimum	Maximum	Range
installer	1263	7378	6115	26	49	23
org.gjt.sp.jedit.browser	2131	5774	3643	23	64	41
org.gjt.sp.jedit.bsh	19575	37417	17842	71	129	58
org.gjt.sp.jedit.buffer	1801	5262	3461	17	28	11
org.gjt.sp.jedit.gui	5716	24454	18738	68	316	248
org.gjt.sp.jedit.io	1988	4585	2597	10	33	23
org.gjt.sp.jedit.msg	587	1196	609	10	18	8
org.gjt.sp.jedit.options	3679	7926	4247	55	112	57
org.gjt.sp.jedit.pluginmgr	1969	4977	3008	31	72	41
org.gjt.sp.jedit.print	315	4723	4408	3	58	55
org.gjt.sp.jedit.search	2595	5967	3372	28	63	35
org.gjt.sp.jedit.syntax	2146	5094	2948	10	23	13
org.gjt.sp.jedit.textarea	5152	18729	13577	17	77	60
org.gjt.sp.util	1042	4459	3417	10	38	28

Figure 3: Overview of selected jEdit Modules

the LCOM value decreased stating the cohesion between the methods increased which results in high maintainability. For these modules of jEdit *org.gjt.sp.jedit.io, org.gjt.sp.util, org.gjt.sp.jedit.syntax, org.gjt.sp.jedit.textarea, installer* the LCOM values are high and the highest values for these modules are 3.89, 2.7, 2.38, 2.03, 2.11 thus with the increase in LCOM values maintainability increase refer in Figure 8 [20]. The module *org.gjt.sp.jedit.io* has drastic changes in different versions.

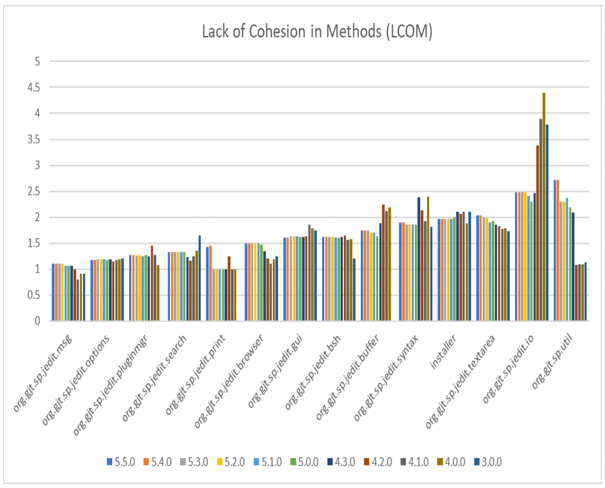
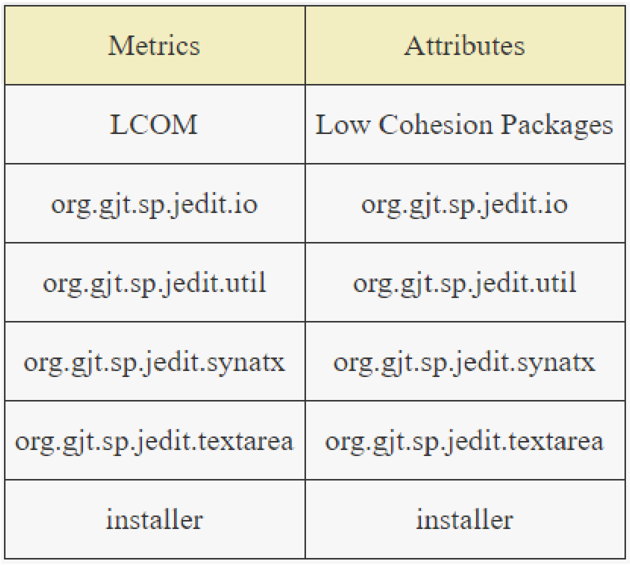


Figure 8: LCOM metrics for jEdit versions

These modules *org.gjt.sp.jedit.io, org.gjt.sp.util, org.gjt.sp.jedit.syntax, org.gjt.sp.jedit.textarea, installer* have low coupling refer in Figure 9. Hence low coupling leads to poor maintainability.

Hence poor cohesion leads to poor maintainability.

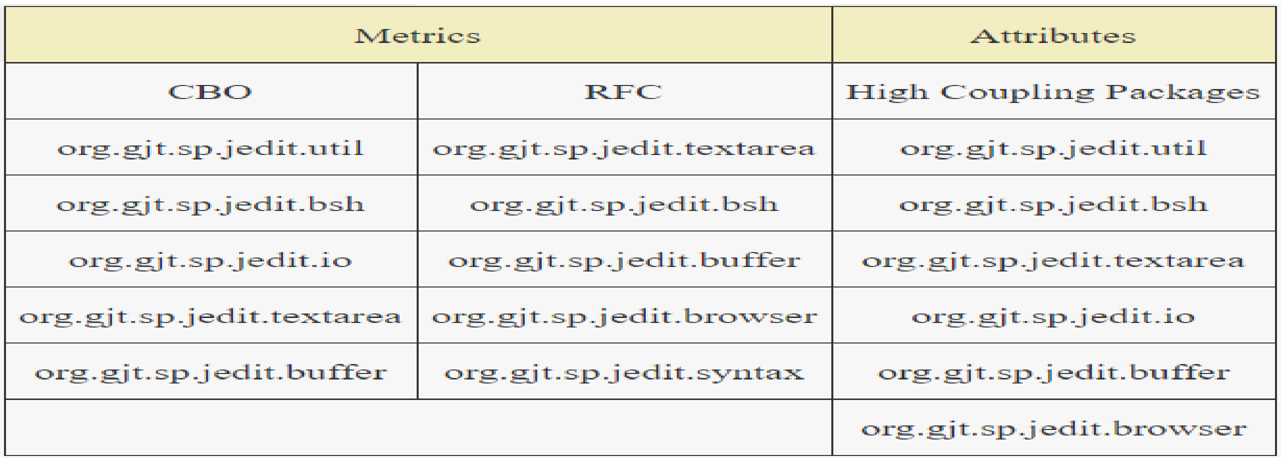


Metrics	Attributes
LCOM	Low Cohesion Packages
org.gjt.sp.jedit.io	org.gjt.sp.jedit.io
org.gjt.sp.jedit.util	org.gjt.sp.jedit.util
org.gjt.sp.jedit.synatx	org.gjt.sp.jedit.synatx
org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.textarea
installer	installer

Figure 9: Low Cohesion Modules

Q3) Which jEdit modules have high complexity?

