Predicting the Behaviour of Open Source Software using Object Oriented Metrics

Uttamjit Kaur
Department of Computer Science
GIMET
Amritsar

Gagandeep Singh
Department of Computer Science
GIMET
Amritsar

ABSTRACT
The aim of this thesis is to study the relationship between maintainability and metrics like lines of code, cyclomatic complexity of open source software. The behavior of open source software can be predicted by calculating maintainability index and reliability index. Prediction of maintainability index will help in better management and maintenance of object oriented software thus reducing the cost of maintenance. The main objective of this thesis is to calculate different metrics like Lines of Code, Cyclomatic Complexity, and Maintainability Index. The study also includes the comparison of these metrics plotted over various open source software. This report summaries the theory about maintainability of different software's and the impact of these metrics on its maintainability. Open Source Software used for study in this thesis is SweetHome3D, FindBugs, Jacob and Jfree. Analyst4J tool is used to calculate values of metrics used for studying the maintainability of SweetHome3D, FindBugs, Jacob and Jfree.The case study has shown that applying software metrics that would measure the different aspects of software would be useful in analyzing, studying and improving the maintainability of software.

General Terms
Software Metrics, OSS.

Keywords
Software Metrics, Object-Oriented Software, OSS, SweetHome3D, FindBugs, JFree and JACOB.

1. INTRODUCTION
Software metrics play an important role in Analyzing some aspect of a Software or product generated during a software project. In general, Software metrics can be divided into three categories. These are product metrics, process metrics and project metrics. Process Metrics: This Metrics used to explain the characteristics of the software product i.e.: Size, Complexity, performance and the level of Quality. Project Metrics: This Metrics is used to improve the software development process and maintenance. Example: Defect removal during the development stage and response time to fix the process. Project Metrics: It depicts the project characteristics and execution. Example: No. of Software Developers, cost, productivity and schedule.

Rapid developments of large scaled software have evolved complexity that makes the quality difficult to control. Software Metrics requires Software Quality over the control of successful execution. Software applications are more complex that leads to software failure resulting in software damage [2]. The metrics focus on internal parts that reflect the complexity of each individual entity, i.e. classes or Methods. The metrics focus on external parts measure the interactions among entities, i.e. inheritance or coupling. Software metrics describe as: "The continuous application of measurement-based techniques to the software development process and its products to supply timely or meaningful techniques together for management information (MI) "[4].

2. OBJECT-ORIENTED
According to Object Oriented approaches, concept is important to define object. An Object is defined by a state (set of properties) and a behaviour (set of operation).A class is the specification of the object, it is the basic prototype from which the object are created. The methods are the operation that can be carried out in a class. The attribute represent the properties of a class. Object-oriented design used to define Well-structured software, as they easy to test and maintain. As, Object-Oriented approaches does not ensure the quality of software nor errors removed during development and maintenance phases. However, different Object-oriented are written in the literature. As they hold the Object-oriented design properties i.e.: Coupling, Complexity, Cohesion and Inheritance to enhance the software quality. The metrics presented here are: method related metrics, class related metrics, inheritance metrics, metrics measuring coupling and metrics measuring general (system) software production characteristics. In this paper six metrics are considered for optimization. These metrics are: DIT (Depth of Inheritance), CBO (Coupling Between Objects), LCOM-CK (Lack of Cohesion of Methods) (as originally proposed by Chidamber&Kemerer), WMC (Weighted Methods per Class), TCC (Tight Class Cohesion), MI (Maintainability Index). The software metrics presented here are grouped into complexity, size and dependency metrics. The metrics are classified into these categories as to identify the attributes the metrics can provide insight into.

2.1 Line of Code (Size metric)
Size metric is the most common measures used to assess the memory requirements, the effort and the development time that is necessary. It has been argued that poor size predication has been a major cause for software failures. This metric is very important in determining the cost that is correlated with development. Additionally, it is useful in preparing schedules and also estimation of efforts required. Complexity is a function of size, which can greatly affect the design flaws and hidden defects resulting in quality problems, cost overruns, and schedule changes. Complexity shall be constantly monitored, measured and controlled. Any impact on size metrics can be shown in the effort performance criterion. The effort metric predicts the effort needed to maintain a project .As its name indicates, the notion behind Lines of code basically is to count the number of lines of source code of a certain software project. Even though it is a simple, it is a strong metric suite to assess the complexity of different software entities.

2.2 McCabe Cyclomatic Complexity (MCC)

Cyclomatic complexity gives the number of paths that may be taken when a program is executed. Methods with a high Cyclomatic complexity tend to be more difficult to understand and maintain. The Cyclomatic complexity metric measures the complexity of a module's decision structure. It can be calculated by counting the number of linearly independent paths through a function or set of functions [6]. It is useful in a situation where higher Cyclomatic complexities associate with greater testing and maintenance requirements. Commonly Complexities measure of higher values corresponds to higher error rates.

2.3 CBO

Coupling between Object classes is the number of classes to which a class is couple.

$$\text{CBO} = \frac{\text{Number of links}}{\text{Number of classes}}$$

Numbers of links are number of classes used associations, use links for all the package's classes. A class used several times by another class is only counted once. Numbers of classes are number of classes of the package, by recursively processing sub-packages and classes, for the UML modelling project, this variable represents, therefore, the total number of classes of the UML modeling project. CBO for a class is a count of the number of other classes to which it is coupled. The theoretical basis behind this metric is to calculate the number of the peripheral classes whose members are called or used as types by members of the current class. To explain it in other words," CBO refers to the number of coupling between classes. When a class let's say, *class1*calls the member functions of another class, *class 2;* coupling will occur. The smaller the CBO, the less the class affects other classes. This means that the more independent the class is, the lesser the probability that an alteration could occur to a depending class; and therefore less maintenance effort may be needed. Concurrently, the bigger the coupling between objects is, the slighter the reusability may the class become. This metric is useful in discovering a situation where excessive coupling limits the availability of a class for reuse, and also results in greater testing and maintenance efforts.

2.4 DIT

Depth of Inheritance Tree is the maximum inheritance path from the class to the root class. The DIT measures the inheritance level upon which a class was built. The value can be achieved by calculating the maximum number of levels in each of the class's inheritance paths. While reuse potential goes up with the number of root programs, so does design complexity, due to more methods and classes being involved. Some studies have shown that higher DIT rate correspond with larger error density and lower quality. The smaller the DIT, the more abstract and simpler the class would become. While the more a class inherits, the more difficult to understood the design.

2.5 Response for a Class (RFC)

The RFC metric measures the general complexity of the calling hierarchy of the methods. The value for RFC can be calculated by counting the methods of a class and the methods that they directly call. Larger RFC counts are commonly correlated with increased testing requirements. Since it includes methods called from outside the class, it can also be a measure of the possible communication between the class and other classes. If the number of methods that can be invoked in response to a message is large, the testing and debugging process of the class would become more difficult and time consuming since it requires very good knowledge of how the methods are interconnected to each other.

2.6 WMC

Weighted Methods per Class (using Cyclomatic Complexity as method weight) is the sum of weights for the methods of a class. It is an indicator of how much effort is required to develop and maintain a particular class. A class with a low WMC usually points to greater polymorphism. A class with a high WMC, indicates that the class is complex (application specific) and therefore harder to reuse and maintain. The lower limit for WMC in Refractor IT is default 1 because a class should consist of at least one function and the upper default limit is 50.

The WMC measures some features of the scope of the methods building a class. It computes the weight of each method, that is, the value of WMC can be attained by summing up the weighted methods of the class. After obtaining WMC value, it can be used to measure the complexity of the decision structure within the methods. It can be helpful in a circumstance where higher WMC values associate with enlarged development, testing and maintenance efforts. Because of inheritance, the testing and maintenance efforts for the derived classes could also increase as a result of higher WMC for a parent class.

2.7 LCOM - Lack of Cohesion of Methods

Cohesion is the degree to which methods within a class are related to one another and work together to provide well bounded behavior. LCOM uses variable or attributes to measure the degree of similarity between methods. We can measure the cohesion for each data field in a class; calculate the percentage of methods that use the data field. LCOM measures how widely member variables are used for sharing data between member functions. It is calculated by counting the pairs of class methods that don't access any of the same class variables reduced by the number of pairs that do. In other words, the "Lack of Cohesion in Methods metric is a measure for the number of not connected method pairs in a class representing independent parts having no cohesion. It represents the difference between the number of method pairs not having instance variables in common, and the number of method pairs having common instance variables." A higher LCOM indicates lower cohesion. This relates with weaker encapsulation, and is a pointer that the class is a candidate for disaggregation into subclasses.

