Measuring Maintainability of Open Source Software Using Metrics

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science and Applications

Submitted By
Sheenam Batra
(Roll No. 601003026)

Under the supervision of:
Ms.Vineeta Bassi
Assistant Professor

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS
THAPAR UNIVERSITY
PATIALA – 147004

July 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, *"Measuring Maintainability of Open Source Software Using Metrics"*, in partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Applications submitted in School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Vineeta Bassi and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Sheenam.
(Sheenam Batra)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Vineeta Bassi 03/07/2012
(Vineeta Bassi)
Assistant Professor
SMCA

Countersigned by

(Dr. S.S.Bhatia)
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGMENT

First of all I am thankful to God for blessings and showing me the right decision. With his mercy, it has been possible for me to reach so far.

I would like to express sincerest thanks to my thesis supervisor Ms. Vineeta Bassi, for her inspiration, guidance, stimulating suggestions, immense help and support throughout the period of this research work. She has provided me with all the necessary resources including motivation and research environment without which it would not have been possible to complete this work. It was a great opportunity for me to do this work under her supervision.

I am thankful to the authors whose work I have consulted and quoted in this work.

I lack words to express my cordial thanks to all my friends for their useful comments and constructive suggestions during all the phases of my life.

Finally, I convey deep sense of gratitude towards my family members for their moral support and encouragement without which it would not have been possible to bring out this thesis.

Sheenam

Sheenam Batra

(601003026)

ABSTRACT

The aim of this thesis is to study the relationship between maintainability and metrics like lines of code, cyclomatic complexity and halstead volume of fifteen different versions of Java open source software. The result illustrate that these metrics are strongly related to the maintainability of open source software.

This report summaries the theory about maintainability of open source software and the impact of these metrics on its maintainability.

Open Source Software used for study in this thesis is JFreeChart. It is an open source library available for Java that allows users to easily generate graphs and charts. It is efficient software when a user wants to regenerate graphs that change on a regular basis.

JHawk tool is used for calculating values of metrics used for studying the maintainability of JFreeChart, open source software. It is a Java based open source framework which can be included in any java application.

The main objective of this thesis is to calculate different metrics like Lines of Code, Cyclomatic Complexity, Halstead Volume and Maintainability Index. The study also includes the comparison of these metrics plotted over various versions of open source software JFreeChart.

Table of Contents

Certificate i

Acknowledgement ii

Abstract iii

Table of Contents iv

List of Figures vi

List of Tables viii

1 Introduction 1

1.1 Open Source Software 1

1.2 Maintenance of Open Source Software 2

1.3 Success Stories of Open Source Software 3

1.4 Open Source Software Licenses 4

1.5 Advantages of Open Source Software 5

1.6 Disadvantages of Open Source Software 5

1.7 Software Maintenance 6

1.7.1 Nature of Maintenance 7

1.7.2 Need of Maintenance 7

1.8 Maintainability Index 8

1.9 JFreeChart 9

1.10 Various Versions of JFreeChart 10

2 Literature Survey 11

2.1 OSS Development 11

2.2 Open Source Software Development Community Model 12

2.3 International Standard Organization Software Maintenance Process 13

2.4 Open Source Software Maintenance Process 14

2.4.1 Defect Management System 16

2.4.2 Version Management System 16

2.5 Comparison between ISO and OSS Maintenance Process 16

2.6 JFreeChart 17

2.7 Packages in Each Version 19

3 Problem Statement 25

3.1 Problem Definition 25

3.2 Problem Analysis 25

4 Experimenting View of JHawk Tool 29

4.1 Introduction of JHawk Tool 29

4.2 JHawk Licensing Options 29

4.3 Snapshots of JHawk Tool 29

5 Results and Discussion 33

6 Conclusion and Future Scope 54

References 55

List of Figures

Figure 2.1	OSS Development Community Model	12
Figure 2.2	ISO Software Maintenance Process	14
Figure 2.3	OSS Maintenance Process	15
Figure 3.1	Number of Classes in Annotations Package	26
Figure 3.2	Number of Classes in Axis Package	26
Figure 3.3	Number of Classes in Block Package	27
Figure 3.4	Number of Classes in Event Package	27
Figure 3.5	Number of Classes in Entity Package	28
Figure 4.1	View of JHawk Welcome Tab	30
Figure 4.2	View of JHawk Select File Tab	31
Figure 4.3	View of JHawk System Tab	32
Figure 5.1	LOC of Annotations Package	34
Figure 5.2	Cyclomatic Complexity of Annotations Package	35
Figure 5.3	Halstead Volume of Annotations Package	36
Figure 5.4	Maintainability Index of Annotations Package	37
Figure 5.5	LOC of Axis Package	38
Figure 5.6	Cyclomatic Complexity of Axis Package	39
Figure 5.7	Halstead Volume of Axis Package	40
Figure 5.8	Maintainability Index of Annotation Package	41
Figure 5.9	LOC of Block Package	42
Figure 5.10	Cyclomatic Complexity of Block Package	43

Figure 5.11 Halstead Volume of Block Package 44

Figure 5.12 Maintainability Index of Block Package 45

Figure 5.13 LOC of Event Package 46

Figure 5.14 Cyclomatic Complexity of Event Package 47

Figure 5.15 Halstead Volume of Event Package 48

Figure 5.16 Maintainability Index of Event Package 49

Figure 5.17 LOC of Entity Package 50

Figure 5.18 Cyclomatic Complexity of Entity Package 51

Figure 5.19 Halstead Volume of Entity Package 52

Figure 5.20 Maintainability Index of Entity Package 53

List of Tables

Table 2.1	Classes of Annotations Package	19
Table 2.2	Classes of Axis Package	20
Table 2.3	Classes of Block Package	22
Table 2.4	Classes of Event Package	23
Table 2.5	Classes of Entity Package	24

Chapter 1
INTRODUCTION

Open Source Software has been getting more interest in the last few years. Many large and small corporations have taken a great concern in this increasing software market that shows some significant differences with traditional software. Businesses as well as educational institutions can get advantage from Open Source software.

The main focus of this thesis is to study about maintenance of open source software. The purpose of this thesis is to calculate the different metrics on which maintainability depends and study the comparison between different versions that how maintainability differs from one version to another. Open source software used in this dissertation is JFreeChart. Tool used to calculate different metrics is JHawk.

This chapter will set the context for the remainder of the thesis. It will introduce the concept of open source software, Maintenance and software evolution. Furthermore, the specific objectives will be set forth and the structure will be outlined.

1.1 Open Source Software

Although Open Source software has existed since 1960's but in the last few years it has got more attention. In 1983 the Free Software Foundation was founded by Richard Stallman the term 'Open Source' was introduced in 1998 [1]. Since then more and more companies have taken an interest in Open Source software.

Open source software is usually developed by volunteers from all over the world working cooperatively.OSS is software whose source code is freely available for anyone to inspect and study. It not only provides the right to inspect and study the source code but also allow us to use it for any desired purpose without monetary or other restrictions. These other purposes include making as many copies as desired, installing on as many

computers as desired, modifying in any desired way, and redistributing in its original or modified form [2].

Open Source Software (OSS) is becoming more vital and extensive these days, so its maintenance has become important concern. Maintainability is one of the critical issues that is being faced by the software industry. As software system cannot sustain in open source community if the software is not maintainable.

The state of Kansas defines: "OSS is software for which the source code is freely and publicly available, though the specific licensing agreements vary as to what one is permitted to do with that code" [3].

The main features that characterize open source software are the freedom that users have to:

- Use the software as they desire in any technically appropriate situation.
- Use the software to fit to their needs. Of course, this includes improving it, fixing its bugs, augmenting its functionality, and studying its operation.
- Redistribute the software to other users, who could themselves use it according to their own needs. This redistribution can be done for free, or at a charge, not fixed beforehand [4].

1.2 Maintenance of Open Source Software

Unlike commercial closed source software systems, open source software projects do not have expert maintenance team. Even for open source software projects that are sponsored by non-profits organizations, such as the Linux, Eclipse and so on, the maintenance work is still carried out by the developer community members. Although differently structured, the maintenance processes of open source software and closed source software are similar. Initially, maintenance requests are submitted and recorded. After then maintenance tasks are assigned, and finally when tasks are finished, the finishing time is also recorded and the task is closed. The main difference between them is that there is

usually tight time-table and plan for closed source software projects maintenance progress whereas the maintenance progress of open source software projects largely depends on the contributions of individuals from the developer community. Thus, common policies of maintenance that are adopted in close source software cannot be directly applied to open source software projects. Another important issue in open source software maintenance is the fast releases of new versions. Open source software communities are often composed of interests driven developers who aim to test and utilize the latest technologies. Such interests have brought a fast speed of software upgrades, as compared to traditional commercial software. For example, there are more than 500 different released versions of Linux since its commencement in 1991 whereas there are less than 80 released versions of Microsoft Windows since 1985 when the first Windows was available in market. The large number of different versions of some open source software projects makes it tough to maintain them all. Since the newer versions usually include bugs fixes and enhancements of previous versions and since they are free, users often switch to the newer version and less attention is paid to the older versions. Speed of maintenance of older versions of open source software might be largely slowed down as compared to newer versions because of shift of users and developers interest. The differences can be attributed to the different behavior patterns of maintenance participants. In closed source software projects, software maintenance is strictly planned and executed and maintenance efforts can be altered upon request. However, in open source software projects maintenance efforts cannot be tightly controlled and they depend on the number of maintenance participants, which frequently changes all the time. A proper model of the maintenance efforts of open source software can help in understanding the unique properties of its maintenance process [5].

1.3 Examples of Open Source Software

1. OpenOffice.org – office suite
2. Mozilla and Firefox – web browsers
3. Tex/Latex – Typesetting and Document preparation Systems

4. MySql – Relational Database Systems

5. GNU/LINUX – Operating systems

6. Apache web server – server [6].

1.4 Open Source Software Licenses

Licenses usually specify that the owner of the program is the company which publishes it, which just sells limited rights to use it. Usually, the conditions specified in licenses of open source software are the result of negotiations between several goals which are in some sense contradictory. Among them, the following can be cited:

- Assure some basic freedoms (redistribution, amendment, use) to the users.
- Guarantee some conditions forced by the authors (citation of the author in derived works, for instance).
- Ensure that derived works are also open source software.

Some of the common Open Source Software Licenses are:

1. **BSD (Berkeley Software Distribution)**
   BSD licenses impose almost no conditions on what a user can do with the software, including charging clients with no compulsion to include source code. Redistributors can do almost anything with the software, including using it for proprietary products. The authors only want their work to be renowned [7].
2. **GPL (GNU General Public License)**
   It allows modification without limitations (if the derived work is also covered by GPL), and complete integration with other software is only possible if that other software is also covered by GPL [7].
3. **MPL (Mozilla Public License)**
   This is the license made by Netscape to give out the code of Mozilla, the new version of it network navigator. It is in many respects similar to the GPL, but perhaps more “enterprise oriented” [7].

1.5 Advantages of Open Source Software

1. **Reliability**

   This is a very common concern with proprietary software. For example a company uses a software product, and relies on developers for upgrades and continued development. If the developers decide to stop development of the product, no one has the right to take the program and continue development. Open source software effectively protects against this because if the group or company that originated the code decides to stop development, it is always possible to support another software group to continue the maintenance and improvement, without authorized or convenient restrictions [8].

2. **Cost**

   By using OSS one can minimize cost as its source code is freely available. Anyone can use the existing code as a base to start new projects. One can collect the information at a negligible cost [8].

3. **Freedom**

   It allows the unlimited alteration and enhancement of a software product. It also makes it possible to port the code to new hardware, to correct it to changing conditions, and to reach a thorough understanding of how the system works [8].

1.6 Disadvantages of Open Source Software

1. **No support exists**

   There is no qualified support available for OSS but there is lots of help available on the Internet and there are many self-motivated forums that can help you install and run open source software [9].

2. **Higher installation cost**

   Most of the open source applications are incompatible with the present day gadgets. Technical support in this case is costlier compared to commercial software, so it increases the installation cost [9].

3. **No guarantee of updates**

   As there is no need to pay for the open source software, nobody is bound to give regular updates. Anybody can get stuck with the same old version without ever getting an update [9].

1.7 Software Maintenance

In a changing environment, the software is also supposed to change. Software maintenance is one of the processes in the software life cycle. According to the Software Body of Knowledge (SWEBOK) [IEE04], software maintenance is defined in the IEEE Standard for Software Maintenance as follows:

"Software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment" [10].

The four main maintenance categories are:

1. **Corrective maintenance**

   Corrective software maintenance involves developing and deploying solutions to problems ("bugs") that arise during use of a software program.

2. **Perfective maintenance**

   Perfective software maintenance involves computer programmers working to improve the way a software program functions or how quickly it processes requests.

3. **Adaptive maintenance**

   The field of technology constantly changes through both hardware and

software developments. Adaptive software maintenance addresses these changes.

4. **Preventive maintenance**

   When computer programmers engage in preventative software maintenance they try to prevent problems with software programs before they occur. Computer programmers test software to make sure the software can handle high data loads and other stressful operations without problems.