Answer : In order to determine the complexity we selected three metrics, Cyclomatic Complexity (CC), Depth of Inheritance (DIT) and Weighted Method per Class (WMC). High complexity results in high maintainability. In most of the modules over different ver-



Metrics		Attributes
CBO	RFC	High Coupling Packages
org.gjt.sp.jedit.util	org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.util
org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.bsh
org.gjt.sp.jedit.io	org.gjt.sp.jedit.buffer	org.gjt.sp.jedit.textarea
org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.browser	org.gjt.sp.jedit.io
org.gjt.sp.jedit.buffer	org.gjt.sp.jedit.syntax	org.gjt.sp.jedit.buffer
		org.gjt.sp.jedit.browser

Figure 7: Highly Coupled Modules

sions the CC values is stable which retain its maintainability. For these modules of jEdit *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.browser, org.gjt.sp.jedit.print, org.gjt.sp.jedit.syntax, org.gjt.sp.jedit.search* the CC values are high and the highest values for these modules are 5.31, 4.82, 4.18, 4.1, 3.96 thus with the increase in CC values maintenance increases refer in Figure 10.

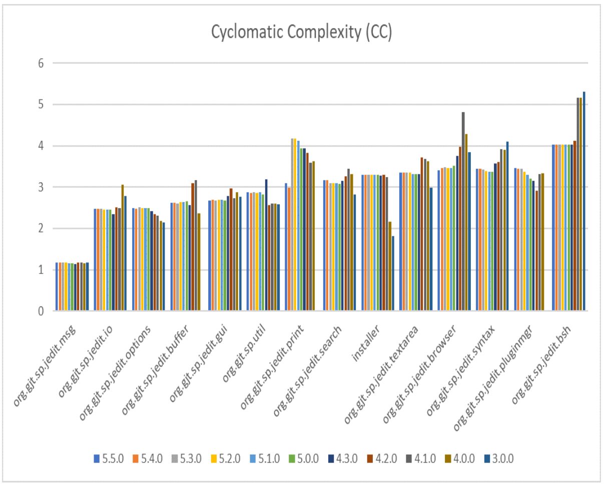


Figure 10: CC metrics for jEdit versions

For these modules of jEdit *org.gjt.sp.jedit.msg, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.browser, org.gjt.sp.jedit.options, org.gjt.sp.jedit.textarea* the DIT values are high and the highest values for these modules are 1.18, 0.5, 0.49, 0.4, 0.39 thus with the increase in DIT values maintenance increases refer in Figure 11 [20]. The DIT values are almost all stable when moving to the latest versions.

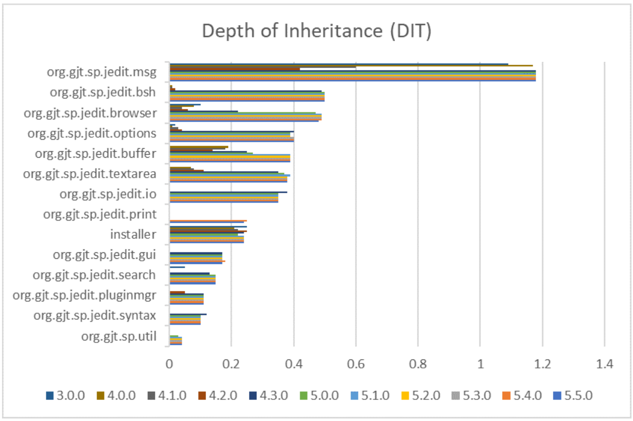


Figure 11: DIT metrics for jEdit versions

For these modules of jEdit *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.textarea, org.gjt.sp.jedit.syntax, org.gjt.sp.jedit.buffer, installer* the WMC values are high and the highest values for these modules are 60.82, 51.27, 30.64, 27.45, 21.42 thus with the increase in WMC values maintenance increases refer in Figure 12. When compared to the other modules WMC values of these modules decreased from the previous versions that helps to retain the maintainability.

The modules *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.msg, org.gjt.sp.jedit.browser, org.gjt.sp.jedit.textarea, org.gjt.sp.jedit.print, org.gjt.sp.jedit.syntax* have high complexity refer in Figure 13. Hence high complexity leads to poor maintainability.

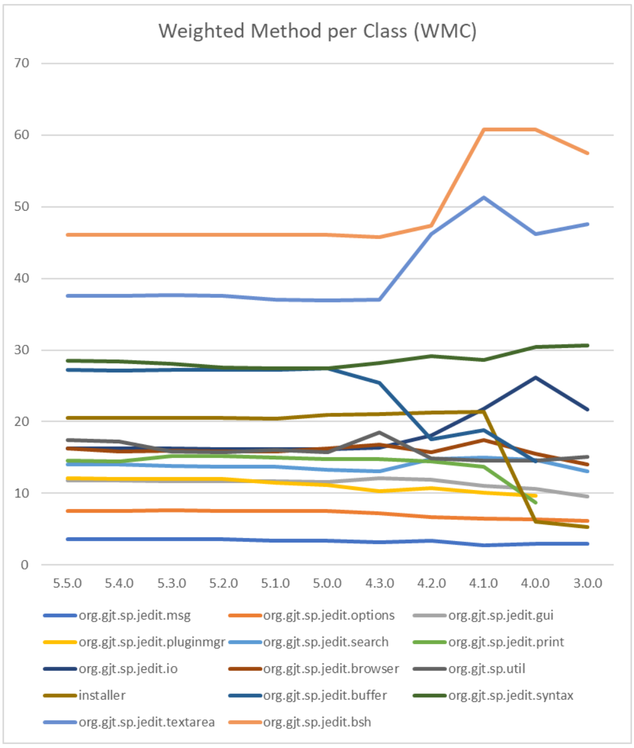


Figure 12: WMC metrics for jEdit versions

Q4) Which jEdit modules have high degree of understandability?
Answer : In order to determine the understandability we selected five metrics, Efferent Coupling (Ce), Afferent Coupling (Ca), Instability (I), Comment Ratio and Commented Lines of Code (CLOC).High understandability results in high maintainability.The Ce values increased when moving to new versions.For these modules of *jEdit org.gjt.sp.jedit.options, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea, org.gjt.sp.jedit.browser, org.gjt.sp. jedit.search* the Ce values are high and the highest values for these modules are 208, 203, 147, 134, 130 thus with the increase in Ce values maintenance increases refer in Figure 15.