2.8 Maintainability Index (MI)

The MI [8] is a composite metric, based on several metrics. It is based on following metrics:

1. Halstead Volume (HV) metric
2. Cyclomatic Complexity (CC) metric
3. Average number of lines of code per module (LOC)
4. Percentage of comment lines per module (COM).

Halstead Volume is a composite metric based on the number of (distinct) operators and (distinct) operands in source code.Cyclomatic Complexity is the number of linearly independent paths through a program. Lines of code is a software metric used to measure the size of the source code. Comments per module are the number of comment lines in the

source code of a module. The formula for maintainability index is:

MI = 171 - 5.2 * In(aveV) - 0.23 * aveV(g') - 16.2 *In (aveLOC)

where

aveV = average Halstead Volume V per module

aveV(g') = average cyclomatic complexity per module

aveLOC = the average count of lines of code (LOC) per module.

3. OPEN SOURCE SOFTWARE

Today, open source software (OSS) provides a lot of services and products for companies, industry, governments, and education organizations or commercial software products. There is a lot of successful OSS, like Apache, PHP, Nginx, MySQL, and MariaDB. OSSD is a kind of distributed software development base using the peer-review technique, and the development team is distributed across the world in different time zones.This increases the difficulty of achieving quality assurance (QA), as do the risky development practices in open source software development (OSSD), such as unclear requirements elicitation, the ad hoc development process, the little attention paid to quality assurance and documentation, and poor project and quality management. The development must consider much more than writing the code somehow. Every organization looks for very good architecture; reliable, testable, and maintainable code; and methodologies to support and maintain their software. Open source software (OSS) is a software product with the source code made public so that anyone can read, analyze, and change or improve the code. The use of this software is under a license, like Apache, GNU, MIT, Mozilla Public, and Eclipse Public License. Open source software development (OSSD) provides high quality assurance through user testing and peer reviews. The quality of these products depends on the size of the product community. The author is discussing about the stakeholders of the OSS community, the quality assurance frameworks and models proposed in some studies, some statistics about OSS, the problems that affect the quality of OSSD, and the advantages and disadvantages of OSS compared to closed source software. This allows us to understand how we can achieve and improve the quality assurance and quality control of OSSD.

3.1 Structure of Developers in OSS

There are four groups of OSS developers:

1. The core developers are a small group of expert developers who are responsible for the main core functionality of the system by writing high-quality code; managing and controlling the system; and making the plans, goals, and roadmap for the product.
2. The contributing developers are a bigger group of developers who directly affect the software development. They have the ability to add new features (depending on code modularity) and to do some other tasks, like fixing bugs and peer-reviewing code 5 .
3. The bug reporters are responsible for testing the system. Some of this task can be done by the core developers and contributing developers, as well as by the users of the system. One of the main tasks for this group is to ensure that more people will test the system on many different platforms (if the system supports this).
4. The users utilize the system; some of them are the developers. Sustainable software development community groups can be described in a simple onion model, as shown in

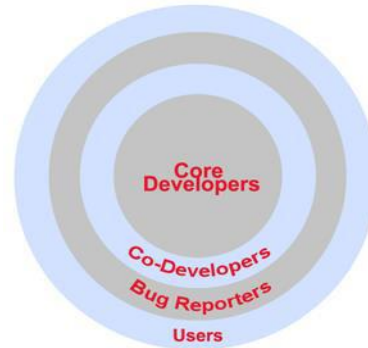

Figure 3.1: The model of software development community

4. OBJECTIVES

1. To identify relationship between various Software Metrics (like: LOC, CC and Maintainability Index) etc.
2. To find and compare the software metrics to search best Maintainability Index.

5. RESEARCH METHODOLOGY

The aim of the case study is to investigate the ability of Software metrics for analysis of complexity in open source software. Software metrics collected directly from source code (internal Metrics) are being used to measure the complexity of different open source software. Following Figure shows the model of our research.

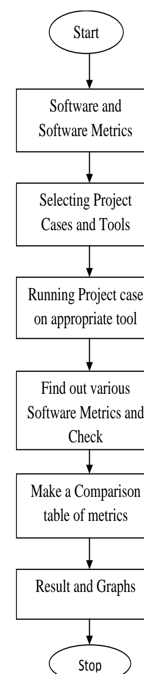

Figure5.1: Research Methodology

5.1 Analytical Framework

A method has been devised prior to conducting the case study. It is composed of:-

1. choosing the set of software to analyze as a means for collecting data
2. selection of suitable set of metrics suite
3. defining the level at which the metrics are applied and tool selection
4. performing analysis and interpretation of data with the use of statistical assessment

For conducting the experiment, the most popular open source software projects that are available in source forge have been selected. Most projects were downloaded from repositories such as Concurrent Version System (CVS) and Subversion (SVN). Additionally, all of the projects chosen are developed in Java since the software tool used to obtain the metric values provides support for the programming language. The majority of the open source projects available in the repositories are developed in Java. Furthermore, since some of the projects had different releases, stable and recent versions of the projects have been selected. The candidate projects were also the once which were available in CVS and SVN without read only access.

On the other hand, a particular set of design metrics suite have been selected since there are a variety of which to choose from that could be applicable for the experiment. The choice is performed with the study of articles of the most important metrics that are proven significant. The case study has used tools for obtaining metric values and for statistical assessments. The tools were chosen after comparing them with related software's. Automated software complexity analysis tools such as Code Analyzer, Analyst4j and Source Monitor were the candidate tools. The Analyst4J was selected as a favourite tool for this experiment since there were few drawbacks in the others. The analyses were performed at class level. After these considerations, the analysis and interpretation of the data was conducted with the use of statistical application software and automated extraction tool for metric values.

5.2 Analysis Plan

The analysis is performed in two steps. These are:-

Step 1: analyzing how the projects react to a particular metric.

Step 2: performing correlation tests and producing pairs that have high correlation.

The first step involves analysis of individual metric for the OSS. At this point, the results produced after the collection of data are put for discussion. The results are portrayed in the form visualization charts that would give an outlook how the projects reacted to the specified metric. In addition, since the metric values vary from one open source to another an interval of values has been used to know where the data lies in the graph. The frequency interval in connection with the number of classes in the projects will serve as a basis for plotting the graph. The interpretation of the data will then be followed.

In the second step, the correlation among the metrics of the open source projects will be used to identify highly correlated metrics. In statistics, correlation is the measure of relationship between different variables. The scales or the types of data used for measurement could be in the form of discrete or interval values. A coefficient of correlation value of 0indicates that there is no relationship between the variables (the metrics in this case), while a correlation coefficient of 1 signify strong relationship. This is particularly useful since identifying highly correlated metrics in the open source software's would enable to identify outliers or the classes which would be complex.

6. RESULTS

In this section different version of an open source software has been taken whose program code is written in different languages. Evaluation of these codes is done using metric Analyst 4J. It is a tool used in eclipse. It consists of large number of source code metrics which are used to compare the results obtained from both the codes. The results achieved are summarized under the various tables as shown later. This chapter presents the analysis of data. The distribution of the data across the selected projects will be shown in the first section then a discussion of the chosen metrics will follow. After that, the discussion of the correlation tests proceeds.

6.1 SweetHome3D

Sweet Home 3D is a free interior design application that helps you draw the plan of your house, arrange furniture on it and visit the results in 3D. Sweet Home 3D is a free interior design application that helps you place your furniture on a house 2D plan, with a 3D preview. This program is aimed at people who want to design their interior quickly, whether they are moving or they just want to redesign their existing home [8].

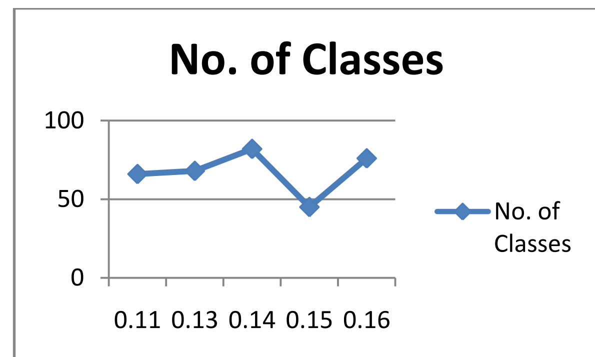

Fig6.1: No. of Classes of SweetHome3D

As Shows in the above graph, the variations of the number of classes have been analyzed. At the initial stages of the product, the No. of Classes rises from version 0.11 to version 0.14. After then it decreases in version 0.15 and increase No. of Classes in version 0.16. The maximum Classes is shown in version 0.14.