1.7.1 Nature of Maintenance

Software maintenance sustains the software product right through its operational life cycle. Change requests are logged and tracked, the impact of projected changes determined, coded and other software artifacts are modified, testing is conducted, and a new version of the software product is released.

Maintainers can learn from the developer's awareness of the software and contact them initially to reduce the maintenance efforts. In some cases, the software engineer cannot be traced or has moved on to other tasks, which creates an additional challenge for the maintainers. Maintenance must take the products of the development, code, or documentation, and support them immediately to evolve/maintain them progressively over the software life cycle [10].

1.7.2 Need of Maintenance

Maintenance is needed to make sure that the software continues to gratify user requirements. Maintenance is relevant to software developed using any software life cycle. The system changes due to remedial and non-corrective software events. Maintenance must be performed in order to:

- Correct faults
- Perk up the plan

- Execute enhancements
- Integrate with other systems
- Adapt programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software
- Retire software [10].

1.8 Maintainability Index (MI)

The MI [11] 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

MI is very useful in improving software system's maintainability. It makes system more maintainable and cost effective during the software development life cycle. Also, it can be used to study and assess different systems by comparing their MI values. It gives an outstanding approach into the source code of a system for direct manual analysis to emphasize areas of the code which require human intervention.

There are various tools available to calculate Maintainability Index of software. Below are few listed tools [12]

- Understand 4 JAVA
- Analyst4J
- JHawk

JHawk tool has been used for calculating the Maintainability Index of Open source software JFreeChart.

1.9 JFreeChart

The open source software that has been used in this dissertation is JFreeChart.

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's wide feature set includes:

- A reliable and well-documented API, which supports a wide range of chart types.
- A flexible design that is easy to expand, and targets both server-side and client-side applications.
- Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG).

- JFreeChart is "open source" or, more specifically, **free software**. It is distributed under the terms of the **GNU Lesser General Public License** (LGPL), which permits use in proprietary applications [13].

1.10 Various Versions of JFreeChart

There are fifteen different versions [14] of JFreeChart have been taken in this dissertation.

1. JFreeChart 1.0.0
2. JFreeChart 1.0.1
3. JFreeChart 1.0.2
4. JFreeChart 1.0.3
5. JFreeChart 1.0.4
6. JFreeChart 1.0.5
7. JFreeChart 1.0.6
8. JFreeChart 1.0.7
9. JFreeChart 1.0.8
10. JFreeChart 1.0.9
11. JFreeChart 1.0.10
12. JFreeChart 1.0.11
13. JFreeChart 1.0.12
14. JFreeChart 1.0.13
15. JFreeChart 1.0.14

Chapter 2

LITERATURE SURVEY

Paul Karvaagh defines open source software as "Open source software is software that must be distributed with source code included or easily available such as by free download from the internet". The source code should be in the same form that a programmer would actually use to maintain it, not a generated or immediate code form. The license of this software will not restrict others from distributing the code or modifications and derived works under the same terms [15].

2.1 OSS Devolvement

Open source software development is the process by which open source software is developed. These softwares are accessible with its source code and are available to study and modify to improve its functionality under an open source license. In the past, the open source software development method has been very amorphous, because of non availability of development tools, phases, etc. However, in recent times there has been much better development, organization, and declaration within the open source community.

Open source software development is a diverse socio-technical process. In order to examine this, it is important to focus both the software and processes affecting it, as well as the panel working on the software: why are the developers working on it and how do they arrange themselves. Projects vary a lot from each other, and there are divergences even in the workings and organizational approach of a single project over time.

It's difficult to acquire an open source project following a more conventional software development method like the waterfall model, because these conventional methods doesn't allow user to go back to a previous phase. In OSS development process the

requirements are hardly ever gathered before the start of the project, instead they are based on early releases of the software product.

What we really need is to know and compute the mechanisms and motivating forces of OSS, so that we can:

- Use the same mechanisms to structure the money-making projects,
- Rapidly acquire or create the missing ingredients to speed up new OSS projects,
- Identifying what role we can take and how much dedication we will have to make in an OSS project,
- How fast we can get an initial working model from an OSS specifying a fixed budget [16].

2.2 Open Source Software Development (OSSD) Community Model

The OSSD model has the following features:

1. Development Community includes those developers which want to contribute their work in open source software development.

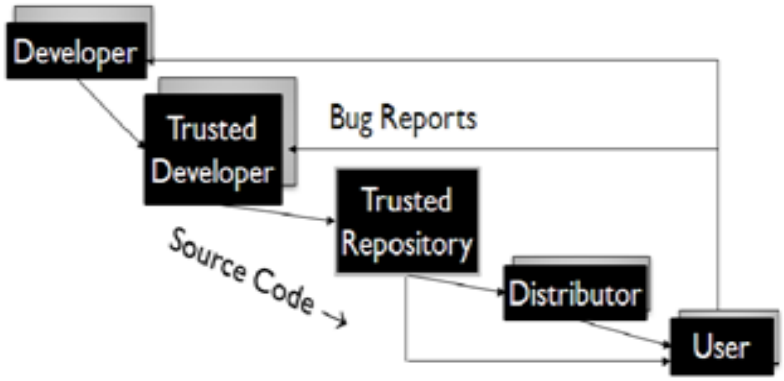

Figure 2.1 OSS Development Community Model [17]

2. Trusted Developers includes those developers which are expertise in a specific domain.

3. Trusted Repository is a repository where source code resides. Whenever trusted developer develop a new code or made some changes in code, they put the code in trusted repository.

4. Now user can directly access the source code from trusted repository as well as through distributors.

5. Distributors also can access the source code from trusted repository.

6. While using the software, if user will find any bug in the source code then user can report it to either developer or trusted developer.

2.3 International Standard Organization (ISO) Software Maintenance Process

An overview of the ISO maintenance process is presented in Figure 2.2

ISO primary software maintenance activities are broken down into following tasks:

1. Process Implementation

In process implementation, first of all maintenance plans and procedures are developed for modification re-quest and finally implement the configuration management process.

2. Problem and Modification

In problem and modification step, tasks are performed for initial analysis and verify problem. After verifying problem, options are developed for implementing the modification and results are documented. Finally obtain approval for modification option.

3. Modification Implementation

In this step, main task is to perform detailed analysis and then develop the code and test the modification.

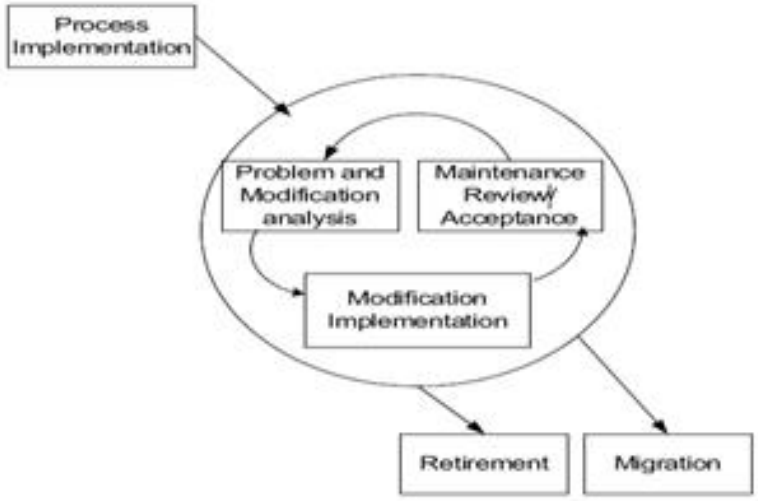

Figure 2.2 ISO Software Maintenance Process [18]

4. Maintenance Review/Acceptance

In this task, various reviews are conducted and then obtain approval for modification.

5. Migration

In migration task, we ensure that the migration should be in accordance with ISO. After then migration plan is developed. Users are notified about migration plans and conduct parallel operations. Then users are notified that migration has started and finally a post-operation review is conducted and ensure that old data is accessible.

6. Software Retirement

In software retirement task, retirement plan is developed. After developing the retirement plan users are notified about retirement plans and conduct parallel operations. Finally users are notified that retirement has started and ensure that old data is accessible.

2.4 Open Source Software Maintenance Process

It use two kinds of management systems

Defect Management System (DMS)

Version Management System (VMS)

Following Activities are involved in Open Source Software:

1. Whenever user finds any defect, he/she can report it to the DMS

2. Developers can retrieve the information reported by users and analyze it.

3. The developer can retrieve the source code from the VMS and modifies the source code to remove the defect.

4. The modification is reviewed before it is accepted and merged to the main version of the source code in the VMS.

5. The status of the defect is updated into the DMS.

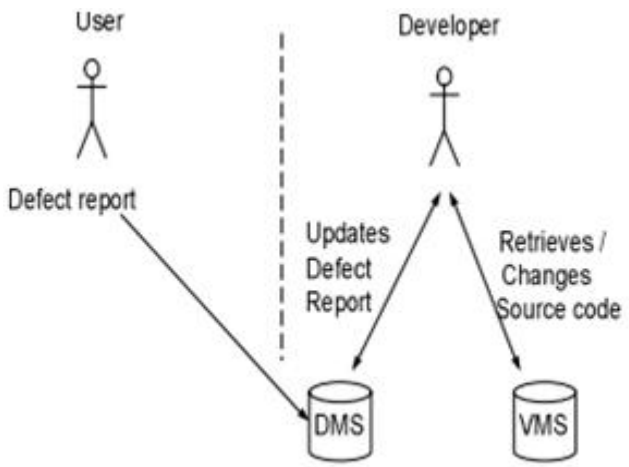

Figure 2.3 OSS Maintenance Process [19]

6. Then, a new version of the software is available from the VMS or from the project web site. Then the users can retrieve the modified version.

2.4.1 Defect Management System (DMS)

In software development, generally defect reports and improvement requests become difficult to manage. When the flow of defects, requested enhancements and support requests increases, it is difficult to manage all of them with email. Therefore, many companies and open source software projects (OSSPs) have built systems for managing these requests called Defect Management System. It store requests and reports in databases from where they can be retrieved, assigned, processed and closed [20].

2.4.2 Version Management System (VMS)

In the software development, the source code is customized daily but sometimes changes are not accurate, so developers have to restore older versions of the source code. Moreover, nowadays when many developers work concurrently, modifications have to be merged in a controllable way. A version management system (VMS) is used to fulfill these requirements. In some contexts, a VMS is also called a version control system or source code management system. The purpose of VMS is to controlling software configuration in the concurrent development, testing and maintenance processes. VMSs store each change, so every revision is retrievable [21].

2.5 Comparison between ISO and OSS Maintenance Processes

The major difference in ISO and OSS maintenance processes is of retirement activity and migration activity. In ISO maintenance process, Retirement and Migration plans are first developed and then notified to the users. Also, it conducts parallel operations to inform users that plans have been started. While in OSS maintenance process there are no such activities. Another difference between ISO and OSS maintenance process is in modification review and acceptance. Modifications are accepted before implementation in the ISO maintenance process but after implementation in the OSS maintenance process. In OSS maintenance process developers can retrieve any of the defects and start

to implement the required modifications. After implementations, the modification is reviewed and can still be discarded [22].

2.6 JFreeChart

It is a free Java chart library. JFreeChart supports pie charts (2D and 3D), bar charts (horizontal and vertical, regular and stacked), line charts, scatter plots, time series charts, Gantt charts, combined plots, thermometers, dials and more. JFreeChart can be used in applications, applets, servlets and JSP. This project is maintained by David Gilbert.

JFreeChart is a free chart library for the Java platform. It runs on the Java Platform (JDK 1.3 or later) and uses the Java 2D API for drawing. JFreeChart is licensed under the terms of the GNU Lesser General Public License (LGPL)

The changes made from one version to another are as follows

Version 1.0.0: This version was released on 2nd December, 2005.

Version 1.0.1: This version was released on 27th January, 2006. Primarily, a bug fix is released in this version. In addition, there are some API adjustments. It is the first stable release of the JFreeChart class library, all future releases in the 1.0.x series will aim to maintain backward compatibility with this release.

Version 1.0.2: This version was released on 25th August, 2006. This release contains both new features and bug fixes.

Version 1.0.3: This version was released on 17th November, 2006. This release contains a new DialPlot class and this release also contains many new features and bug fixes.

Version 1.0.4: This version was released on 9th February, 2007. This release contains number of additions to the API for usability, and many bug fixes.

Version 1.0.5: This version was released on 23rd March, 2007. This release contains a new DeviationRenderer class and there are also enhancements to a number of existing classes and numerous bug fixes.

Version 1.0.6: This version was released on 15th June, 2007. In this release, the VectorRenderer and associated dataset classes have been promoted to the standard API from the experimental source tree. There are also minor feature additions and bugs fixes in this release.

Version 1.0.7: This version was released on 14th November, 2007. There have not been any major changes in this release but numerous bug fixes are there.

Version 1.0.8: This version was released on 23rd November, 2007. Primarily, there are new bug fixes in this release.

Version 1.0.9: This version was released on 4th January, 2008. This release fixes a security advisory with respect to the HTML image maps generated by JFreeChart.

Version 1.0.10: This version was released on 9th June, 2008. It was just another maintenance release.