For these modules of jEdit *org.gjt.sp.util, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea, org.gjt.sp. jedit.options, org.gjt.sp.jedit.io* the Ca values are high and the highest values for these modules are 141, 133.5, 118, 86, 83 thus with the increase in Ca values maintenance increases. The Ce values has been increased over the latest versions of jEdit refer in Figure 14.

For these modules of jEdit *org.gjt.sp.jedit.print, org.gjt.sp.jedit.search, org.gjt.sp.jedit.options* the I values are high and the highest values for these modules are 0.92, 0.81, 0.78 thus with the increase in I values maintenance increases refer in Figure 16.

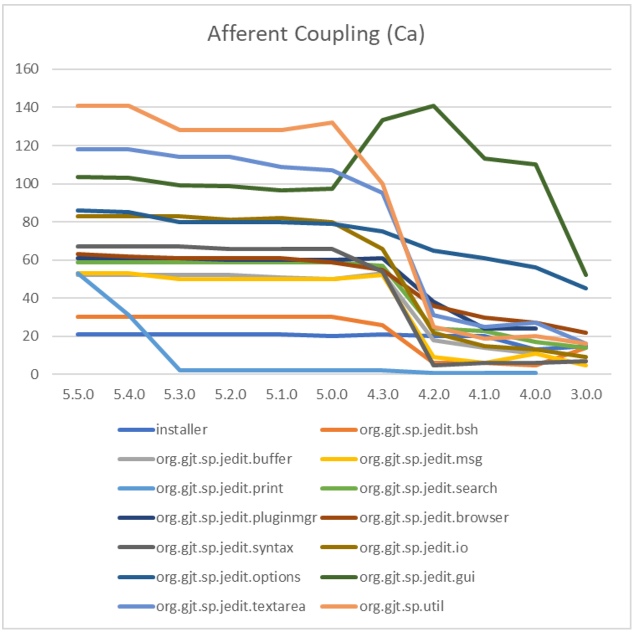


Figure 14: Ca metrics for jEdit versions

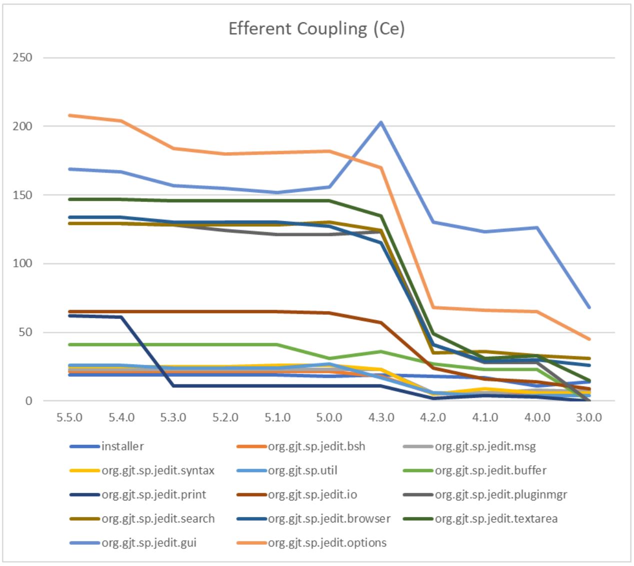
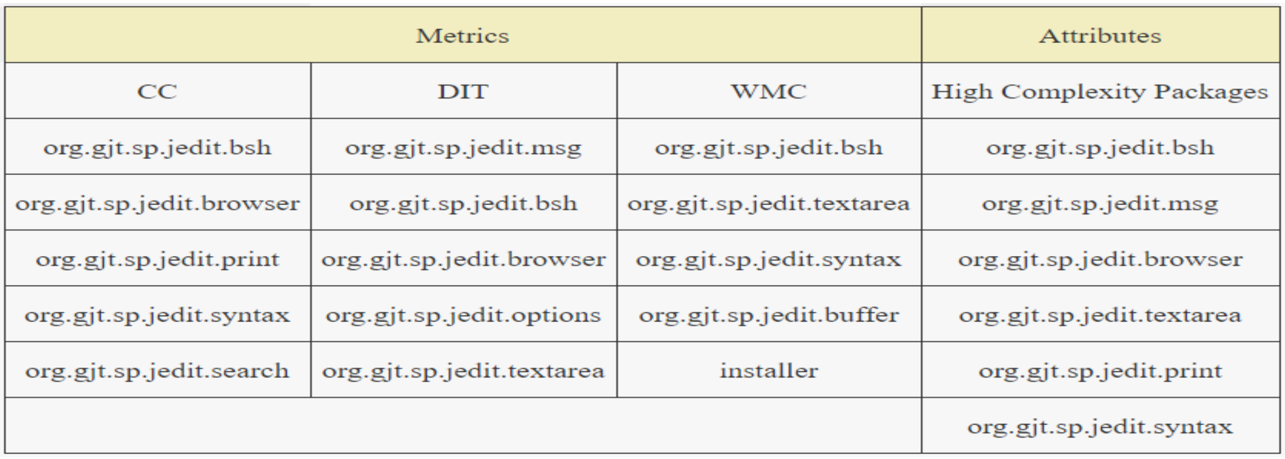


Figure 15: Ce metrics for jEdit versions

By analyzing the older versions of jEdit the values of instability has been increased when compared to the latest releases of jEdit, but the above modules have instability values higher that are difficult to maintain.