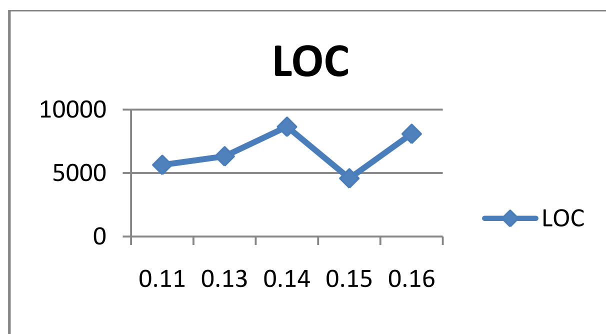

Fig6.2: Line of Code of SweetHome3D

The trend in LOC shows a consistent increase from version 0.11 to version 0.14. However, a major decrease in LOC observed in version 0.15 and slight increases in version 0.16.The study of first three versions shows that there has been an increase of LOC in these versions which is very significant. Some functionality seems to be added towards the end by adding more LOC. Therefore, it can reasonably be concluded that there has been significant additions in the product.

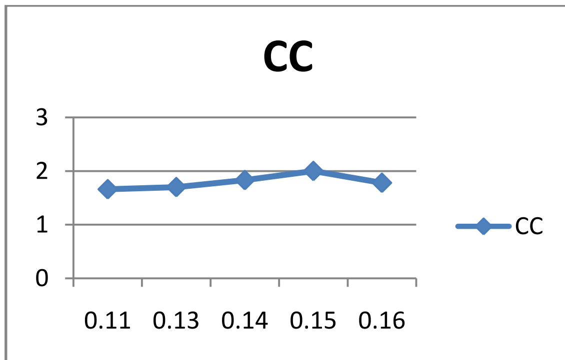

Fig6.3: Cyclomatic Complexity of SweetHome3D

At the initial stages of the product, the Cyclomatic Complexity rises from version 0.11 to version 0.15. After then it slightly decreases in version 0.16. The maximum complexity is shown in version 0.15.

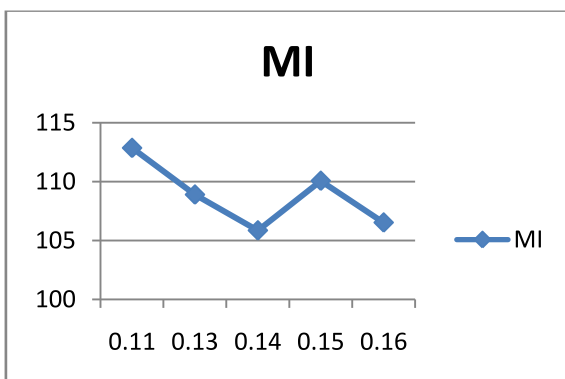

Fig6.4: Maintainability Index of SweetHome3D

Software is considered maintainable if its maintainability index is in a higher range. The graph shows the variations of MI. In version 0.11 MI is at level 112.86.Then it slightly decreases in version 0.11 to version 0.14. After that value of MI gets increases in version 0.15. Again it decreases in version 0.16. Maximum value of MI shown in version 0.11 which shows that version 0.11 is more maintainable as compared to other versions.

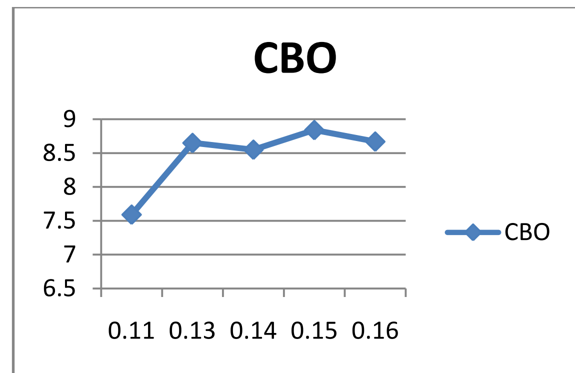

Fig6.5: Coupling between Objects of SweetHome3D

As shown in above graph, CBO is at level 7.59 in version 0.11. After that a huge increases observed in version 0.13. Then it slightly decreases in version 0.14. Then again increases in version 0.15. After that slightly decreases in version 0.16.Maximum CBO in version 0.15. Higher CBO indicates classes that may be difficult to understand and more difficult to maintain.

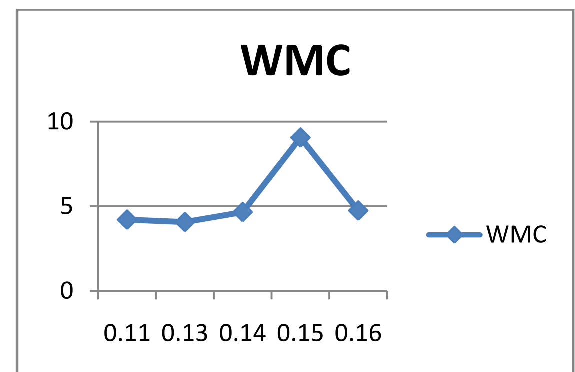

Fig6.6: Weighted Method per Class of SweetHome3D

The above graph of WMC reveals that, all the classes have a WMC less than 10. WMC is at level 4.21 in version 0.11. Then slightly decreases in version 0.13. Again increases from version 0.14 to version 0.15. After that huge decreases in version 0.16. Maximum WMC is in version 0.15. Higher WMC indicates classes that may be difficult to understand and more difficult to maintain.

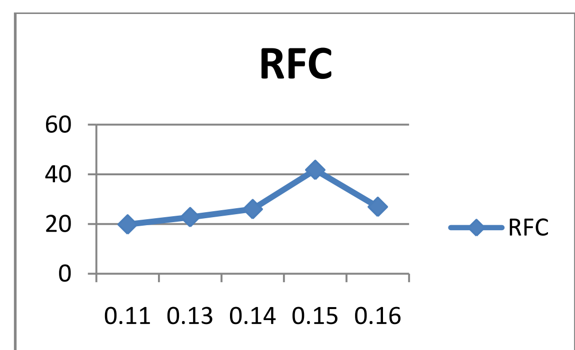

Fig6.7: Response for a class of SweetHome3D

The trend in RFC shows a consistent increase from version 0.11 to version 0.14. However, a major increase in RFC observed in version 0.15 and slight decreases in version 0.16.The study of first four versions shows that there has been an increase of RFC in these versions which is very significant.

Some functionality seems to be added towards the end by adding more classes. Therefore, it can reasonably be concluded that there has been significant additions in the product.

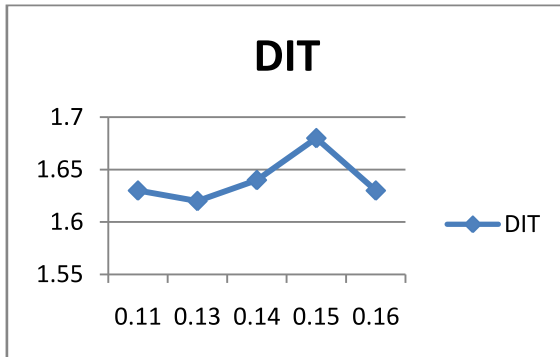

Fig6.8: Depth of inheritance tree of SweetHome3D

As shown in above graph, DIT is at level 1.63 in version 0.11. After that a slightly decreases observed in version 0.13. Then it slightly increases from version 0.14 to version 0.15. Then again decreases in version 0.16. Maximum DIT in version 0.15. Higher DIT indicates greater design complexity.

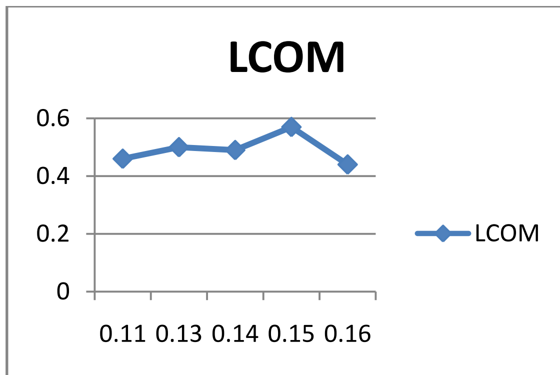

Fig6.9: Lack of Cohesion of SweetHome3D

As the above graph shows, that LCOM is at level 0.46. However, LCOM increases from version 0.11 to version 0.13. Then slightly decreases in version 0.14. Again increases in version 0.15 and decreases in version 0.16. Maximum LCOM in version 0.15. As, high cohesion indicates good class subdivision.