Version 1.0.11: This version was released on 19th September, 2008. This release includes a new chart theming mechanism and there are numerous other feature enhancements and bug fixes are also there.

Version 1.0.12: This version was launched on 31st December, 2008. This release has an added support for minor tick marks, mapping data sets to more than one axis and there are numerous bug fixes in this release.

Version 1.0.13: This was launched on 17th April, 2009. There are some significant new features in this release.

Version 1.0.14: This version is the most recent version of JFreeChart. This was released on 20th November, 2011. This release contains support for multiple and logarithmic axes. In addition, there are some bug fixes also [23].

2.7 Packages in Each Version

There are various no. of packages in each version. Five common packages Annotations, Axis, Block, Event, and Entity have been taken. There is different no. of classes in each package which are given in below tables [24].

Annotations:

Table 2.1 Classes of Annotations package

Version 1.0.0	**Version1.0.1**	**Version 1.0.2**	**Version 1.0.3**
1.AbstractXYAnnotation	Same no. of classes are used w.r.t the previous version.	Same no. of classes are used w.r.t the previous version.	1 class has been added w.r.t the previous versions.
2.CategoryAnnotation			
3.CategoryLineAnnotation			
4.CategoryTextAnnotation			
5.TextAnnotation			
6.XYAnnotation			
7.XYBoxAnnotation			
8.XYDrawableAnnotation			15. CategoryPointerAnnotation
9.XYImageannotation			
10.XYLineAnnotation			
11.XYPointerAnnotation			
12.XYPolygonAnnotation			
13.XYShapeAnnotation			
14.XYTextAnnotation			

<table>
<tr><th>Version 1.0.4 to 1.0.10</th><th>Version 1.0.11</th><th>Version 1.0.12</th><th>Version 1.0.13</th><th>Version 1.0.14</th></tr>
<tr><td rowspan="3">Same no. of classes has been used as compared to the previous versions</td><td>Two classes have been added.</td><td rowspan="3">Same no. of classes w.r.t. previous versions.</td><td>One class has been added.</td><td>Two classes have been added.</td></tr>
<tr><td>16. XYDataImageAnnotation</td><td rowspan="2">18. XYAnnotationBoundsInfo</td><td>19.AbstractAnnotation</td></tr>
<tr><td>17. XYTitleAnnotation</td><td>20. Annotation</td></tr>
</table>

Axis:

Table 2.2 Classes of Axis package

<table>
<tr><th>Version 1.0.0</th><th>Version 1.0.1 to 1.0.6</th><th>Version 1.0.7</th><th>Version 1.0.8 to 1.0.12</th></tr>
<tr><td>1.Axis</td><td rowspan="9">Same number of classes have been used as compared to the previous versions.</td><td>2 more classes have been added.</td><td rowspan="9">Same no of classes have been used as compared to the previous versions.</td></tr>
<tr><td>2.AxisCollection</td><td rowspan="8"></td></tr>
<tr><td>3.AxisLocation</td></tr>
<tr><td>4.AxisSpace</td></tr>
<tr><td>5.AxisState</td></tr>
<tr><td>6.CategoryAnchor</td></tr>
<tr><td>7.CategoryAxis3D</td></tr>
<tr><td>8.CategoryAxis</td></tr>
<tr><td>9.CategoryLabelPosition</td></tr>
</table>

10.CategoryLabelPositions			
11.CategoryLabelWidthtype			
12.CategoryTick			
13.ColorBar			
14.CompassFormat			
15.CyclicNumberAxis			
16.DateAxis			
17.DateTick			
18.DateTickMarkposition			
19.DateTickUnit			
20.ExtendedCategoryAxis			
21.LogarithmicAxis			
22.MarkerAxisBand			
23.ModuloAxis			
24.MonthDateFormat			
25.NumberAxis3D			
26.NumberAxis			
27.NumberTick			
28.NumberTickUnit			
29.PeriodAxis			
30.PeriodAxisLabelInfo			
31.QuarterDateFormat			
32.SegmentedTimeLine			
33.StandardTickUnitSource			
34.SubCategoryAxis			
35.SymbolAxis			
36.Tick			
37.TickUnit			
38.TickUnits			

39.TickUnitSource			
40.TimeLine			
41.ValueAxis		43. TickType	
42.ValueTick		44. LogAxis	

Version 1.0.13	**Version 1.0.14**
one more class has been added.	Same no of classes have been used as compared to the previous version.
45. DateTickUnitType	

Block:

Table 2.3 Classes of Block package

Version 1.0.0	**Version 1.0.1 to 1.0.4**	**Version 1.0.5**	**Version 1.0.6 to 1.0.14**
1.AbstractBlock	Same no of classes have been used as compared to the previous version.	Two Classes have been added	Same no of classes have been used as compared to the previous version.
2.Arrangement			
3.Block			
4.BlockBorder			
5.BlockContainer			
6.BlockParams			
7.BlockResult			
8.BorderArrangement			
9.CenterArrangement			
10.ColorBlock			
11.ColumnArrangement			
12.EmptyBlock			

13.EntityBlockParams			
14.EntityBlockResult			
15.FlowArrangement			
16.GridArrangement			
17.LabelBlock			
18.LengthConstraintType		20.BlockFrame	
19.RectangleConstraint		21.LineBorder	

Event:

Table 2.4 Classes of Event package

Version 1.0.0	**Version 1.0.1 to 1.0.2**	**Version 1.0.3**	**Version 1.0.4 to 1.0.12**
1.AxisChangeEvent	Same no. of classes w.r.t the previous version.	2 more classes have been added.	Same no. of classes w.r.t the previous versions.
2.AxisChangeListener			
3.ChartChangeEvent			
4.ChartChangeEventType			
5.ChartChangeListener			
6.ChartProgressEvent			
7.ChartProgressListener			
8.PlotChangeEvent			
9.PlotChangeListener			
10.RendererChangeEvent		14.MarkerChangeEvent	
11.RendererChangeListener			
12.TitleChangeEvent		15.MarkerChangeListener	
13.TitleChangeListener			

Version 1.0.13	Version 1.0.14
2 classes have been added	2 classes have been added
16.OverlayChangeEvent	18.AnnotationChangeEvent
17.OverlayChangeListener	19.AnnotationChangeListener

Entity:

Table 2.5 Classes of Entity package

Version 1.0.0	Version 1.0.1 to 1.0.2	Version 1.0.3	Version 1.0.4 to 1.0.12
1.CategoryItemEntity	Same no. of classes w.r.t the previous version	One class have been added	Same no. of classes w.r.t the previous version.
2.ChartEntity			
3.ContourEntity			
4.EntityCollection			
5.LegendItemEntity			
6.PieSectionEntity			
7.StandardEntityCollection			
8.TickLabelEntity			
9.XYAnnotationEntity			
10.XYItemEntity		11.CategoryLabelEntity	

Version 1.0.13	Version 1.0.14
Four classes have been added	Same no. of classes w.r.t the previous version.
12.AxisEntity	
13.JFreeChartEntity	
14.PlotEntity	
15.TitleEntity	

Chapter 3

PROBLEM STATEMENT

3.1 Problem Definition

Maintainability depends upon different metrics like Lines of code (LOC), cyclomatic complexity and halstead volume. So, by calculating these metrics, one can predict maintainability of software. An open source software JFreeChart is used with 15 different versions and the versions are compared with each other by measuring metrics such as LOC, McCabe's complexity, Halstead volume, Maintainability Index etc using JHawk tool**.**

3.2 Problem Analysis

Various versions of JFreeChart have been studied and the variations in number of classes are observed during each version. Five packages have been considered and the evolution of number of classes with the corresponding versions is observed. These Five packages are annotations, axis, event, block and entity.

Annotations:

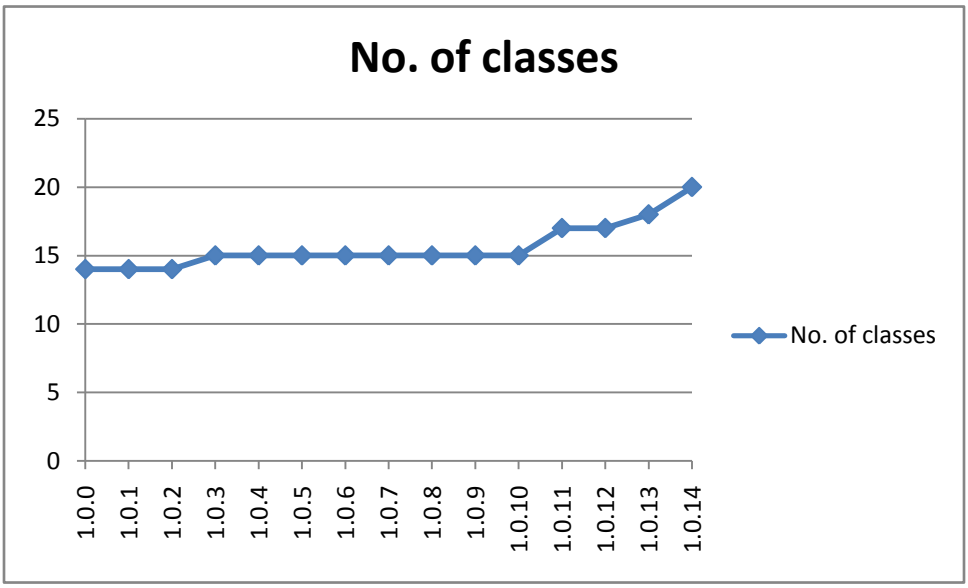

Figure 3.1 No. of classes of Annotations package

Axis:

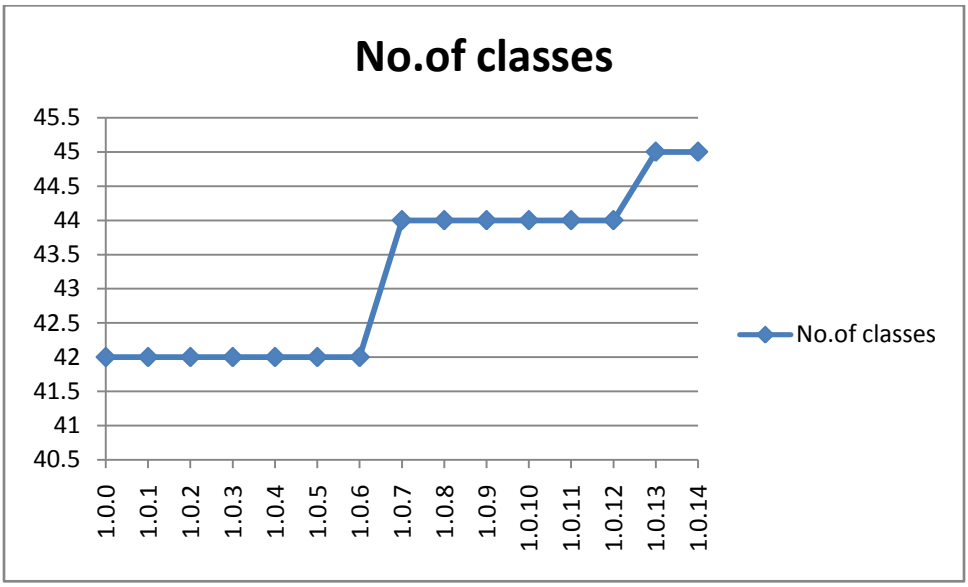

Figure 3.2 No. of classes of Axis package

Block:

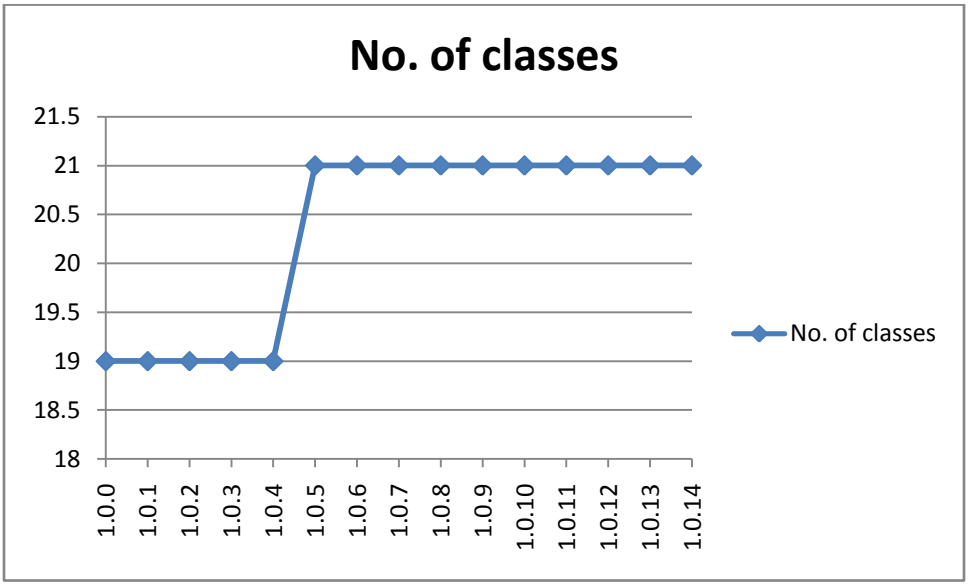

Figure 3.3 No. of classes of Block package

Event:

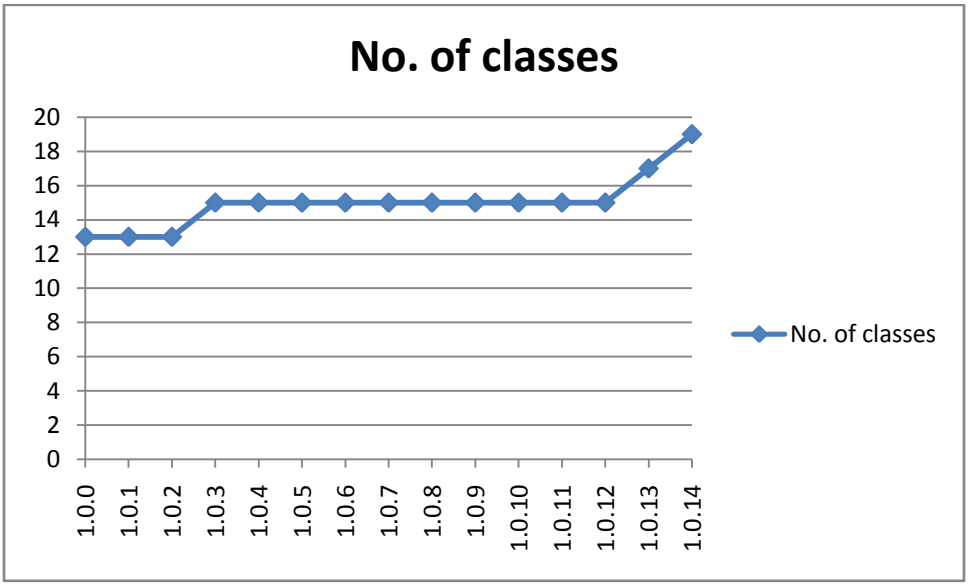

Figure 3.4 No. of classes of Event package

Entity:

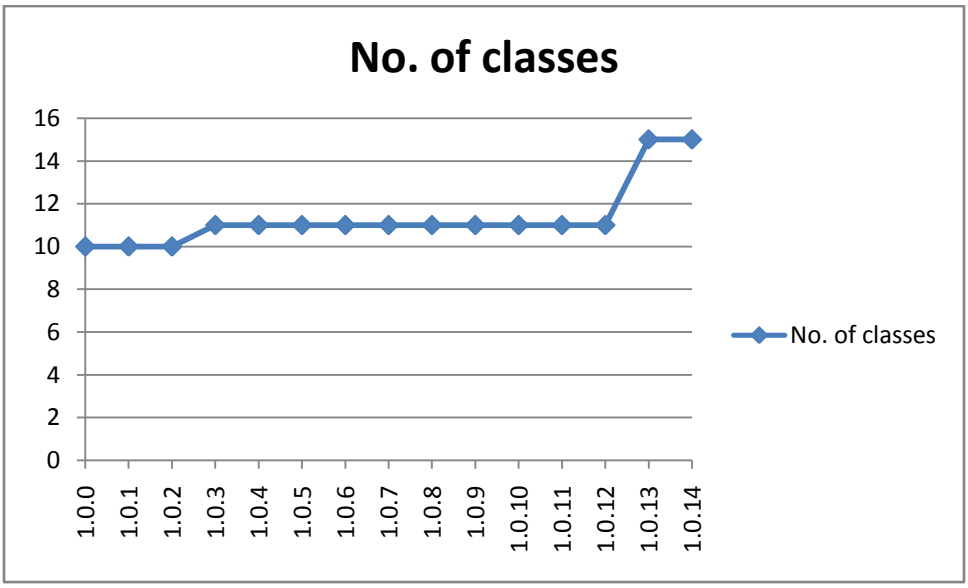

Figure 3.5 No. of classes of Entity package

The variations of the number of classes have been analyzed. Now to completely analyze how the different metrics like LOC, cyclomatic complexity, halstead volume affects the maintainability of a software, these metrics will have been measured using JHawk tool.

Chapter 4

Experimenting View of JHawk Tool

4.1 Introduction of JHawk Tool

JHawk was released in 1999 and has become leader in provision of software metrics to java developer. JHawk is a Java based open source framework which can be integrated in any java application. JHawk executes the modules and generates a graphical report which can be analyzed to find different metrics of an application [25].

4.2 JHawk Licensing Options

All licenses of JHawk are for one year. It does not mean that the product will stop to work after one year. It means that you will receive the update of product without any cost for one year after purchasing the license. You will be asked to extend the license for one more year before year completion– if you accept this offer then you will continue to receive updates for another year. If you do not accept the offer before the end of the year you will not receive any updates and if you desire to accept the most recent copy of JHawk you will have to purchase a new license. On the other hand, Academic Research licensees will obtain free updates for 3 years [26].

4.3 Snapshots of JHawk Tool

Figure 4.1 shows the Welcome tab which provides the fast access to different features of JHawk tool. Initially only the ‘Select Files’ tab and the ‘Preferences’ tab are enabled. The ‘Results’ and ‘Export’ tabs are disabled until a set of Java files have been analyzed. There are four options on the Welcome tab. Any of the option can be selected by pressing icon corresponding to it.

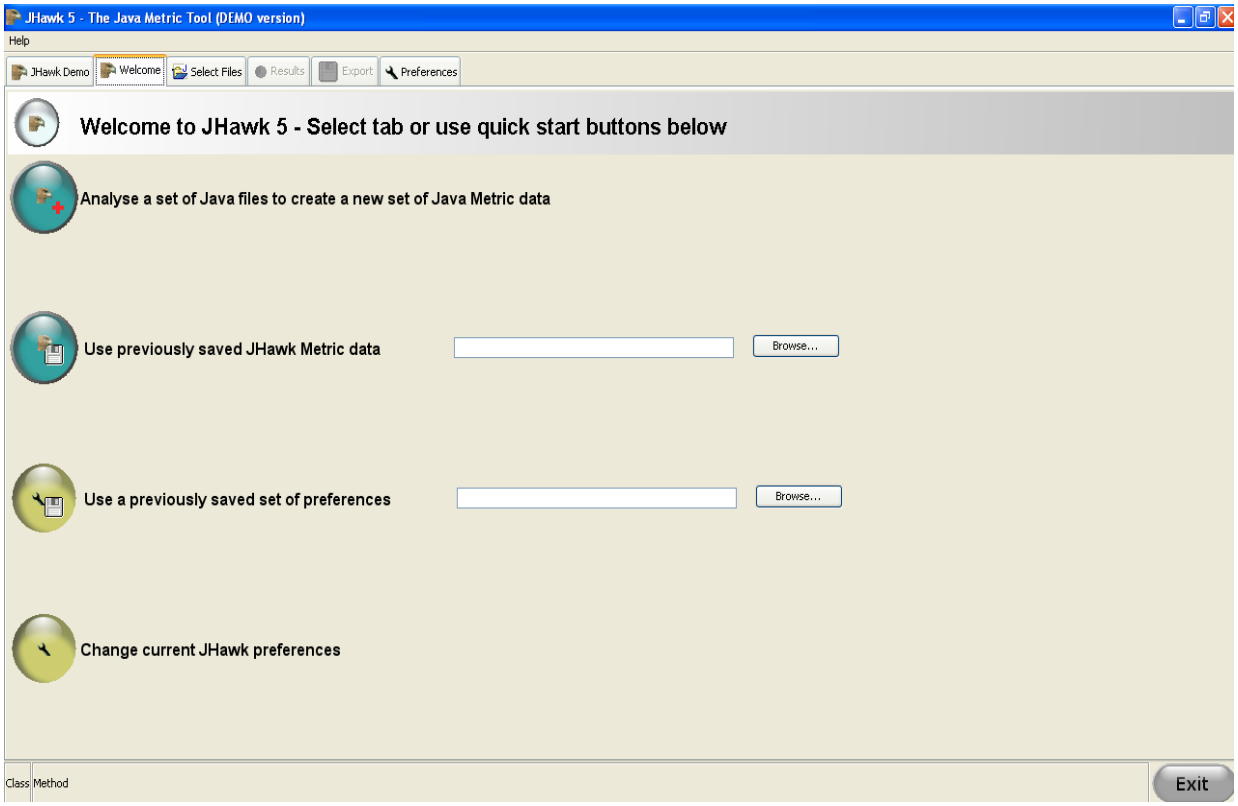

Figure 4.1 View of Welcome tab

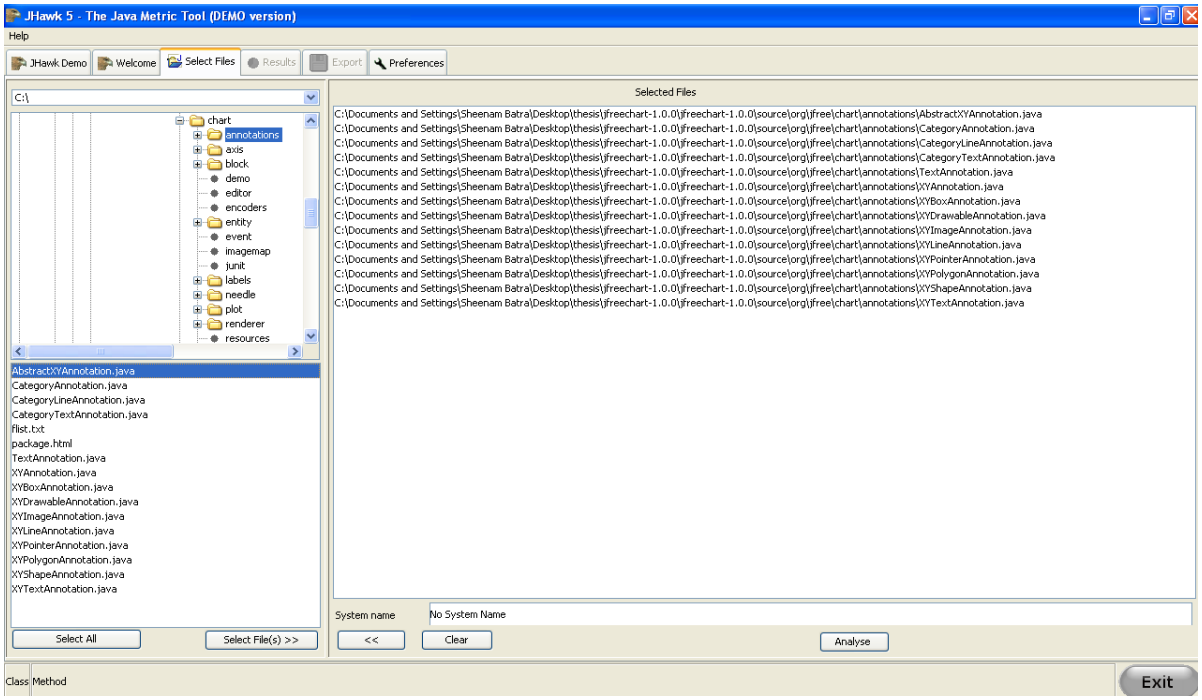

Figure 4.2 View of Select files tab

Figure 4.2 shows the “select files” tab which allows you to select the java files that we want to analyze. To analyze all these files select the root directory of your source code then press the ‘Select All’ button ,then all the Java file paths listed in the ‘Selected Files’ tab on the right hand side. But only the ‘.java’ files in the directories will be selected. To analyze all these files press the ‘Analyse’ button and wait until the other tabs change from the ‘disabled’ state to their ‘enabled’ state.

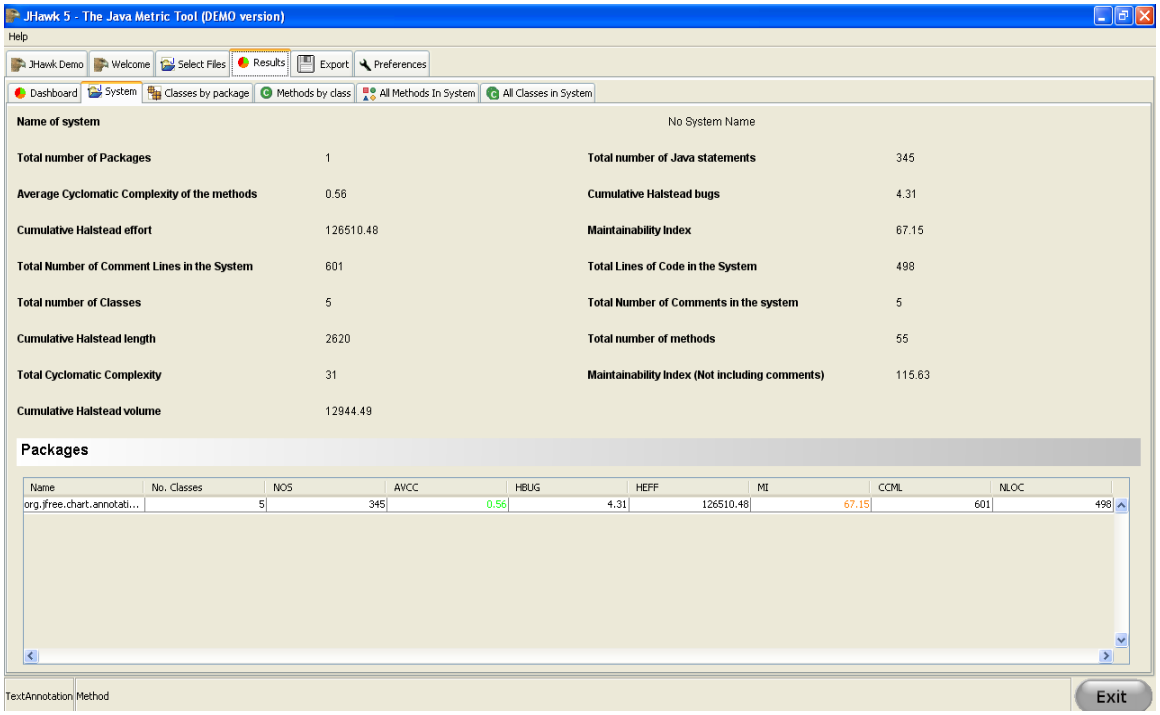

Figure 4.3 View of System tab

Figure 4.3 shows the System sub-tab which shows metrics data at the System level and presents metrics data for each of the packages in tabular form.