For these modules of jEdit *org.gjt.sp.jedit.msg, org.gjt.sp.jedit.io, org.gjt.sp.jedit.buffer, org.gjt.sp.util* the Comment Ratio values are high and the highest values for these modules are 66.78, 46.1, 43.61, 40.98 thus with the increase in Comment Ratio values maintenance increases. The comment ratio values are almost stable over all the versions of jEdit with retains maintainability refer Figure 17.



Metrics			Attributes
CC	DIT	WMC	High Complexity Packages
org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.msg	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.bsh
org.gjt.sp.jedit.browser	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.msg
org.gjt.sp.jedit.print	org.gjt.sp.jedit.browser	org.gjt.sp.jedit.syntax	org.gjt.sp.jedit.browser
org.gjt.sp.jedit.syntax	org.gjt.sp.jedit.options	org.gjt.sp.jedit.buffer	org.gjt.sp.jedit.textarea
org.gjt.sp.jedit.search	org.gjt.sp.jedit.textarea	installer	org.gjt.sp.jedit.print
			org.gjt.sp.jedit.syntax

Figure 13: More complexed Modules

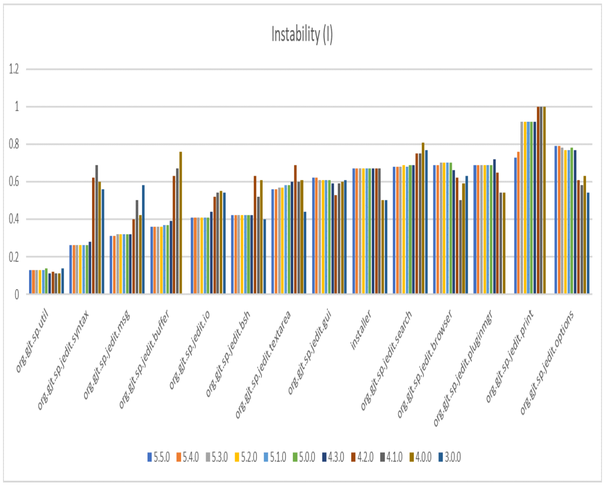


Figure 16: I metrics for jEDit versions

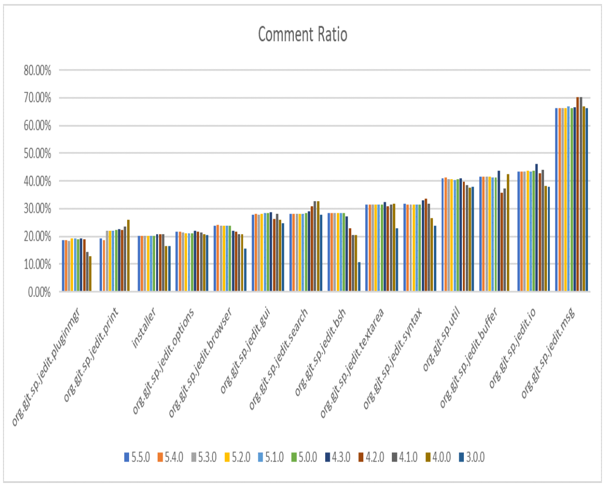


Figure 17: Comment Ratio metrics for jEdit metrics

For these modules of jEdit *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* the CLOC values are high and the highest values for these modules are 10143, 6822, 5895 thus with the increase in CLOC values maintenance increases refer Figure 18. The CLOC values have been increased over the latest releases of jEdit which serves as a great source for code understandability.

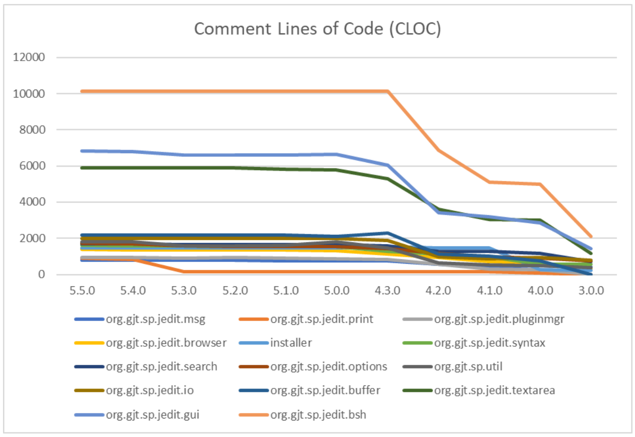
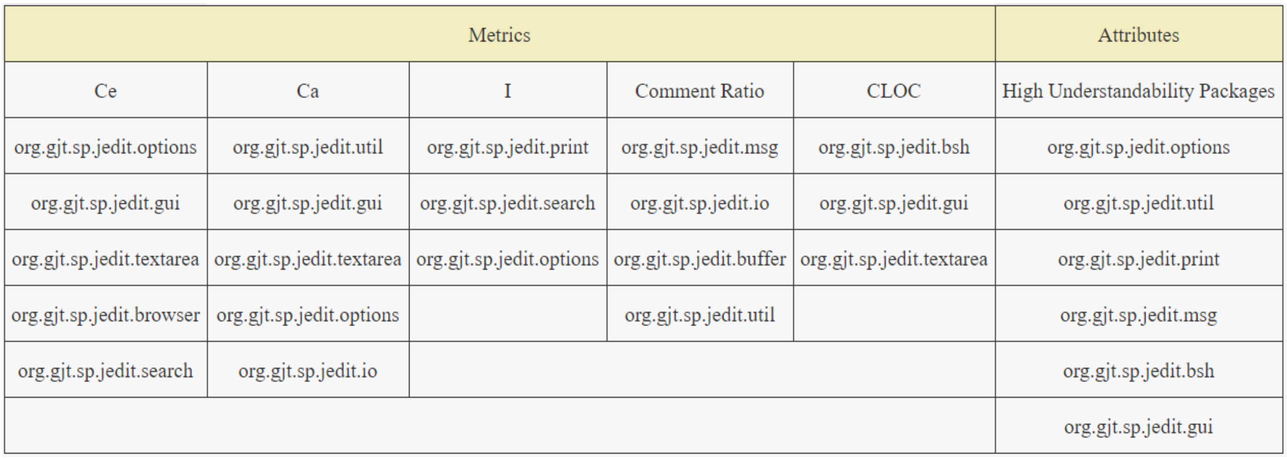


Figure 18: CLOC metrics for jEdit metrics

The modules *org.gjt.sp.jedit.options, org.gjt.sp.util, org.gjt.sp.jedit.print, org.gjt.sp.jedit.msg, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui* have high understandability refer in Figure 19. Hence high understandability leads to low maintainability.