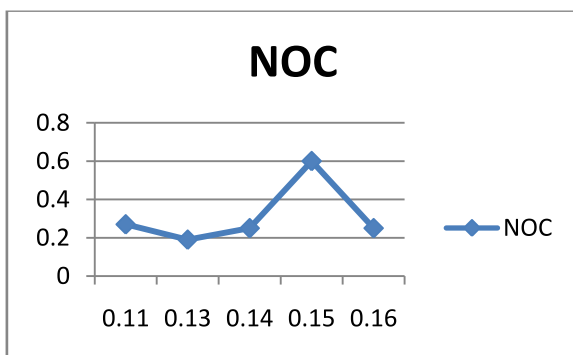

Fig6.10: Number of Children of SweetHome3D

In the above graph, NOC is at level o.27. However, slightly decreases from version 0.11 to version 0.13.Then increases in version 0.14 to version 0.15. Again decreases in version 0.16. Minimum NOC in version 0.13. Higher NOC indicates improper abstraction of the parent and misuse of sub classing..

The above table shows, five versions of SweetHome3D and their relationship between metrics. The result shows that the increases in No. of Classes, Lines of Code, Cyclomatic Complexity, Response of a Class, Lack of Cohesion, Coupling Between Object, Depth of Inheritance will decreases Maintainability Index, Weighted Method per Class and Number of Children from version 0.11 to version 0.13. Then increases in No. of Classes, LOC, CC, WMC, RFC, and NOC will decreases in LCOM, CBO, DIT and MI from version 0.13 to version 0.14. However, decreases in No. of Classes, LOC, DIT, and NOC will increases in CC, WMC, RFC, LCOM, CBO and MI from version 0.14 to version 0.15. Again increases in Classes, LOC, DIT, NOC will decreases in CC, WMC, RFC, LCOM, CBO and MI from version 0.15 to version 0.16.Therefore, by looking at the trends change in No. of Classes, LOC, CC, RFC, LCOM and CBO, it is concluded that as the Classes increases, the LOC also increases, CC increases, RFC also increases and CBO increases, LCOM also increases. Classes, RFC, LCOM in this case are in direct relationship with LOC, CC and CBO.

6.2 FindBugs

FindBugs is an Open Source Software created by Bill Pugh and David Haveever, which looks for bugs in java code. It was static analysis to identify hundreds of different potential types of errors in java programs. FingBugs operates on java byte code rather than source code.The change made from one version to another are as follows:

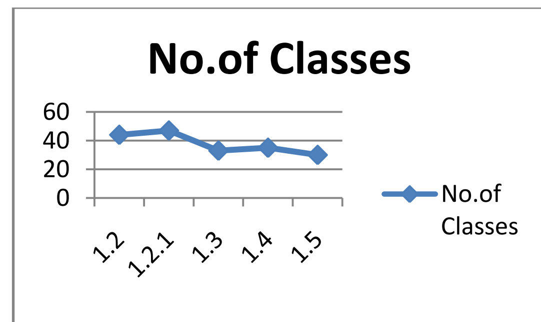

Fig6.11: No.of Classes of FindBugs

As Shows in the above graph, the variations of the number of classes have been analyzed. At the initial stages of the product, the No. of Classes rises from version 1.2 to version 1.2.1. After then it decreases in version 1.3 and increase No. of Classes in version 1.4. Then increase in version 1.5. The maximum Classes are shown in version 1.2.1.

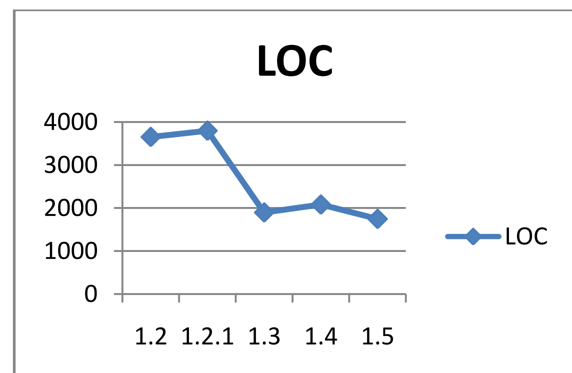

Fig6.12: Lines of Code of FindBugs

The trend in LOC shows a consistent increase from version 1.2 to version 1.2.1. However, a major decrease in LOC observed in version 1.3 and slight increases in version 1.4. Again decreases on version 1.5. The study of first five versions shows that there has been an increase in LOC, also increases in Classes. Some functionality seems to be added towards the end by adding more LOC. Therefore, it can reasonably be concluded that there has been significant additions in the product. Minimum LOC is shown in version 1.5.

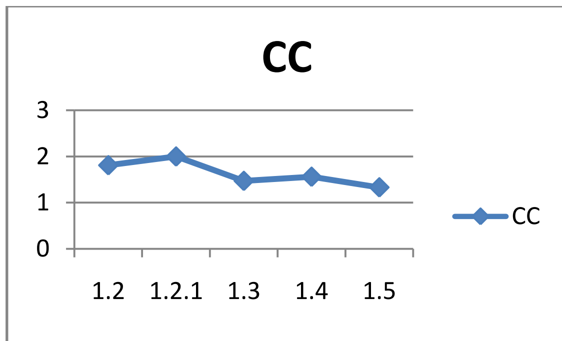

Fig6.13: Cyclomatic Complexity of FindBugs

At the initial stages of the product, the Cyclomatic Complexity rises from version 1.2 to version 1.2.1. After then it slightly decreases in version 1.3. Then increases in version 1.4 and decreases in version 1.5. The minimum complexity is shown in version 1.5.

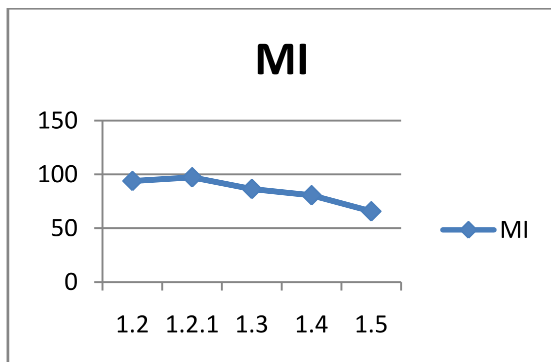

Fig6.14: Maintability Index of FindBugs

Software is considered maintainable if its maintainability index is in a higher range. The graph shows the variations of MI. In version 1.2 MI is at level 93.9. Then it decreases from version 1.2.1 to version 1.5.. Maximum value of MI shown in version 1.2 which shows that version 1.2 is more maintainable as compared to other versions.

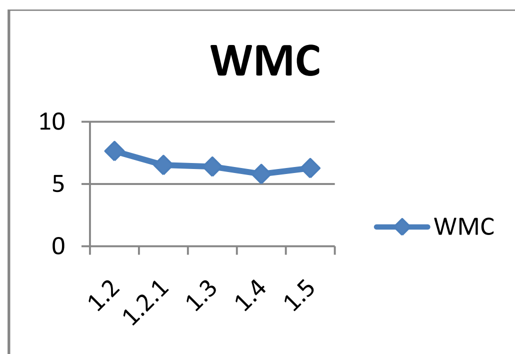

Fig6.15: Weighted Method per class of FindBugs

The above graph of WMC reveals that, all the classes have a WMC less than 10. WMC is at level 7.65 in version 1.2. Then decreases from version 1.2.1 to version 1.4. Again increases in version 1.5. Higher WMC indicates classes that may bedifficult to understand and more difficult to maintain. Minimum WMC is shown in version 1.4.

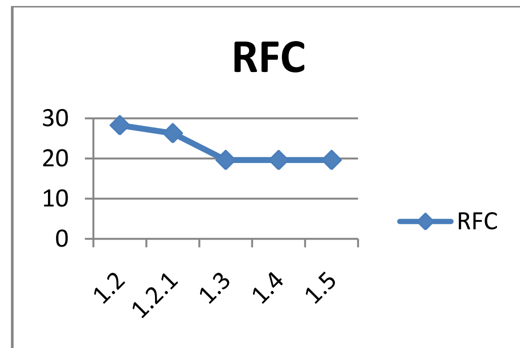

Fig6.16: Response for a class of FindBugs

The above graph shows, RFC is at level 28.29. However, a major decreases in RFC observed in version 1.2.1 to version 1.3. The trend in RFC shows a consistent decrease from version 1.3 to version 1.5.Minimum RFC is shown in version 1.4.

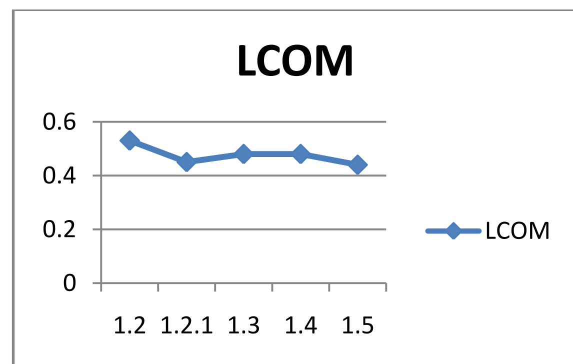

Fig6.17: Lack of Cohesion of FindBugs

As the above graph shows, that LCOM is at level 0.53 in version 1.2. However, LCOM decreases from version 1.2 to version 1.2.1. Then slightly increases in version 1.2.1 to version 1.4. Again decreases in version 1.5. Maximum LCOM in version 1.2. As, high cohesion indicates good class subdivision.