Chapter 5

Results and Discussion

As in this dissertation, open source software JFreeChart has been taken. Fifteen different versions of JFreeChart have been studied. Using JHawk tool four parameters have been calculated like lines of code, cyclomatic complexity, halstead volume, maintainability index on which maintainability depends. Five common packages of each version have been taken. These packages are Annotations, Axis, Block, Event and Entity. Graphs of different calculated parameters of each package corresponding to each version have been plotted and studied the comparison of each parameter from one version to another and how it affects the maintainability of product.

The study investigated the relationships between a number of metrics and maintainability based on different number of versions of java open source software. More specifically, four metrics have been calculated from fifteen versions of open source software written in Java.

The results show that the metrics that have taken are strongly related to the maintainability of open source software. When used together, these metrics can predict the maintainability of software quite accurately.

Further are the graphs of different packages. The x-axis denotes different versions of open source software and y-axis denotes the corresponding parameters like LOC, cyclomatic complexity, halstead volume and maintainability index in each of the graph.

Annotations

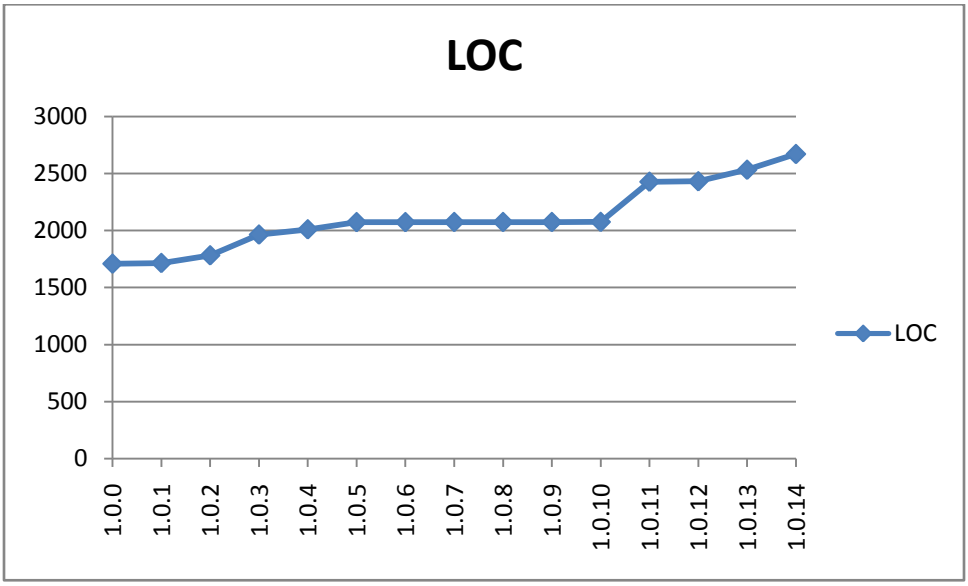

Figure 5.1 LOC of Annotations package

The trend in LOC shows a consistent increase from version 1.0.0 to version 1.0.5. However, it remains constant from version 1.0.6 to version 1.0.10. After that, a major increase in LOC observed from version 1.0.10 to version 1.0.11 and following a slight increase from version 1.0.11 to version 1.0.14. The whole trend shows that there have been some additions in initial 6 versions by the way of incorporation of new code. Then, the product gets stabilized. The study of first six 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.

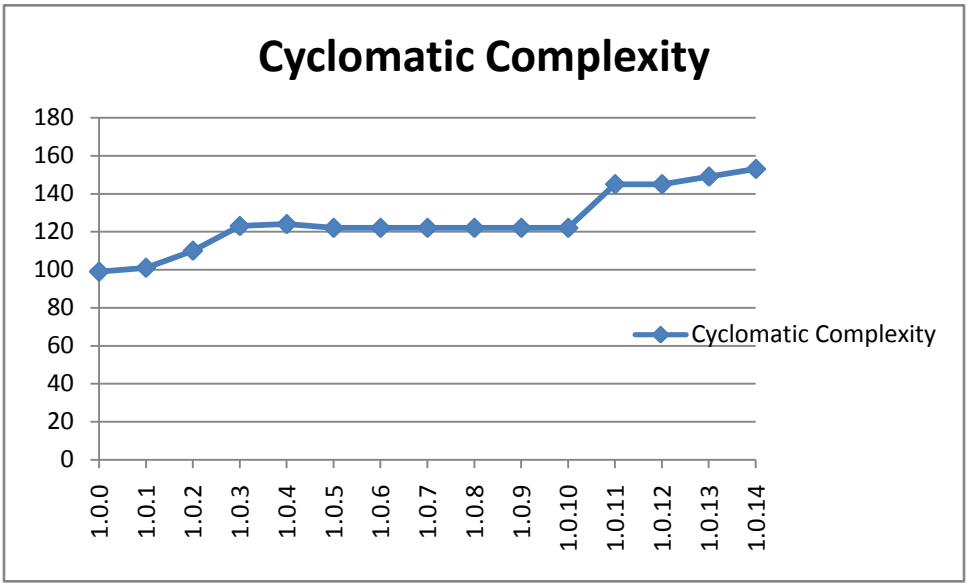

Figure 5.2 Cyclomatic Complexity of Annotations package

At the initial stages of the product, the McCabe's complexity is reasonable at level 100 from version 1.0.0 to version 1.0.1. Then it rise to level 110 in version 1.0.2. Again it rises from version 1.0.2 to version 1.0.4. After then it slightly decreases in version 1.0.5 and remains constant till version 1.0.10. Then it again increases in version 1.0.11 and remains same in version 1.0.12. In version 1.0.13 it gets rises to level 149. The maximum complexity is shown in version 1.0.14 which can be attributed to the rise of number of lines of code in the same version. Therefore, by looking at the trends in change in LOC and McCabe's complexity, it is concluded that as the LOC increases, the complexity also increases. Complexity in this case is in direct relationship with the change in LOC.

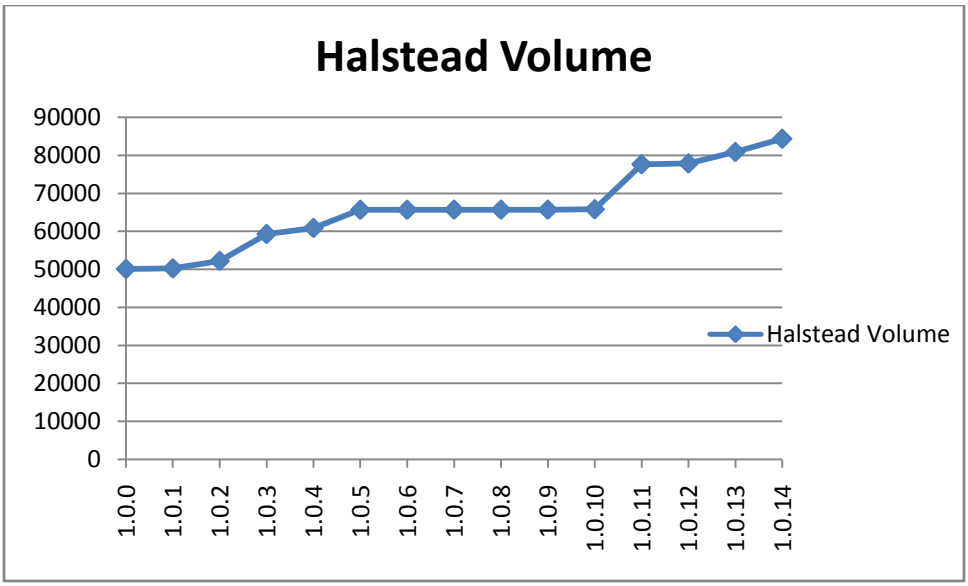

Figure 5.3 Halstead Volume of Annotations package

As shown in above graph, halstead volume is at level about 50000 in version 1.0.0 and version 1.0.1. After that a slight increase observed from version 1.0.1 to version 1.0.5. Then it remains constant in version 1.0.5 and 1.0.6. Then it again increases slightly in version 1.0.7 and get stabilized up to version 1.0.9. Again it increases from version 1.0.10 to 1.0.14. So it can be concluded that halstead volume also varies according to LOC and cyclomatic complexity.

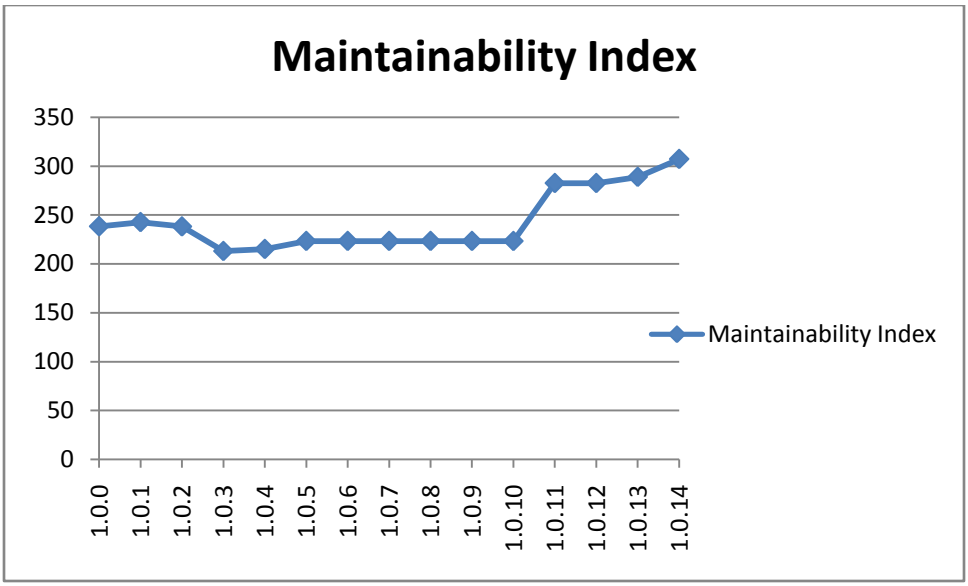

Figure 5.4 MI of Annotations package

Maintainability Index (MI) is calculated using the following formula

Maintainability Index = 171 - 5.2 * avg (Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2*avg (Lines of Code)

Where Halstead volume is a measure calculated from the number of operators and operands in the module.

Software is considered maintainable if its maintainability index is in a higher range. The graph shows the variations of MI. In version 1.0.0 MI is at level 238. Then it slightly increases in version 1.0.1. After that value of MI gets decreases in version 1.0.2 to version 1.0.3. Again it increases from version 1.0.3 to version 1.0.5 and get stabilized up to version 1.0.10. Finally it increases from version 1.0.10 to version 1.0.14. Maximum value of MI shown in version 1.0.14 which shows that version 1.0.14 is more maintainable as compared to previous versions.