Q5) Which jEdit modules have been modified over different versions regarding size?
Answer : In order to determine the size we selected three metrics, Lines of Code (LOC), Number of Children (NOC) and Number of Methods (NOM). More change in size results in high maintainability. For these modules of jEdit *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* the LOC values are high and the highest values for these modules are 35646, 24454, 18729 thus with the increase in LOC values maintenance increases refer Figure 20. For



Metrics					Attributes
Ce	Ca	I	Comment Ratio	CLOC	High Understandability Packages
org.gjt.sp.jedit.options	org.gjt.sp.jedit.util	org.gjt.sp.jedit.print	org.gjt.sp.jedit.msg	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.options
org.gjt.sp.jedit.gui	org.gjt.sp.jedit.gui	org.gjt.sp.jedit.search	org.gjt.sp.jedit.io	org.gjt.sp.jedit.gui	org.gjt.sp.jedit.util
org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.options	org.gjt.sp.jedit.buffer	org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.print
org.gjt.sp.jedit.browser	org.gjt.sp.jedit.options		org.gjt.sp.jedit.util		org.gjt.sp.jedit.msg
org.gjt.sp.jedit.search	org.gjt.sp.jedit.io				org.gjt.sp.jedit.bsh
					org.gjt.sp.jedit.gui

Figure 19: Low Understandability Modules

all the jEdit modules in all versions the LOC values increased because of the addition of new features to the modules.

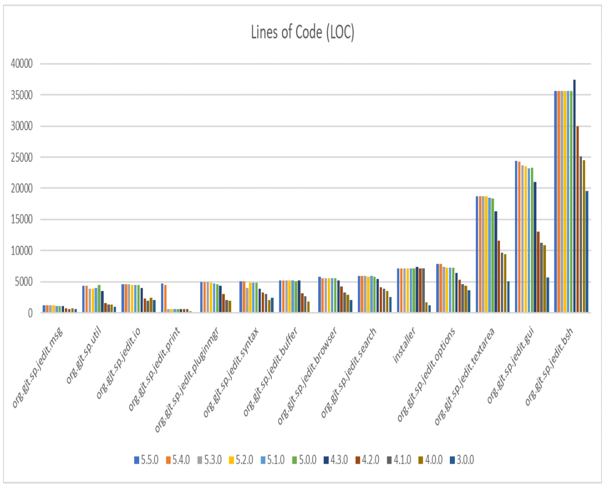


Figure 20: LOC metrics for jEdit versions

For these modules of jEdit *org.gjt.sp.util, org.gjt.sp.jedit.buffer, org.gjt.sp.jedit.bsh* the NOC values are high and the highest values for these modules are 0.54, 0.52, 0.57 thus with the increase in NOC values maintenance increases refer Figure 21 [20]. The NOC values fluctuated in the older versions of jEdit where as it remains stable in the latest versions.

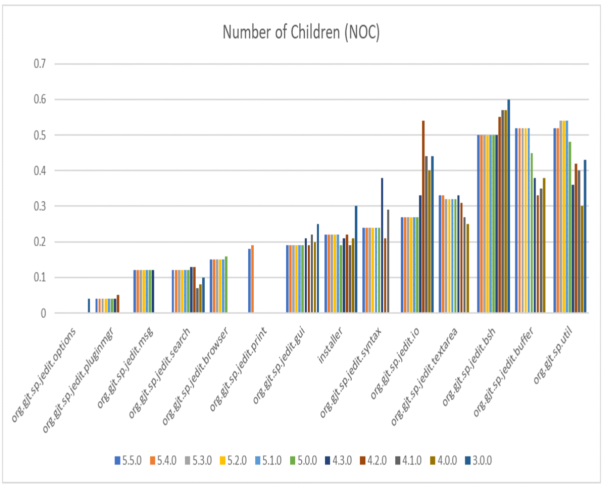
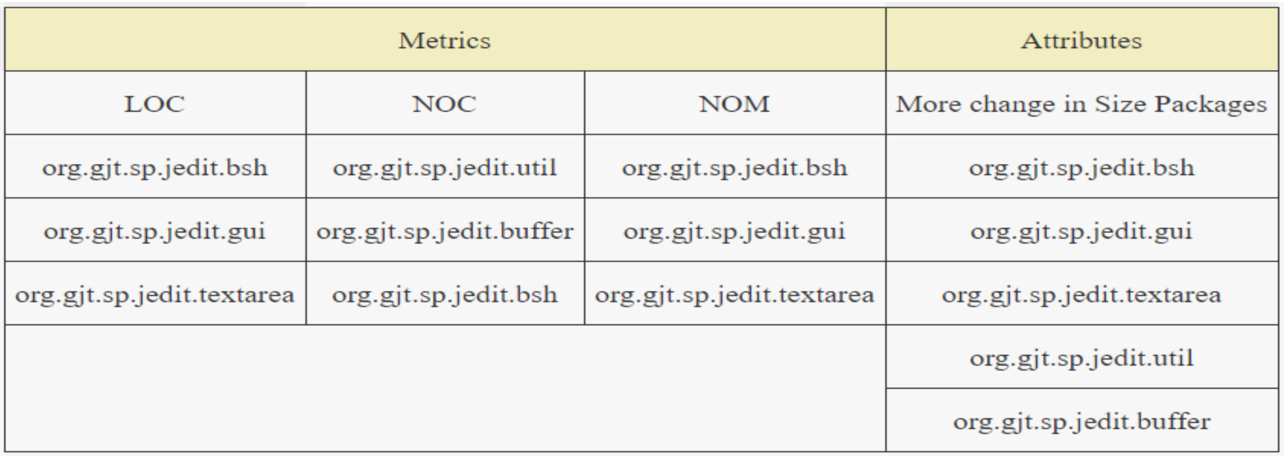


Figure 21: NOC metrics for jEdit versions

For these modules of jEdit *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* the NOM values are high and the highest values for these modules are 1471, 1271, 847 thus with the increase in NOM values maintenance increases refer Figure 22 [20]. By analyzing the modules, NOM values in the older versions of jEdit are lesser where it is increased in the latest versions.