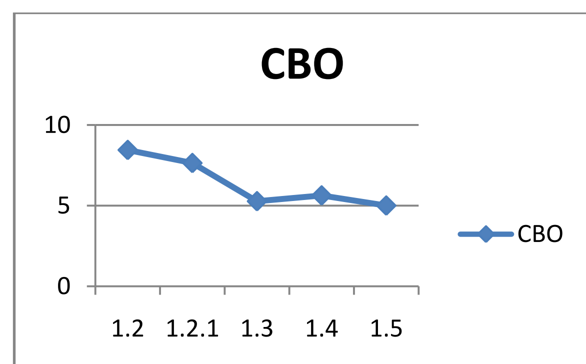

Fig6.18: Coupling between Object of FindBugs

As shown in above graph, CBO is at level 8.45 in version 1.2. After that a huge decreases observed in version 1.2 to version 1.3. Then it slightly increases in version 1.4. Then again decreases in version 1.5. Minimum CBO in version 1.5. Higher CBO indicates classes that may be difficult to understand and more difficult to maintain.

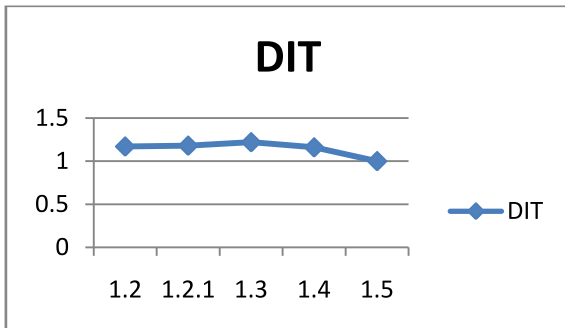

Fig6.19: Depth of Inheritance of FindBugz

As shown in above graph, DIT is at level 1.17 in version 1.2. After that a slightly increases observed from version 1.2.1 to version 1.3. Then it slightly decreases from version 1.4 to version 1.5. Minimum DIT in version 1.5. Higher DIT indicates greater design complexity.

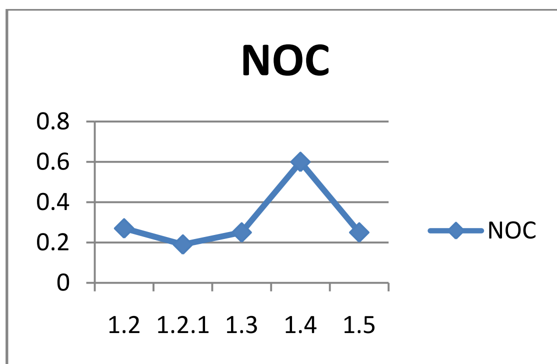

Fig6.20: Number of Children of FindBugs

In the above graph, NOC is at level 0.27. However, slightly decreases from version 1.2 to version 1.2.1.Then increases in version 1.3 to version 1.4. Again decreases in version 1.5. Minimum NOC in version 1.2.1. Higher NOC indicates improper abstraction of the parent and misuse of sub classing.

The above table shows, five versions of FindBugs and their relationship between metrics. The result shows that the increases in No. of Classes, Lines of Code, Cyclomatic Complexity, Depth of Inheritance, and MI will decrease Weighted Method per Class, RFC, LCOM, CBO and Number of Children from version 1.2 to version 1.2.1. Then decreases in No. of Classes, LOC, CC, WMC, RFC, CBO and MI will increases in LCOM, DIT and NOC from version 1.2.1 to version 1.3. However, increase in No. of Classes, LOC, CC, LCOM, CBO will decreases in WMC, RFC, DIT, MI and NOC from version 1.3 to version 1.4. Again decreases in Classes, LOC, CC, LCOM, CBO, DIT, MI will increases in WMC, RFC, and NOC from version 1.4 to version 1.5.Therefore, by looking at the trends change in No. of Classes, LOC, CC, RFC, WMC, it is concluded that as the Classes increases, the LOC and CC also increases, WMC increases, RFC also increases. Classes in this case are in direct relationship with LOC and CC, WMC also direct relationship with RFC.

6.3 JACOB

JACOB is a JAVA-COM Bridge that allows you to call COM Automation components from java. It uses JNI to make native calls to the COM libraries. JACOB runs on x86 and x64 environments supporting 32 bit and 64 bit JVMs.As of versions 1.8,the following things are true about JACOB: The project license changes from the LGPL to BSD, JACOB is now complied with java 1.4.2

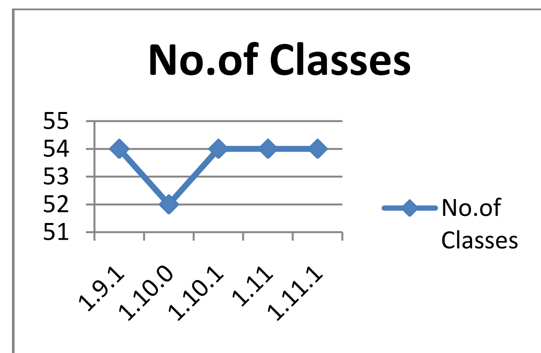

Fig6.21: No. of Classes of Jacob

As Shows in the above graph, the variations of the number of classes have been analyzed. At the initial stages of the product, the No. of Classes decrease from version 1.9.1 to version 1.10. After then it shows a consistent increase from version 1.10.1 to version 1.11.1.. The maximum Classes are shown in version 1.9.1, 1.10.1, 1.11 and 1.11.1.

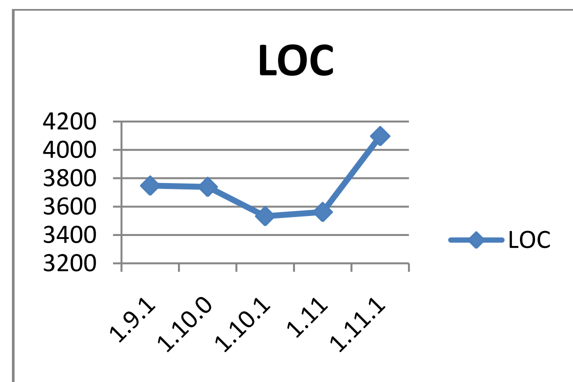

Fig6.22: Lines of Code of Jacob

The trend in LOC shows a consistent increase from version 1.9.1 to version 1.10. However, a major decrease in LOC observed in version 1.10.1 and slight increases in version 1.11 to version 1.11.1.Minimum LOC observed in version 1.10.1.Higher LOC increase the complexity of code.

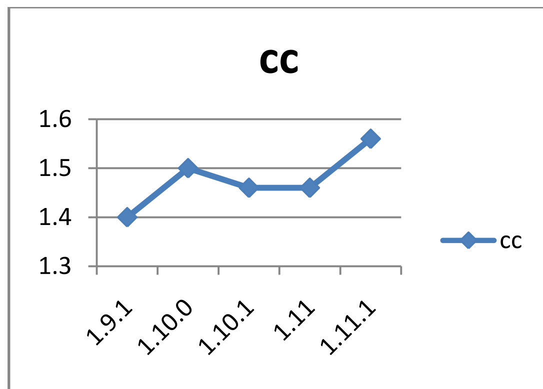

Fig6.23: Cyclomatic Complexity of Jacob

At the initial stages of the product, the Cyclomatic Complexity increases from version 1.9.1 to version 1.10. After then it slightly decreases in version 1.10.1 and consistent in version 1.11. Then again increases in version 1.11.1. The minimum complexity is shown in version 1.9.1.

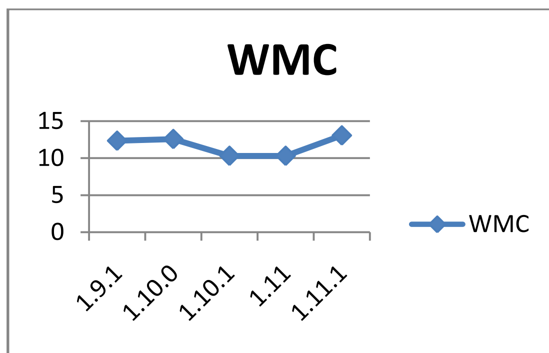

Fig6.24: Weighted Method per Class of Jacob

The above graph of WMC reveals that, all the classes have a WMC more than 10. WMC is at level 12.35 in version 1.9.1. Then increases from version 1.9.1 to version 1.10. Then decreases in version 1.10.1 and consistent in version 1.11. Again increases in version 1.11.1 Higher WMC indicates classes that may be difficult to understand and more difficult to maintain. Minimum WMC is shown in version 1.10.1 and 1.11.