Axis

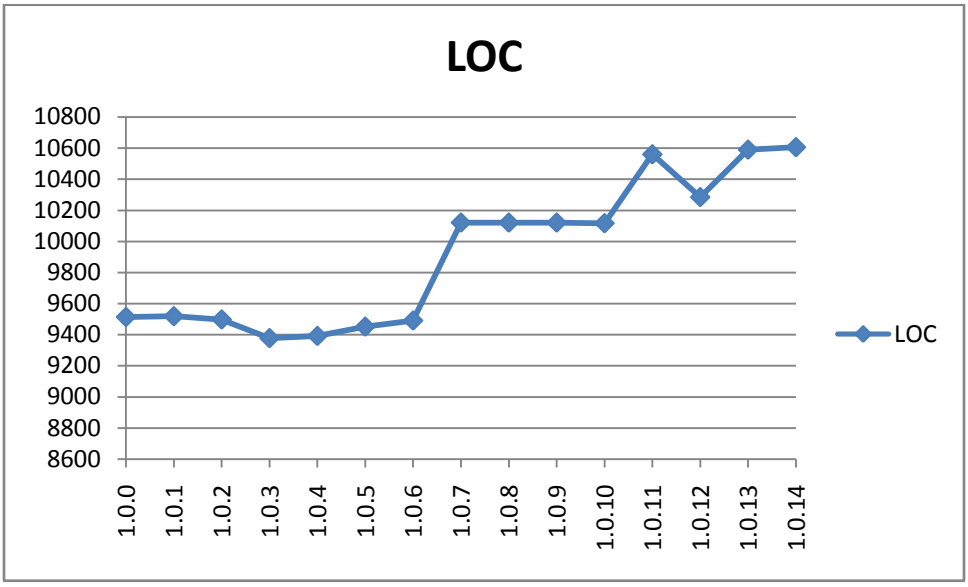

Figure 5.5 LOC of Axis package

The above graph shows the LOC variations in axis package. As graph shows that value of LOC is almost at same level in version 1.0.0 and version 1.0.1. Then it get decreases from version 1.0.1 to version 1.0.4 because in these versions no new additions have been done to the product. Some changes have been done by modifying the existing code. Again from version 1.0.4 to version 1.0.7 value of LOC get increases and get stabilized up to version 1.0.10. After then it increases in version 1.0.11 and get decreases in version 1.0.12. Finally from version 1.0.12 to version 1.0.14, it gets increases.

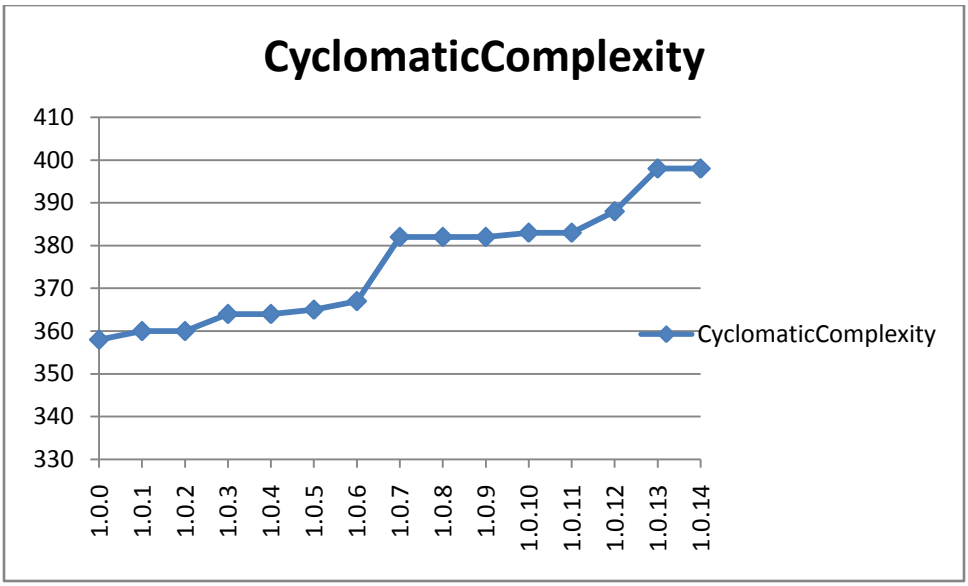

Figure 5.6 Cyclomatic Complexity of Axis package

As above graph shows, value of cyclomatic complexity is at level 358 in version 1.0.0. Then it slightly increases in version 1.0.1 and remains same in version 1.0.2. Again a slight increase observed in version 1.0.3 and value remains constant in version 1.0.4. Again it increases from version 1.0.4 to version 1.0.7. After that value gets increases in version 1.0.7 and gets stabilized till version 1.0.9. Then a small increase observed in version 1.0.10 and remains same in version 1.0.11. Again from version 1.0.11 to version 1.0.13 it gets increases and get stabilized towards end.

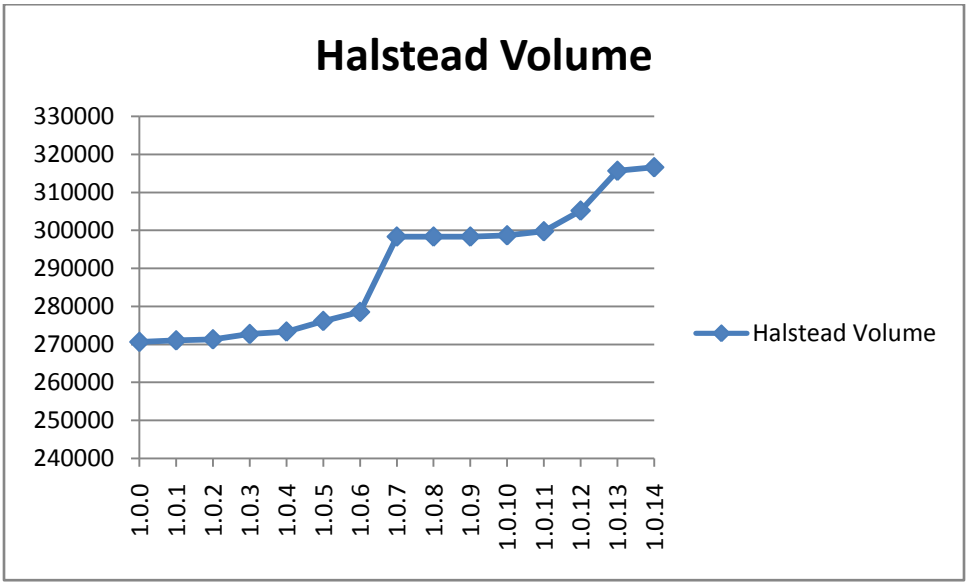

Figure 5.7 Halstead Volume of Axis package

As above graph shows Halsted Volume varies similarly as Cyclomatic complexity and LOC. In version 1.0.0 halstead volume is at level about 27000. After that it slightly increases from version 1.0.1 to version 1.0.7 and remains constant from version 1.0.7 to version 1.0.9. Again value of halstead volume increases from version 1.0.9 to version 1.0.14. So it can be concluded that halstead volume is in direct relationship with LOC and cyclomatic complexity.

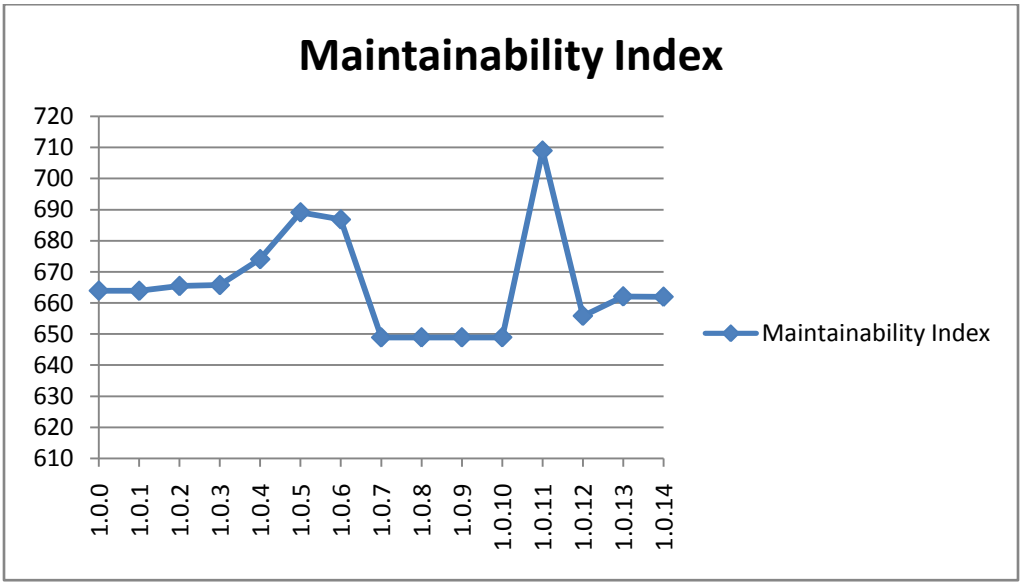

Figure 5.8 MI of Axis package

Above graph shows the variations of MI in axis package. Value of MI remains constant in version 1.0.0 and version 1.0.1. Then it slightly increases in version 1.0.2 and remains constant up to version 1.0.3. Again it increases from version 1.0.3 to version 1.0.5. Then from version 1.0.5 to version 1.0.7, value of MI decreases and get stabilized from version 1.0.7 to version 1.0.10. In version 1.0.11 maximum value of MI observed. Again it decreases in version 1.0.12. Finally, from version 1.0.12 to version 1.0.14 value of MI get increases slightly. So from above graph it can be concluded that version 1.0.11 is more maintainable as compared to other versions.

Block

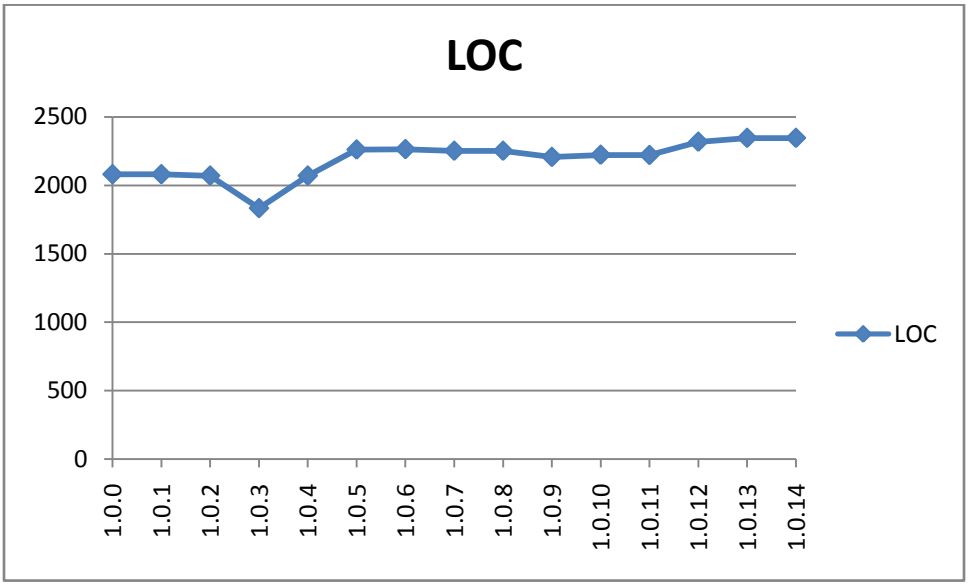

Figure 5.9 LOC of Block package

The above trend in LOC shows consistent behavior from version 1.0.0 to version 1.0.2. Then there is slight decrease in value of LOC observed from version 1.0.2 to version 1.0.3 because no new features added during these versions despite some bug fixes. Then again an increase observed from version 1.0.3 to version 1.0.6. After then it decreases slightly in version 1.0.7 and remains constant till version 1.0.8. Then in version 1.0.9 it gets decreases. Again it slightly increases from version 1.0.9 to version 1.0.13 and get stabilized till version 1.0.14. So it can be concluded that there is no significant enhancements done to the product apart from some bug fixes.

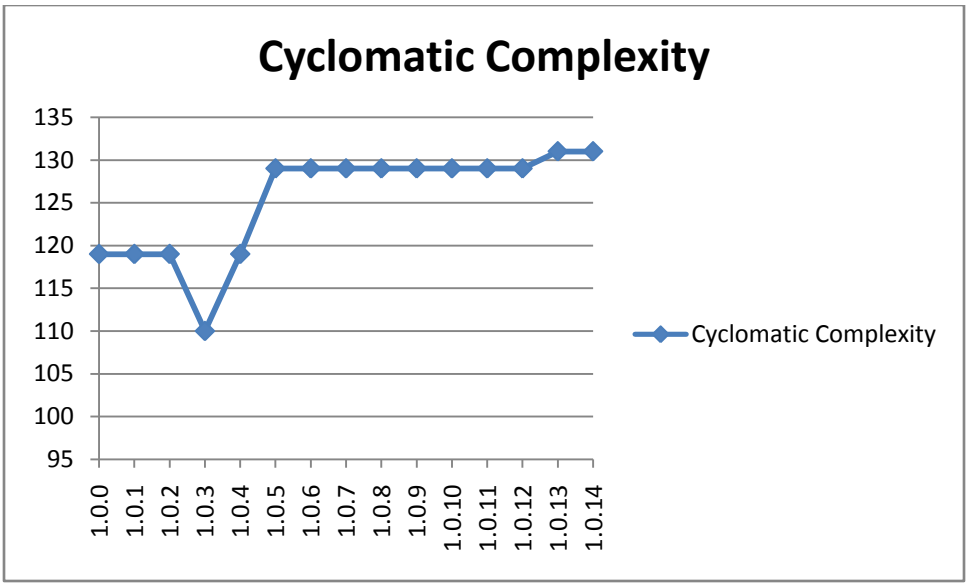

Figure 5.10 Cyclomatic Complexity of Block package

The trend in cyclomatic complexity also shows the similar variations as of LOC. It remains constant at level 119 during version 1.0.0 to version 1.0.2. Then it decreases in version 1.0.3 to level 110. Again it rises from version 1.0.3 to version 1.0.5 and gets stabilized up to version 1.0.12. Ultimately it increases from version 1.0.12 to version 1.0.14.