Metrics			Attributes
LOC	NOC	NOM	More change in Size Packages
org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.util	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.bsh
org.gjt.sp.jedit.gui	org.gjt.sp.jedit.buffer	org.gjt.sp.jedit.gui	org.gjt.sp.jedit.gui
org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.bsh	org.gjt.sp.jedit.textarea	org.gjt.sp.jedit.textarea
			org.gjt.sp.jedit.util
			org.gjt.sp.jedit.buffer

Figure 23: Large Size Modules

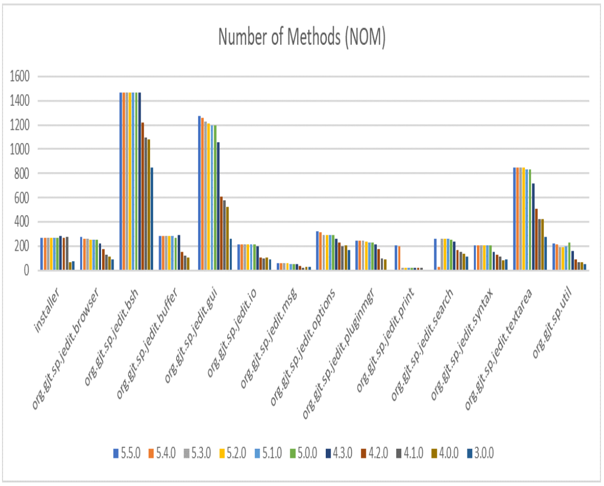
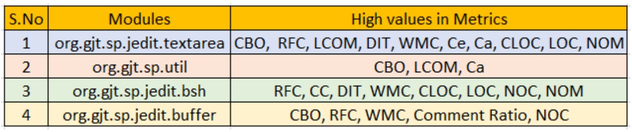


Figure 22: NOM metrics for jEdit versions

The modules *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea, org.gjt.sp. util, org.gjt.sp. jedit.buffer* have more change in size packages refer Figure 23. Hence modules with change in size have poor maintainability.

4.2.1 Reflection Points

1) Which of the product modules will be more difficult to maintain and Why?
Answer : The difficulty in maintenance can be identified by the attributes such as code structure, code complexity, code understandability, code size. The modules that are poor in their structure, having high complexity, large in size and low degree of understandability are more difficult to understand. After analyzing the metric values and answering the GQM questions we found that the modules *org.gjt.sp.jedit.textarea, org.gjt.sp.util, org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.buffer* refer in Figure 24 are more difficult to maintain than other modules. When proceeding to the newer versions these modules size, complexity, instability has been increased. These modules have low cohesion and high coupling values which results in a poor structure. With the increase in size of the code in each module the NOC, NOM values increases thereby resulting in poor understandability. Hence these product modules are more difficult to maintain in terms of structure, complexity, understandability and size.



S.No	Modules	High values in Metrics
1	org.gjt.sp.jedit.textarea	CBO, RFC, LCOM, DIT, WMC, Ce, Ca, CLOC, LOC, NOM
2	org.gjt.sp.util	CBO, LCOM, Ca
3	org.gjt.sp.jedit.bsh	RFC, CC, DIT, WMC, CLOC, LOC, NOC, NOM
4	org.gjt.sp.jedit.buffer	CBO, RFC, WMC, Comment Ratio, NOC

Figure 24: Modules that are more difficult to maintain

2) What can be done to improve the maintainability of these modules?
Answer : The above mentioned modules are difficult to maintain. In order to improve the maintainability of these modules the attributes such as code structure, code complexity, code understandability and code size should be well maintained. In order to improve the structure of the code the modules should have high cohesion and low coupling. Tight coupling has its effect on maintainability. Hence in order to reduce the coupling the logic should be mentioned separately in their classes or methods which is helpful while making changes and it reduces the chances of breakages.Hence the structure of the code can be increased. In order to improve the complexity the dependencies should be removed. The complexities can be reduced by adding dummy packages and making changes to them. By adding comments to the code makes it more understandable. The code size should be reduced by analyzing and cut shorting the logic in a simpler way. Hence by making these changes the maintainability of the modules can be improved. And there are also many methods that helps to improve the maintainability of the modules.

3)How various internal product attributes

related to module size. Do modules with large size have poor structure as well?
Answer : The internal attributes that are analyzed in this project are code structure, code complexity, code understandability. All the internal attributes are related to size of the module. As the size of the modules increases the code structure turns bad.This can be verified by the results of the packages *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.textarea* refer in Figure 26, 25.

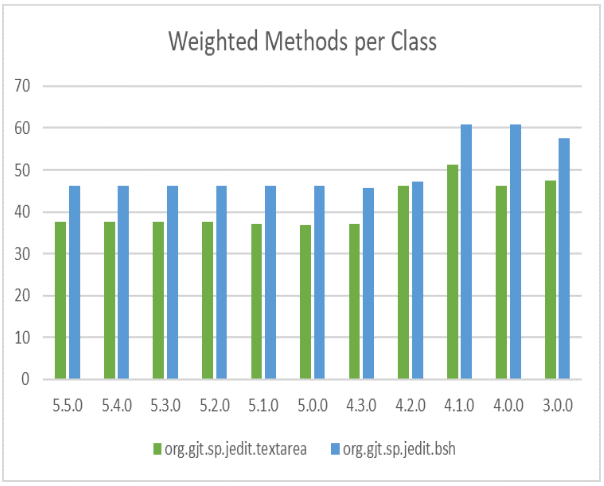


Figure 25: Size of Modules in terms of WMC

With the increase in the size number of classes and methods increases resulting in increase in complexity. Increase in code size also increases the connections between classes and methods which makes the modules tough to understand.