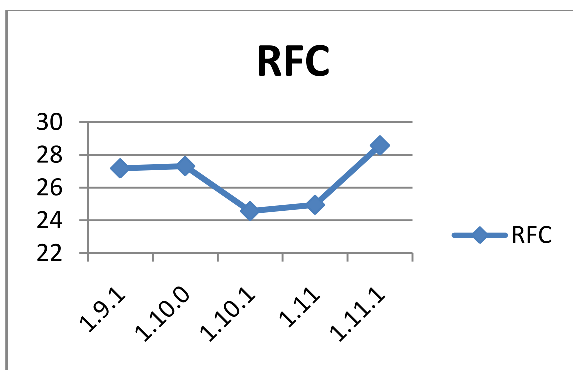

Fig6.25: Response for a Class of Jacob

The trend in RFC shows a consistent increase from version 1.9.1 to version 1.10. However, a major decrease in RFC observed in version 1.10.1 and slight increases in version 1.11 to version 1.11.1.Minimum RFC observed in version 1.10.1.

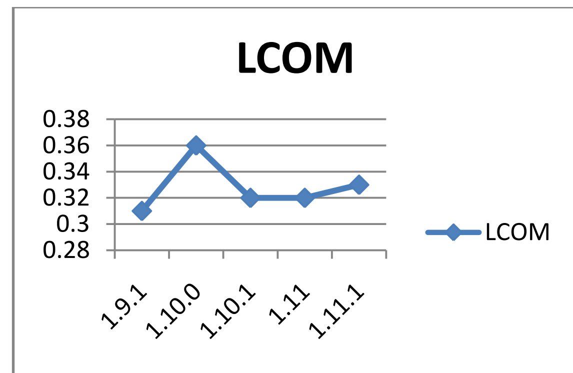

Fig6.26: Lack of Cohesion of Jacob

As the above graph shows, that LCOM is at level 0.31. However, LCOM increases from version 1.9.1to version 1.10. Then slightly decreases in version 1.10.1 and consistent in version 1.11. Again increases in version 1.11.1. Maximum LCOM in version 1.10. As, high cohesion indicates good class subdivision.

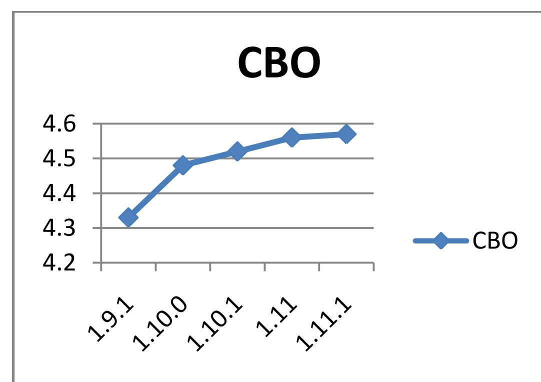

Fig6.27: Coupling between Object of Jacob

As shown in above graph, CBO is at level 4.33 in version 1.9.1. After that a huge increases observed in version 1.10 to version 1.11.1.Minimum CBO in version 1.9.1. Higher CBO indicates classes that may be difficult to understand and more difficult to maintain.

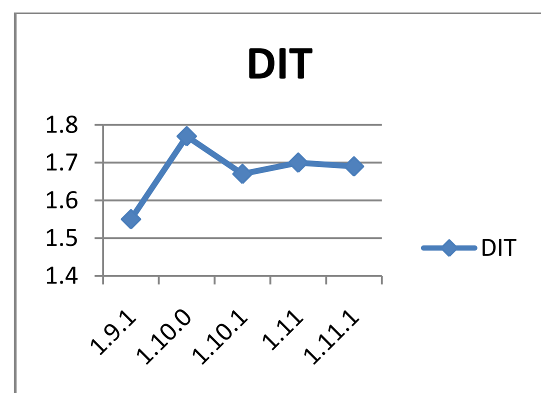

Fig6.28: Depth of Inheritance of Jacob

As shown in above graph, DIT is at level 1.55 in version 1.9.1. Then increases in version 1.10. After that a slightly decreases observed in version 1.10.1. Then it slightly increases from version 1.11. Then again decreases in version 1.11.1. Minimum DIT in version 1.9.1. Higher DIT indicates greater design complexity.

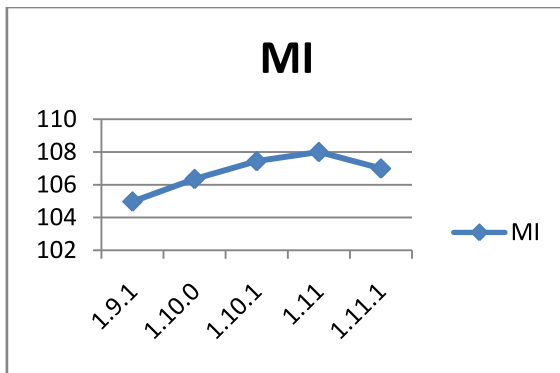

Fig6.29: Maintainability Index of Jacob

Software is considered maintainable if its maintainability index is in a higher range. The graph shows the variations of MI. In version 1.9.1 MI is at level 104.97.Then it increases from version 1.9.1 to version 1.11. After that value of MI gets decreases in version 1.11.1. Maximum value of MI shown in version 1.11 which shows that version 1.11 is more maintainable as compared to other versions.

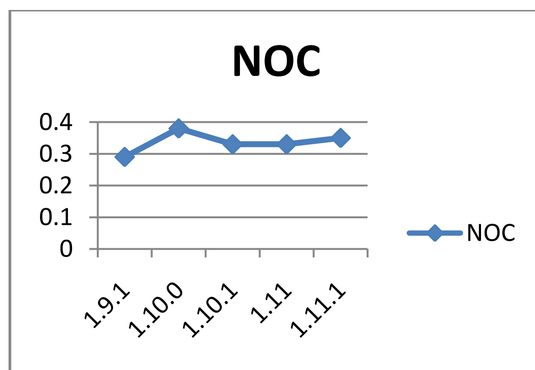

Fig6.30: Number of Children of Jacob

In the above graph, NOC is at level 0.29. However, slightly increases from version 1.9.1 to version 1.10.Then decreases in version 1.10.1 and remain consistent in version 1.11. Again increases in version 1.11.1. Minimum NOC in version 1.9.1. Higher NOC indicates improper abstraction of the parent and misuse of sub classing.

The above table shows, five versions of Jacob and their relationship between metrics. The result shows that the decreases in No. of Classes, Lines of Code will increases CC, Weighted Method per Class, RFC, LCOM, CBO , DIT, MI and Number of Children from version 1.9.1 to version 1.10. Then increases in No. of Classes, CBO and MI will decreases in LOC, CC,WMC, RFC, LCOM, DIT and NOC from version 1.10 to version 1.10.1. However, increase in No. of Classes, LOC, RFC, CBO and MI will decreases in CC,WMC, LCOM, DIT, MI and NOC from version 1.10.1to version 1.11. Again increases in Classes, LOC, CC, WMC, RFC, LCOM, CBO, DIT, and NOC will decreases MI from version 1.11to version 1.11.1.Therefore, by looking at the trends change in CC, WMC, LCOM, DIT, NOC. It is concluded that as the CC increases, the WMC, LCOM, DIT and NOC also increases. CC in this case is in direct relationship with WMC, LCOM, DIT and NOC.

6.4 JFree

JFreeChart is a free Java chart library that makes it easy for developers to display professional quality charts in their applications [7]. It supports bar charts, pie charts, line charts, time series charts, scatter plots, histograms, simple Gantt charts, Pareto charts, bubble plots, dials, thermometers and more. The JFreeChart project was founded thirteen years ago, in February 2000, by David Gilbert.

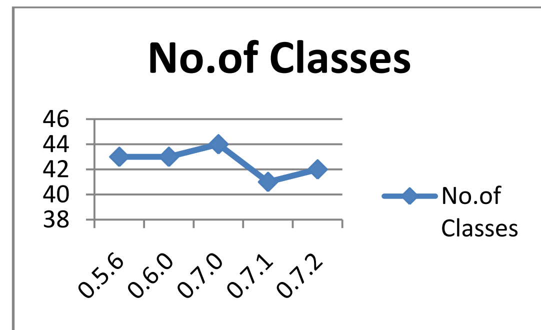

Fig6.31: No. of Classes of Jfree

As Shows in the above graph, the variations of the number of classes have been analyzed. At the initial stages of the product, the No. of Classes increase from version 0.5.6 to version 0.6.0. After then it shows decrease from version 0.7.0 to version 0.7.1 and increases in version 0.7.2. The maximum Classes are shown in version 0.7.0.