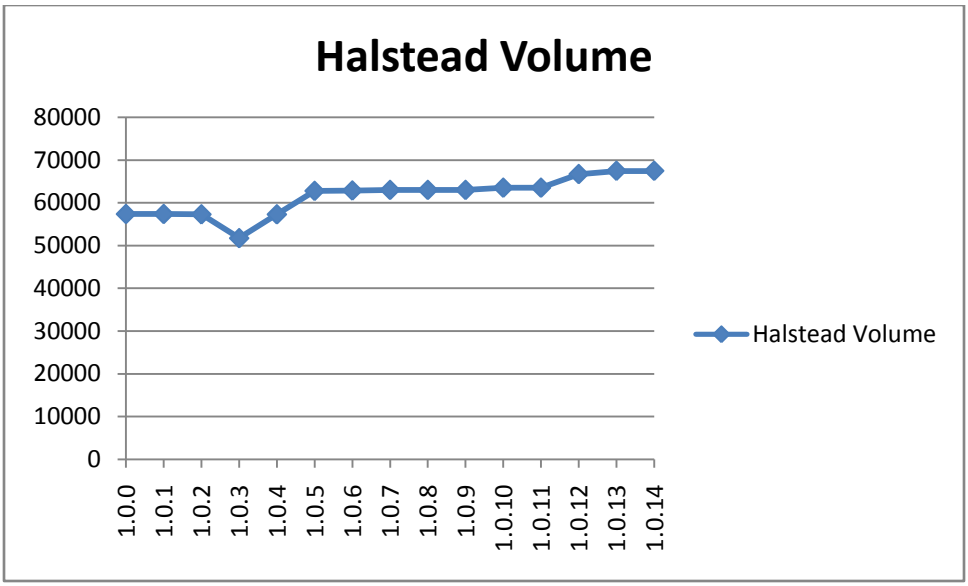

Figure 5.11 Halstead Volume of Block package

Above graph of halstead volume also shows the similar variations as of LOC and cyclomatic complexity. Value of halstead volume shows consistent behavior from version 1.0.0 to version 1.0.2. Then it gets decreases in version 1.0.3. After then it increases from version 1.0.3 to version 1.0.7 and remains constant till version 1.0.9. Finally it get increases from version 1.0.9 to version 1.0.13 and get stabilized towards version 1.0.14.

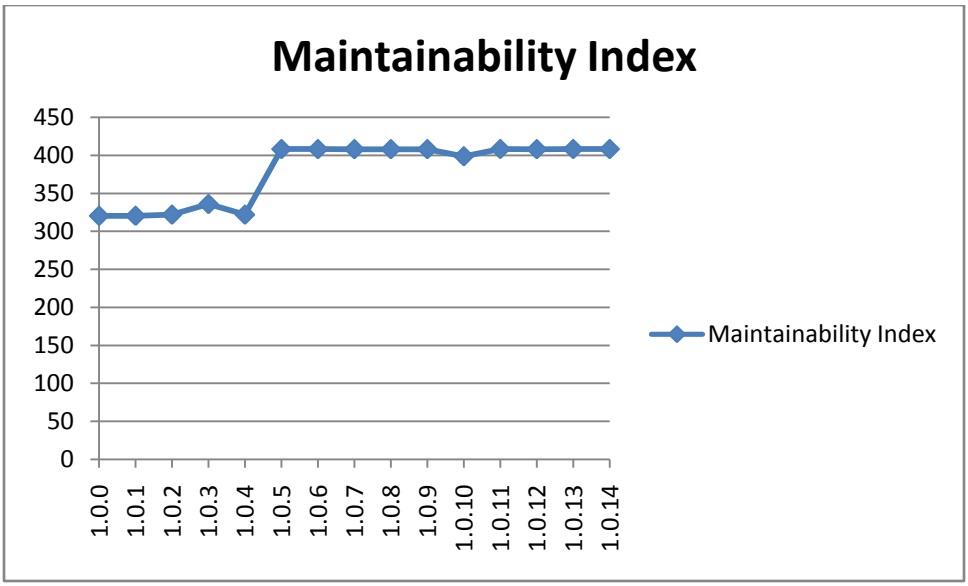

Figure 5.12 MI of Block package

Maintainability index depends upon different metrics like LOC, cyclomatic complexity and halstead volume. So maintainability index also varies according to these metrics. As graph shows value of MI remains constant from version 1.0.0 to version 1.0.2 and increases in version 1.0.3. Then a slight decrease observed in version 1.0.4. Then it increases in version 1.0.5 and remains constant till version 1.0.9. Then a slight decrease in value of MI during version 1.0.10 causes the product to be less maintainable. Again it increases in version 1.0.11 and remains constant towards version 1.0.14.

Event

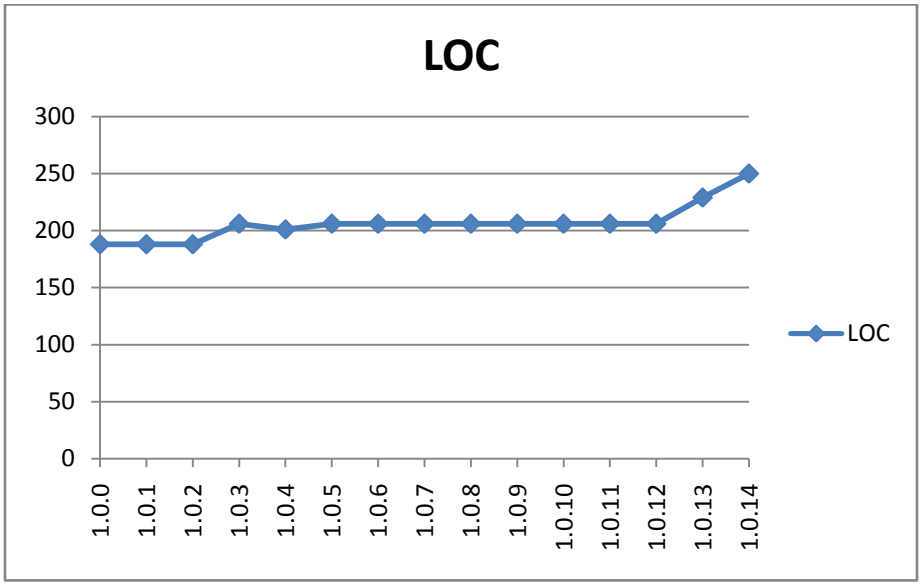

Figure 5.13 LOC of Event package

As LOC trend in above graph shows consistent variations from version 1.0.0 to version 1.0.2 because no new classes added during these versions. Only few bug fixes are there due to some change in existing code. Then it shows an increase in version 1.0.3 because two new classes have added which causes LOC to be high during this version. After then a slight decrease observed from version 1.0.3 to version 1.0.4 and increase from version 1.0.4 to version 1.0.5 and remains constant till version 1.0.12 because no new features have been added during these versions. At the end it shows a slight increase during version 1.0.13 to 1.0.14 because of addition of two new classes which caused lines of code to be high.

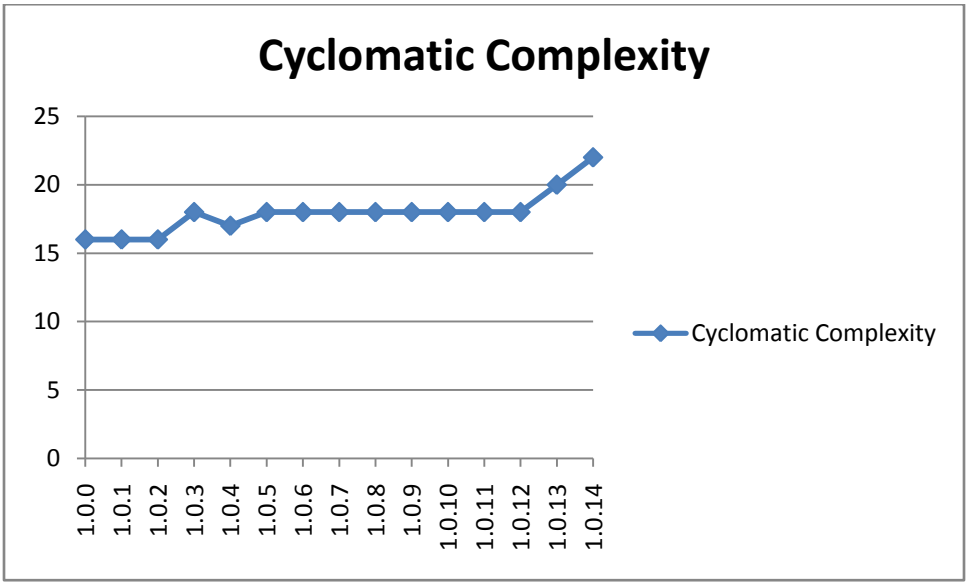

Figure 5.14 Cyclomatic Complexity of Event package

As cyclomatic complexity is in direct relationship with LOC, it also shows same variations as of LOC. As graph shows value of cyclomatic complexity remains constant from version 1.0.0 to version 1.0.2. Then it slightly increases in version 1.0.3. Again it gets decreases during version 1.0.4. After then it increases in version 1.0.5 and remains constant till version 1.0.12. Finally it rises from version 1.0.12 to version 1.0.14.

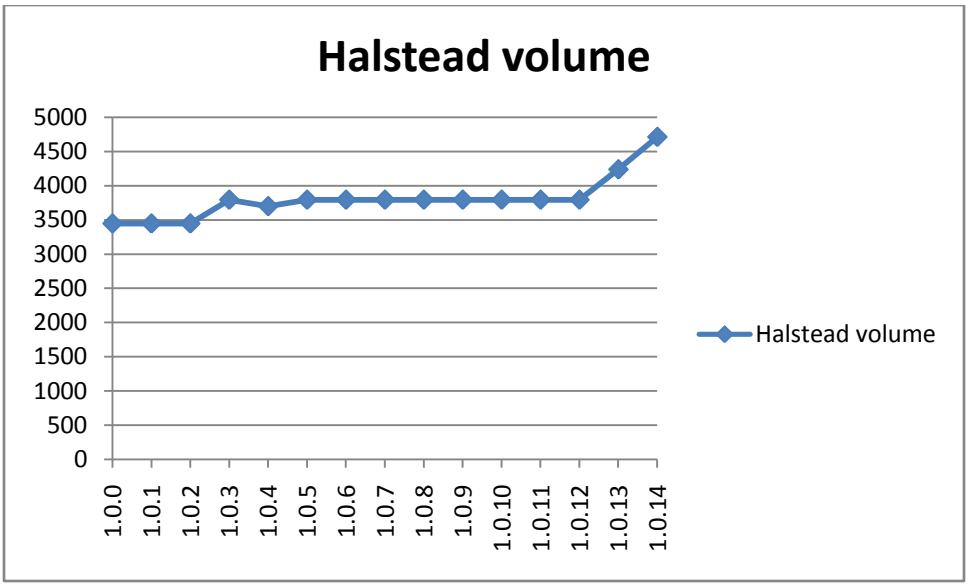

Figure 5.15 Halstead Volume of Event package

Figure 5.15 shows the variations in Halstead volume. Initially, it remains constant at level 3500 from version 1.0.0 to version 1.0.2. Then a small increase observed in version1.0.3. In version 1.0.4, it gets decreases. Again its value rises in version 1.0.5 and gets stabilized till version 1.0.12. Ultimately value of halstead volume increases from version 1.0.12 to version 1.0.14. It can be concluded that halstead volume is in direct relationship with LOC and cyclomatic complexity and shows the similar variations.

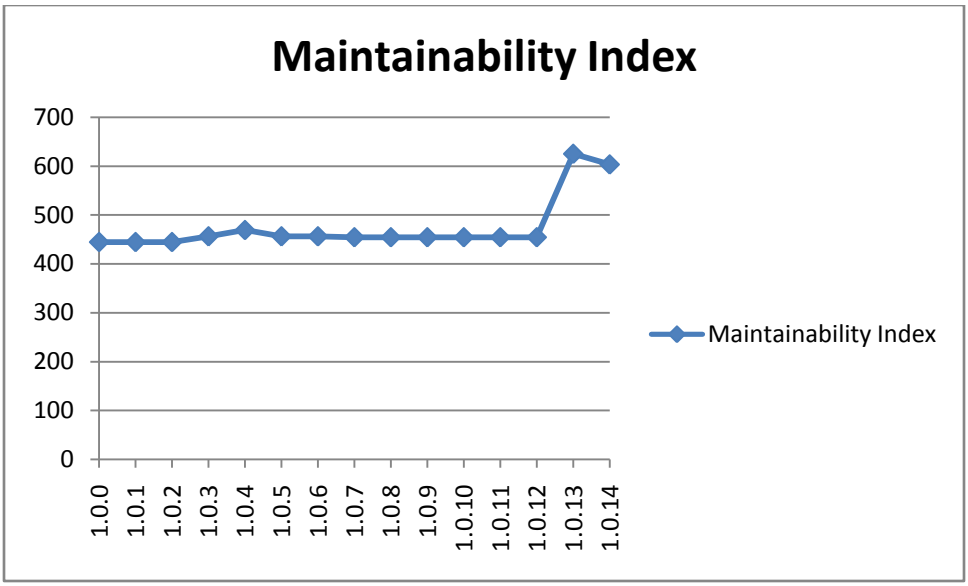

Figure 5.16 MI of Event package

Graph shows value of MI gets stabilized from version 1.0.0 to version 1.0.2. Then a small increase in MI from version 1.0.2 to version 1.0.4 caused the product to be more maintainable than previous versions. Then during version 1.0.5 value of MI gets decreases and remains same in version 1.0.6. Again its value gets decreases during 1.0.7 version and remains consistent till version 1.0.12. After that major increase in version 1.0.13 observed. Then MI value decreases in version 1.0.14 which means the product is less maintainable than previous versions because of increase in LOC.