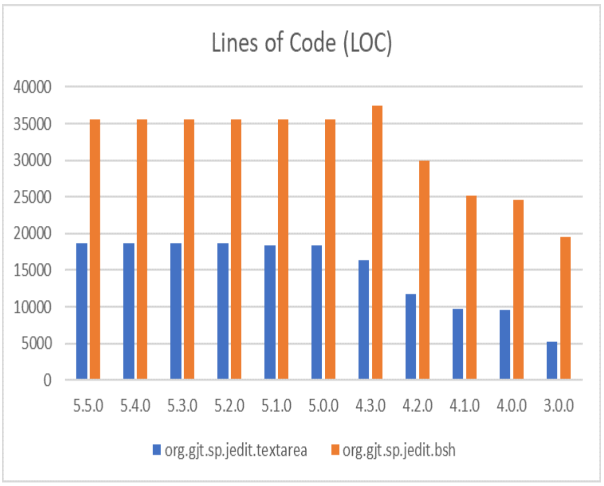
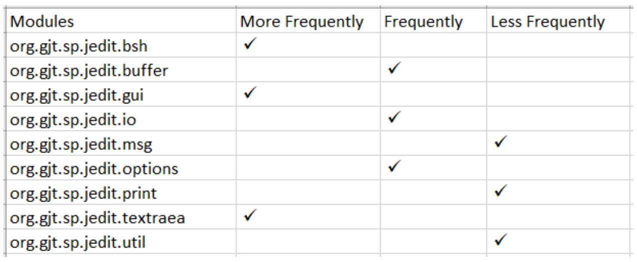


Figure 26: Size of Modules in terms of LOC

4)Identify the modules that were frequently updated, substantially and observe the newly added modules posses poor/better code structure in 11 versions of jEdit.

Answer : From the analyses we made that the frequently updated modules that are based on metrics Lines of Code (LOC), Commented Lines of Code (CLOC) which are been changed from the different versions of jEdit. The modules *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* in LOC are changed drastically from the earlier versions of jEdit to latest releases. While in CLOC the same above modules *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* are drastically increased. With the increase in the metrics LOC make code more unstable which leads to increase in complexity. With the frequent modification, these modules have poor code structure due to high coupling and poor cohesion .

- The modules that are more frequently updated are *org.gjt.sp.jedit.bsh, org.gjt.sp.jedit.gui, org.gjt.sp.jedit.textarea* refer Figure 27.
- The modules that are frequently updated are *org.gjt.sp.jedit.buffer, org.gjt.sp.jedit.io, org.gjt.sp.jedit.options* refer Figure 27.
- The modules that are less frequently updated are *org.gjt.sp.jedit.msg, org.gjt.sp.jedit.print, org.gjt.sp.jedit.util* refer Figure 27.



Modules	More Frequently	Frequently	Less Frequently
org.gjt.sp.jedit.bsh	✓		
org.gjt.sp.jedit.buffer		✓	
org.gjt.sp.jedit.gui	✓		
org.gjt.sp.jedit.io		✓	
org.gjt.sp.jedit.msg			✓
org.gjt.sp.jedit.options		✓	
org.gjt.sp.jedit.print			✓
org.gjt.sp.jedit.textraea	✓		
org.gjt.sp.jedit.util			✓

Figure 27: Frequently Updated Modules

5 Discussion

5.1 Related Work

The quality model basically describes the quality in terms of a software metrics and attributes. No doubt these attributes affects the quality of software. In this paper the author have performed an experiment to propose the changeability prediction model that is based on the structure of ISO/IEC 9126 for the better software maintainability. They have used the multi-layer perception as a classifier method and data set of 137 JAVA classes from the jEdit the open source project. The main purpose of this paper is accessing the changeability level of JAVA code but unfortunately the model they have derived inn this project cannot predict the changeability of classes at level 1 and level 2 [21].

For any software the key quality attribute is the software maintainability because the success of software is dependent on this. Now a days the software system is going very complex and if the size of the software increase with the passage of time then this complexity increase more for any software. In this paper the author have utilized the data mining of some new predictor metrics instead of traditionally used software metrics to predicting the maintainability of the software systems. This prediction model was constructed by using the static code metrics of 4 different open source systems. The result shows that accurate model can be construct for software maintainability by doing the data mining of the code metrics [22].

Software development is growing very fast with the time and more complex applications are under development and some are completed, the new build systems are complex. Researchers identified different algorithms to measure the maintainability. jedit is an open source software which have multiple versions all versions are experimental base. In this research they continue to analyze the report which is generated on basis of Choosing the 41 version pairs of jEdit. These reports show the effect of maintainability and coupling. The processes was based on 4 internal attributes that are structural complexity, inheritance, cohesion and coupling [23].

Software quality measurement is one of the most debatable and burning research topic now a days. There are many models to measure the quality factors of the software especially maintainability. Maintainability includes all the changes that have been made at different stages after delivery and the quality measures ensure the state of the modified software . Researchers morteza Aadi and Hassan srashidi contribute their efforts and develop the new model for measuring the maintainability of the software. New model is evaluated on PHP frame work. Result shows that newly proposed model is much efficient because of using and calculating more metrics. In this experiment to show the behavior of the maintenance Hidden Markov model was used, According to the result accurate time period to meet the threshold [24].

5.2 Comparision of Results

- In [21] this study the author evaluated the maintainability by predicting the changeability whereas in our study the maintainability of the software is evaluated using internal attributes which mainly focus on the quality.

- In [22] this study the author focused on new data mining metrics in order to determine the maintainability of the software. He analyzed four open source software systems.In our study we evaluated the maintainability of only jEdit open source system by using internal attributes and the metrics related.

- In [23] this study the researchers measured the maintainability of jEdit versions using different metric algorithms.They also used four internal attributes in order to evaluate the maintainability. In our study we measured the maintainability of the jEdit system using metrics that are extracted from the tool.We used code structure, code complexity, code size and code understandablity as internal attributes.