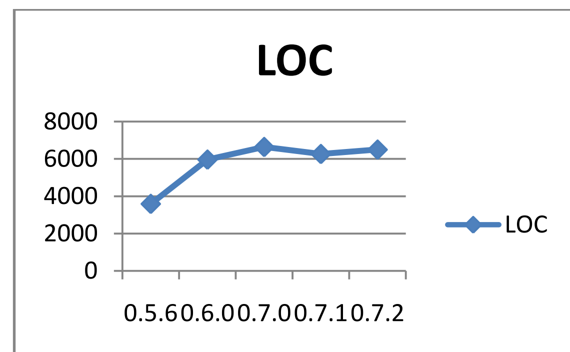

Fig6.32: lines of Code of jFree

The trend in LOC shows a consistent increase from version 0.5.6 to version 0.7.0. However, a major decrease in LOC observed in version 0.7.1 and slight increases in version 0.7.2.The study of first three versions shows that there has been an increase of LOC in these versions which is very significant. Some functionality seems to be added towards the end by adding more LOC. Therefore, it can reasonably be concluded that there has been significant additions in the product.

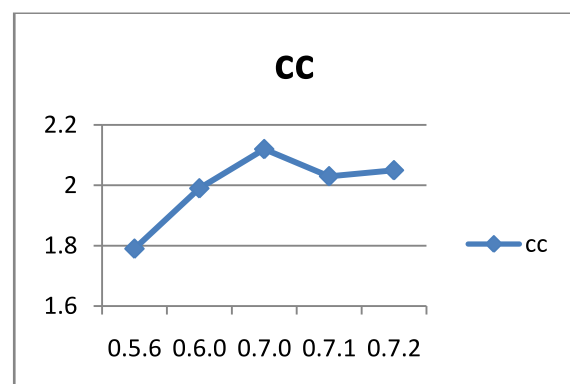

Fig6.33: Cyclomatic Complexity of jFree

At the initial stages of the product, the Cyclomatic Complexity rises from version 0.5.6 to version 0.7.0. After then it slightly decreases in version 0.7.1 and increases in version 0.7.2. The minimum complexity is shown in version 0.5.6.

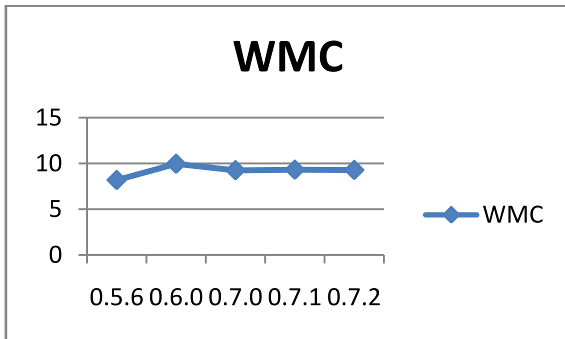

Fig6.34: Weighted Method per Class of jFree

The above graph of WMC reveals that, all the classes have a WMC less than 10. WMC is at level 8.18 in version 1.5.6. Then increases from version 0.5.6 to version 0.6.0. Then decreases in version 0.7.0 and consistent in version 0.7.1. Again decreases in version 0.7.2 Higher WMC indicates classes that may be difficult to understand and more difficult to maintain. Minimum WMC is shown in version 0.5.6.

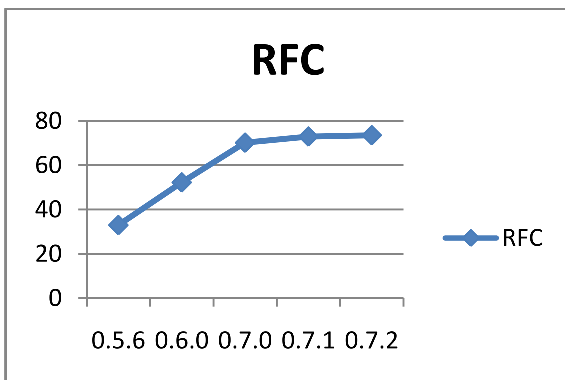

Fig6.35: Response for a Class of jFree

The trend in RFC shows a consistent increase from version 0.5.6 to version 0.7.2. The study of first five versions shows that there has been an increase of RFC in these versions which is very significant. Some functionality seems to be added towards the end by adding more classes. Therefore, it can reasonably be concluded that there has been significant additions in the product.

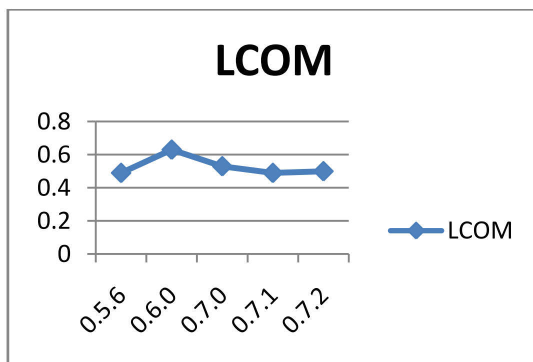

Fig6.36: Lack of Cohesion of jFree

As the above graph shows, that LCOM is at level 0.. However, LCOM increases from version 0.5.6 to version 0.6.0. Then slightly decreases in version 0.7.0 to version 0.7.1. Again increases in version 0.7.2.Maximum LCOM in version 0.6.0. As, high cohesion indicates good class subdivision.

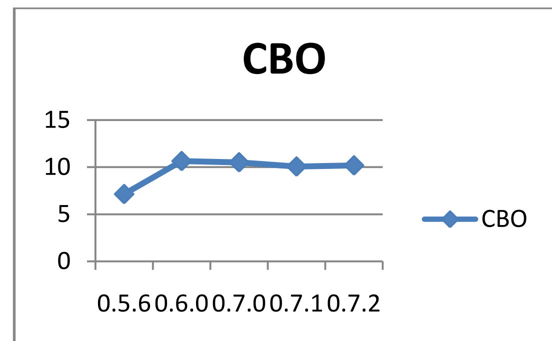

Fig6.37: Coupling between Object of jFree

As shown in above graph, CBO is at level 7.14 in version 0.5.6. After that a huge increases observed in version 0.5.6 to version 0.6.0. Then it slightly decreases in version 0.7.0 to version 0.7.1. Then again increases in version 0.7.2.Minimum CBO in version 0.5.6. Higher CBO indicates classes that may be difficult to understand and more difficult to maintain.

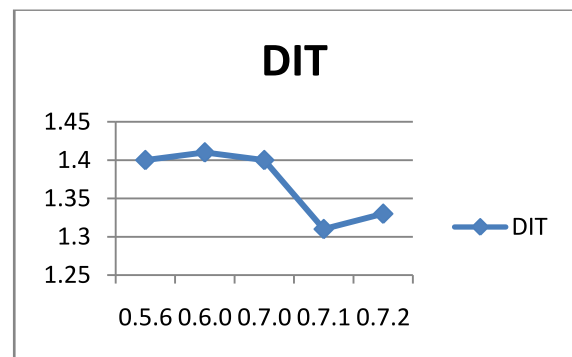

Fig6.38: Depth of Inheritance of jFree

As shown in above graph, DIT is at level 1.4 in version 0.5.6. After that a slightly increases observed in version 0.6.0. Then it slightly decreases from version 0.7.0 to version 0.7.2. Minimum DIT in version 0.7.1. Higher DIT indicates greater design complexity.

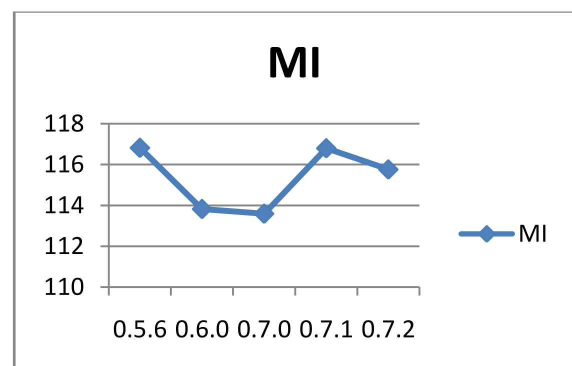

Fig6.39: Maintainability Index of jFree

Software is considered maintainable if its maintainability index is in a higher range. The graph shows the variations of MI. In version 0.5.6 MI is at level 116.82.Then it slightly

decreases in version 0.5.6 to version 0.6.0. After that value of MI gets increases in version 0.7.0 to version 0.7.1. Again it decreases in version 0.7.2. Maximum value of MI shown in version 0.5.6 which shows that version 0.5.6 is more maintainable as compared to other versions.