- In [24] this study the researchers evaluated the sub characteristics of the maintainability using metrics that are related to the sub category. The maintainability is measured for PHP framework which is taken as system under study in this paper. In our study we measured mainatainability for jEdit which an open source text editor written in Java. The metrics we choose to evaluate the maintainability depends upon the internal attributes and questions formulated.

5.3 Reflection on the Project

- In this project we learn how to apply the GQM framework. We also know how to formulate the question related to goal and we also know how to classify the metrics for the questions formulated.

- In this project, in order to do an empirical study i.e case study we analyzed various previous studies and gain knowledge about how to select and conduct an empirical study.

- We gained more knowledge about selecting metrics in order to evaluate the quality of the software. During this analyses we learned a lot about metrics we referred so many studies in order to justify the metric for specific attribute.

- In order to evaluate the measures we used different statistical measures and visualization methods. Hence we gain more information about different statistical measures and also show up the results in different visualization methods.

- From this project we gained information about how to measure the maintainability of the system and how the metrics are used in practical.

6 Threats To Validity

- In order to achieve the goal of the project we selected certain metrics for analysis.The selected metrics may or may not be the suited ones for the attribute but we provided the selection of metric with justification.

  There may be several other metrics that describe the attribute where the results can be differed.

- The results can be variant if we use other types of metrics.The metrics selection is limited to only few.Hence there can be some changes with the results if we use any another types of metrics.By selecting only specific type of modules the effect of attributes can be changed.The modules selection is also limited in order to get better results.
- The selection of tool can alter the results.Different tools are providing different values for metrics.Hence this can be a threat and the results can be varied.

7 Conclusion

In this study, the last eleven versions of jEdit software system are studied and the modules that are difficult to maintain are obtained. The metrics reloaded is used to extract the metric values and analyses is performed for the values. The GQM framework is used to formulate the questions and answer them. The analyzed results are used to answer the questions.

The modules that have bad structure,high complexity, large size and poor understandability are observed and reported in this study.

References

[1] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*, IEEE, 1999, pp. 242–249 (cit. on p. 1).

[2] M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *Journal of systems and software*, vol. 76, no. 2, pp. 147–156, 2005 (cit. on p. 1).

[3] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002 (cit. on p. 1).

[4] R. Marinescu, "Measurement and quality in object-oriented design," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*, IEEE, 2005, pp. 701–704 (cit. on p. 1).

[5] S. Parthasarathy and N. Anbazhagan, "Analyzing the software quality metrics for object oriented technology," *Inform. Technol. J*, vol. 5, pp. 1053–1057, 2006 (cit. on p. 1).

[6] J. R. Feagin, A. M. Orum, and G. Sjoberg, *A case for the case study.* UNC Press Books, 1991 (cit. on p. 2).

[7] L. Krusenvik, *Using Case Studies as a Scientific Method: Advantages and Disadvantages.* 2016 (cit. on p. 2).

[8] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach.* CRC press, 2014 (cit. on p. 3).

[9] V. R. B. G. Caldiera and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994 (cit. on p. 3).

[10] A. Sharma and S. K. Dubey, "Comparison Comparison of Software Quality Metrics Software Quality Metrics Software Quality Metrics for Object-Oriented Oriented Oriented System," (cit. on pp. 3–5).

[11] P. Berander and P. Jfffdfffdnsson, "A goal question metric based approach for efficient measurement framework definition," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ACM, 2006, pp. 316–325 (cit. on p. 3).

[12] W. Li, "Another metric suite for object-oriented programming," *Journal of Systems and Software*, vol. 44, no. 2, pp. 155–162, 1998 (cit. on pp. 3, 4).

[13] *OO Software Metrics - Class Level Java metrics - LCOM, UWCS, Coupling, RFC, MPC, CBO, Fan In, Fan Out* (cit. on pp. 4, 5).

[14] G. Muketha, C. O. Ondago, and K. O. Ondimu, "Complexity metrics for measuring the understandability and maintainability of business process models using goal-question-metric (GQM)," 2008 (cit. on p. 4).

[15] M. Sarker, "An overview of object oriented design metrics," *From Department of Computer Science, Ume\a a University, Sweden*, 2005 (cit. on pp. 4, 5).

[16] *Virtual Machinery - Sidebar 3 - WMC, CBO, RFC, LCOM, DIT, NOC - 'The Chidamber and Kemerer Metrics'* (cit. on p. 5).

[17] S. S. Stevens, "On the theory of scales of measurement," 1946 (cit. on p. 5).

[18] M. Xenos, D. Stavrinoudis, K. Zikouli, *et al.*, "Object-oriented metrics-a survey," in *Proceedings of the FESMA*, 2000, pp. 1–10 (cit. on pp. 5, 6).

[19] P. Singh, S. Verma, and O. P. Vyas, "Cross company and within company fault prediction using object oriented metrics," *International Journal of Computer Applications*, vol. 74, no. 8, 2013 (cit. on p. 6).

[20] J. Lindell and M. Hagglund, "Maintainability metrics for object oriented systems," *Software Quality*, 2004 (cit. on pp. 8, 9, 12).

[21] S. Rongviriyapanish, T. Wisuttikul, B. Charoendouysil, *et al.*, "Changeability prediction model for java class based on multiple layer perceptron neural network," in *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, IEEE, 2016, pp. 1–6 (cit. on pp. 14, 15).

[22] A. Kaur, K. Kaur, and K. Pathak, "Software maintainability prediction by data mining of software code metrics," in *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)*, IEEE, 2014, pp. 1–6 (cit. on p. 15).

[23] A. Molnar and S. Motogna, "Discovering maintainability changes in large software systems," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, ACM, 2017, pp. 88–93 (cit. on p. 15).

[24] M. Asadi and H. Rashidi, "A Model for Object-Oriented Software Maintainability Measurement," *International Journal of Intelligent Systems and Applications*, vol. 8, no. 1, p. 60, 2016 (cit. on p. 15).