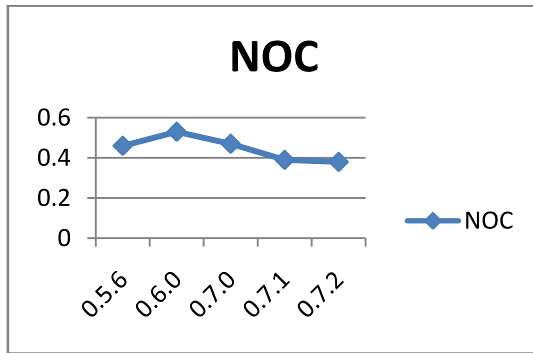

Fig6.40: Number of Children of jFree

In the above graph, NOC is at level o.46. However, slightly increases from version 0.5.6 to version 0.6.0.Then decreases in version 0.7.0 to version 0.7.2. Minimum NOC in version 0.7.2. Higher NOC indicates improper abstraction of the parent and misuse of sub classing.

The above table shows, five versions of jFree and their relationship between metrics. The result shows that the increases in No. of Classes, Lines of Code, WMC, RFC, LCOM, CBO, DIT and NOC will decreases MI, from version 0.5.6 to version 0.6.0. Then increases in No. of Classes, LOC, CC, RFC will decreases in WMC, LCOM, CBO, DIT , MI and NOC from version 0.6.0 to version 0.7.0. However, decrease in No. of Classes, LOC, CC, LCOM, CBO, DIT and NOC will increases in WMC, RFC and MI from version 0.7.0 to version 0.7.1. Again increases in Classes, LOC, CC, RFC, LCOM, CBO, DIT will decreases MI, WMC, AND NOC from version 0.7.1 to version 0.7.2.Therefore, by looking at the trends change in LOC, CC, LCOM, CBO, DIT. It is concluded that as the CC increases, the LOC and CC also increases. CC in this case is in direct relationship with LOC and Classes. As increases in LCOM, CBO and DIT also increases. LCOM is in direct relationship between CBO and DIT.

Table5.5: Comparison values of OSS using Analust4J

Project Name	Versions	LOC	CC	WMC	RFC	LCOM	CBO	DIT	MI	NOC
sweetHome3D	0.11	5646	1.66	4.21	19.82	0.46	7.59	1.66	112.86	0.27
	0.13	6323	1.7	4.07	22.75	0.5	8.65	1.68	108.9	0.19
	0.14	8654	1.83	4.66	25.98	0.49	8.55	1.64	105.86	0.25
	0.15	4585	2.0	9.06	41.8	0.57	8.84	1.62	110.07	0.6
	0.16	8095	1.78	4.75	26.92	0.44	8.67	1.63	106.53	0.25
FindBugs	1.2	3654	1.81	7.65	28.29	0.53	8.45	1.17	93.9	0.21
	1.2.1	3799	2.0	6.53	26.31	0.45	7.64	1.18	97.31	0.23
	1.3	1899	1.47	6.4	19.63	0.48	5.27	1.22	86.46	0.2
	1.4	2082	1.56	5.81	19.62	0.48	5.63	1.16	80.58	0.19
	1.5	1749	1.33	6.28	19.63	0.44	5.0	1.0	65.63	0.11
Jacob	1.9.1	3748	1.4	12.35	27.17	0.37	4.33	1.55	104.97	0.29
	1.10	3739	1.5	12.58	27.31	0.36	4.48	1.77	106.35	0.38
	1.10.1	3533	1.46	10.31	24.56	0.32	4.52	1.67	107.44	0.33
	1.11	3562	1.46	10.31	24.94	0.32	4.56	1.67	108.0	0.33
	1.11.1	4097	1.56	13.07	28.57	0.33	4.57	1.69	106.99	0.35
jFree	0.5.6	3585	1.79	8.18	32.93	0.49	7.14	1.4	116.82	0.46
	0.6.0	5962	1.99	9.96	52.2	0.63	10.65	1.41	113.82	0.53

	0.7.0	6639	2.12	9.25	70.13	0.53	10.52	1.4	113.59	0.47
	0.7.1	6262	2.03	9.31	72.9	0.49	10.07	1.31	116.8	0.39
	0.7.2	6498	2.05	9.28	73.49	0.5	10.19	1.33	115.76	0.38

The result shows that the increases in Classes will increases Line of Code and Coupling Between Objects increases Lack of Cohesion and Depth in Inheritance also increases. But when Classes decreases then MI increases. there is a sudden rise in the value of LOC, then it means that there have been some significant additions to the product. So it can be concluded that LOC is in direct relationship with No. of Classes. Similarly CBO also varies as LCOM and DIT varies Using these metrics together, one can easily predict that how maintainable a system is.

Table5.6: MI Values of Software

Software	Versions	MI
SweetHome3D	0.15	110.07
FindBugs	1.2	93.9
jFree	0.7.1	116.8
Jacob	1.11	108.0

According the above table, SweetHome3D version 0.15 have been taken as highly MI of SweetHome3D, as No.of Classes decreases in version 0.15 , will increases MI value that means version 0.15 is highly maintainable as compared o other versions of SweetHome3D. FindBugs version 1.2 have been taken as highly MI of FindBugs, as MI value of version 1.2 increases and Software highly maintainable jFree version 0.7.1 have been taken as highly maintainable version. Then Jacob version 1.11 have been taken as highly maintainable as compared to previous versions. Result concluded that best MI have been taken from jFree version 0.7.1 with 116.8 Metric value.

7. CONCLUSION/FUTURE SCOPE

This paper, based on a data set of 5 versions of java open source software namely SweetHome3D, FindBugs, Jfree and Jacob, the relationship between different metrics and maintainability of open source software have been investigated. The work not only analyzed the influence of individual metrics, but also reported their ability to predict how maintainable a system is, when these metrics are used together. Results show that these metrics are strongly related to maintainability of open source software.

The result shows that the increase in No. of Classes will increase Lines of Code and Coupling Between Objects increases Lack of Cohesion and Depth in Inheritance also increases. But when Classes decreases then MI increases. There is a sudden rise in the value of LOC, then it means that there have been some significant additions to the product. So it can be concluded that LOC is in direct relationship with No. of Classes. Similarly CBO also varies as LCOM and DIT varies Using these metrics together, one can easily predict that how maintainable a system is. . Result concluded that jFree version 0.7.1 with 116.8 MI value is best among all the software versions.

As different versions of open source software such as, SweetHome3D, FindBugs, Jfree and Jacob JFreeChart have been analyzed. Five versions of these software's have been taken and various metrics have been calculated. But if better results are required and if results are required on a broad basis, then more software versions should be taken. The bigger the number of versions is, better will be the results. Moreover, the value of different metrics can be calculated based on each class in corresponding package. Also there are no specific ranges defined for the metrics.Future work will be to define the acceptable ranges for all the metrics so as to maintain the quality of the software over its lifecycle.

8. REFERENCES

[1] "An Overview of Object-Oriented Design Metrics" Daniel RodriguezRachel HarrisonRUCS/2001/TR/A March2001.

[2] "Applying and Interpreting Object Oriented Metrics "Presenter: Dr. Linda H. RosenbergTrack: TrackMeasures/Metrics.

[3] CHIDAMBER-KEMERER (CK) AND LORENZE-KIDD (LK) METRICS TO ASSESS JAVA PROGRAMS Jubair J. Al-Ja'afer and Khair Eddin M. Sabri King Abdullah II School for Information Technology, University of Jordan, Jordan.

[4] International Journal of Engineering Research & Management Technology March 2014 Volume-1, Issue-2"Software Quality Metrics: Concept and Significance".

[5] International Journal of Advanced Computer Science andApplications, Vol. 3, No. 1, 2012 "Survey on Impact of Software Metrics on Software Quality" Mrinal SinghRawat1, Arpita Mittal2 Sanjay Kumar Dubey3.

[6] International Journal of Computer Applications (0975 8887) Volume 63– No.3, February2013 "Metrics inEvaluating Software Defects "Chen-Huei Chou School of Business College of Charleston Charleston, SC, USA

[7] https://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/

[8] https://sourceforge.net/projects/sweethome3d/files/SweetHome3D/

[9] McCabe Software. 2012. Metrics & Thresholds in McCabe IQ. Available at: http://www.mccabe.com/pdf.

[10] O'Neill, D. 1996. National Software Quality Experiment Results 1992-1996. In Proceedings of the Eighth Annual Software Technology Conference. pp. 21-26.

[11] Rosenberg, L. 1997. Metrics for Object-Oriented Environment, In Proceedings of EFAITP/AIE Third Annual Software Metrics Conference.

[12] Watson, A. H., McCabe, T. J., and Wallace, D. R. 1996. Structured testing: A testing methodology using The cyclomatic complexity metric. National Institute of Standards and Technology Special Publication 500-235.