Entity

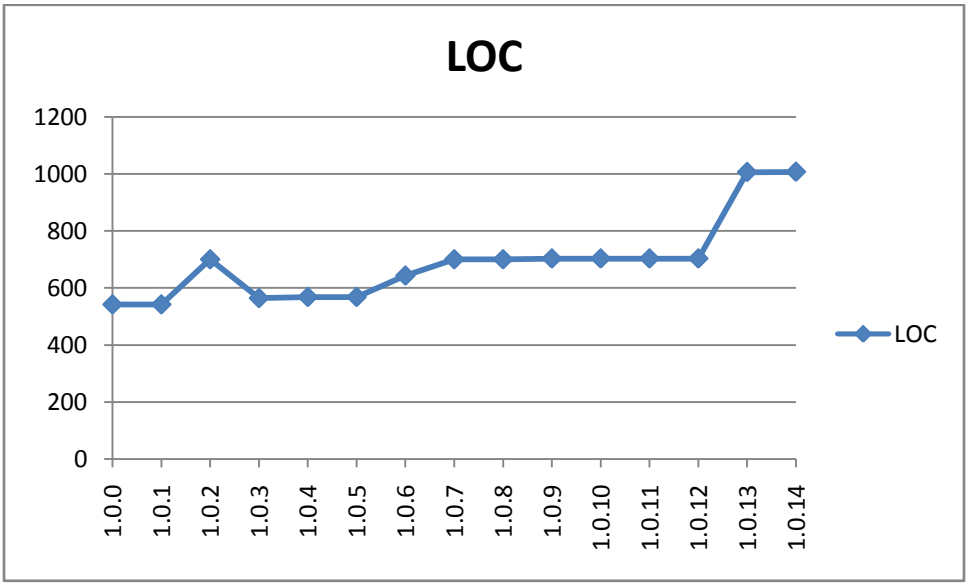

Figure 5.17 LOC of Entity package

Above graph shows LOC variations for Entity Package. As graph shows LOC remains constant from version 1.0.0 to version 1.0.1 because in these versions no new features have been added. Then in 1.0.2 version, one new class has added due to which number of lines of code rises. After then during version 1.0.3 LOC value gets decrease before being stabilized. Then a slight increase observed in version 1.0.4 and remains constant till version 1.0.5 due to some bug fixes but no new class has added to these versions. Then it increases from version 1.0.5 to version 1.0.7 and get stabilized till version 1.0.12. Again in version 1.0.13, value of LOC rises because there are four new classes has been added to this version. Slight increase observed in version 1.0.14 which means that not much enhancements were done in this version.

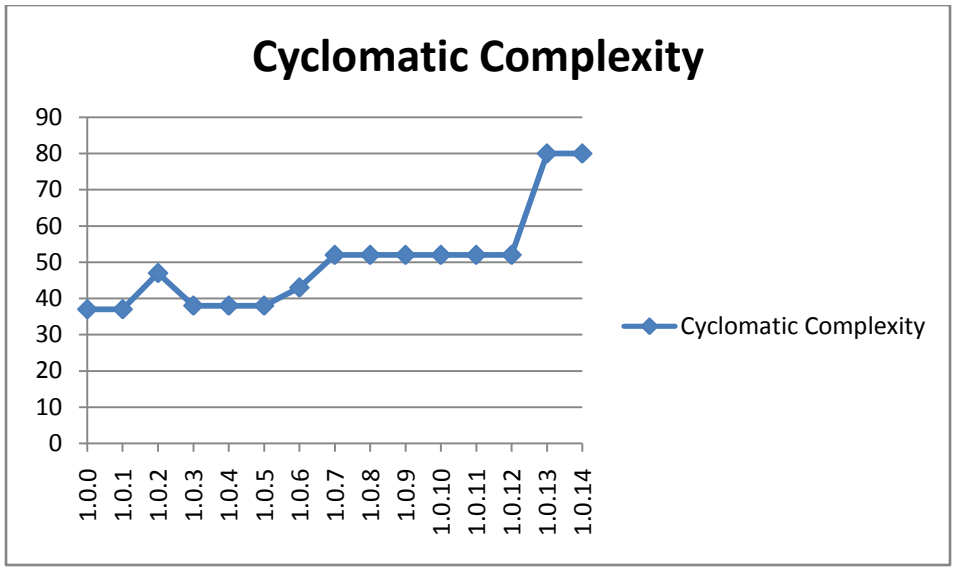

Figure 5.18 Cyclomatic Complexity of Entity package

Value of cyclomatic complexity shows constant behavior from version 1.0.0 to version 1.0.1. Then it increases during version 1.0.2. After then its value gets decreases during version 1.0.3 and remains same till version 1.0.5. Then it increases from version 1.0.5 to version 1.0.7 and remains consistent till version 1.0.12. Again it increases in version 1.0.13 and shows constant behavior towards version 1.0.14.

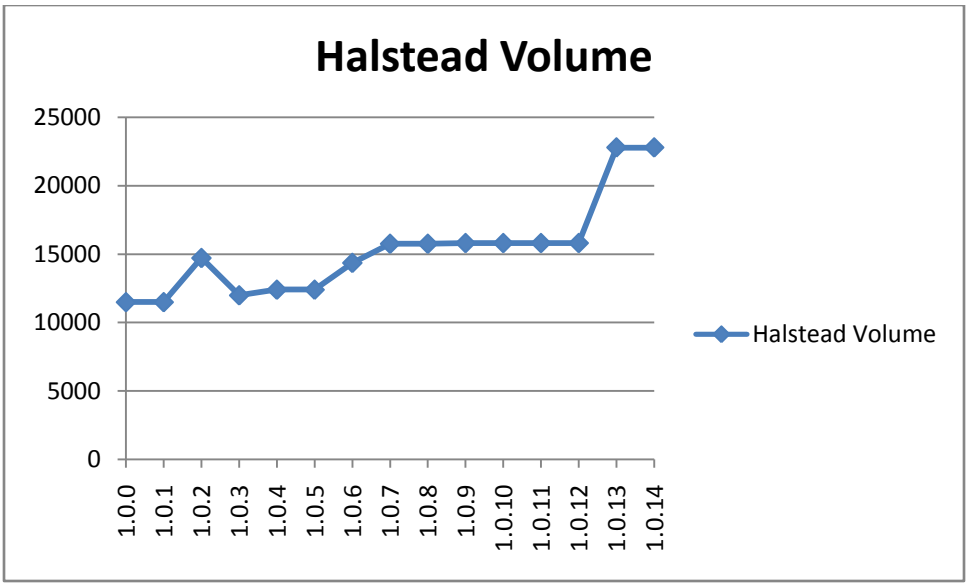

Figure 5.19 Halstead Volume of Entity package

Above graph shows the variations in Halstead volume. Initially, it remains constant at level 11000 from version 1.0.0 to version 1.0.1. Then trend shows a small increase in 1.0.2 version but again falls in version 1.0.3. Then again a small increase observed during version 1.0.4 and remains constant in version 1.0.5. After then value of halstead volume increases from version 1.0.5 to version 1.0.7 and gets stabilized till version 1.0.12. Then it shows increasing pattern from version 1.0.12 to version 1.0.13 and remains constant in version 1.0.14. Halstead volume is in direct relationship with LOC and cyclomatic complexity and shows the similar variations.

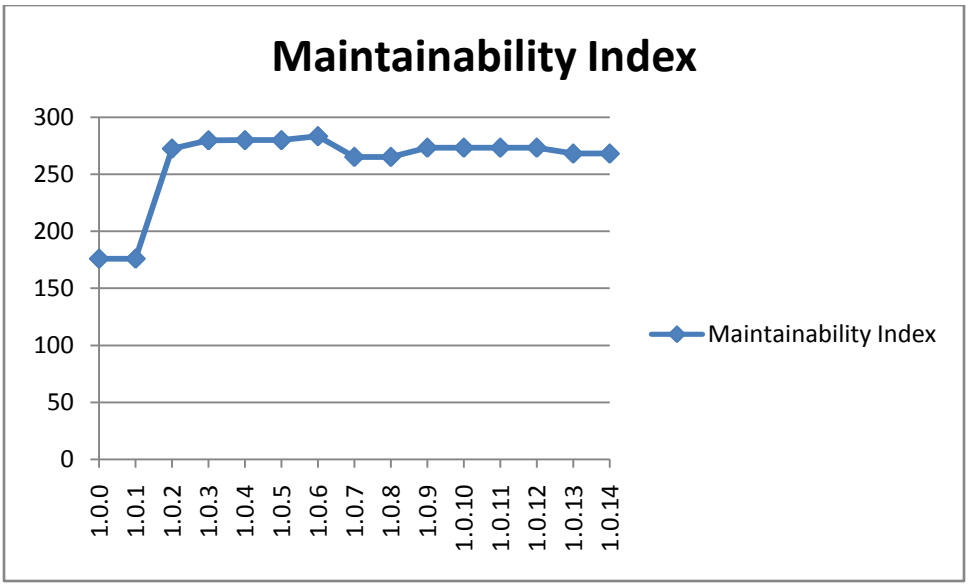

Figure 5.20 MI of Entity package

As graph shows value of MI get stabilized from version 1.0.0 to version 1.0.1. Then a sudden increase in value of MI from version 1.0.1 to version 1.0.3 observed and the value get stabilized from version 1.0.4 to version 1.0.5 .Then during version 1.0.6 value again rises to the level of 283. After then it decrease till version 1.0.8.In version 1.0.9 value of MI again increases and get stabilized till version 1.0.12. During version 1.0.13 and 1.0.14, value again get decreases which means the product is less maintainable as compared to previous versions.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

In this thesis, based on a data set of 15 versions of Java open source software JFreeChart, the relationships between different metrics and the maintainability of open source software have been investigated. The work not only analyzed the influences of individual metrics, but also reported their ability to predict how maintainable a system is, when these metrics are used together. Results have shown these metrics are strongly related to the maintainability of open source software.

The result shows that the increase in line of code, cyclomatic complexity and halstead volume will decrease the maintainability of software. If 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 cyclomatic complexity. Similarly halstead volume also varies as LOC and cyclomatic complexity varies. Using these metrics together, one can easily predict that how maintainable a system is.

6.2 Future Scope

As different versions of open source software, JFreeChart have been analyzed. Fifteen versions of this software 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 software with large versions can 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.

REFERENCES

[1] Otso Kivekas, "Free/Open Source Software Development: Results and Research Methods", University Of Helsinki, June 2008.

[2] "Open source software", *http://www.linfo.org/open_source.html*.

[3] "Free software", *http://www.gnu.org/philosophy/free-software-for freedom.html.*

[4] Nathan Newman, "The Origins and Future of Open Source Software".

[5] C.J.Xiong, Y.F.Li, M. Xie, S.H. Ng, T.N. Goh, "A Model of Open Source Software Maintenance Activities", Proceedings of the 2009 IEEE IEEM, 2009.

[6] Prof. Dr. Sabina Jeschke, Dr. Claudia Muller, Sven Grottke, "Lecture on Free/Libre and Open Source Software", Institute of Information Technology Services, University of Stuttgart, 2009.

[7] "Licenses of open source software", *http://www.opensource.org/licenses/agpl-v3.html*.

[8] "Advantages of Open Source Software", *http://open_source.gbdirect.co.uk/migration/ benefit.html.*

[9] "Disadvantages of open source software", *http://open_sourcetechnologies.blogspot. com /2008/ 06/disadvantages-of-open-source-software.html*

[10] Guide to the Software Engineering Body of Knowledge (SWEBOK).IEEE Computer society, Los Alamitos, California, 2004.

[11] Tobias Kuipers, Joost Visser, "Maintainability Index Revisited- position paper", 2007.

[12] M.I.Sarwar, W.Tanveer, I.Sarwar, "Performance of MI Tools in Perspective of Open Source Software", In Proceeding of 2nd International Conference on Digital Object Identifier, pp.1-4, 2009.

[13] "JFreeChart", *http://www.jfree.org/jfreechart.*

[14] "Versions of JFreeChart", *http://sourceforge.net/projects/jfreechart /files/ 1.%20 JFreeChart.*

[15] Paul Karvanagh ,"Open Source Software: Implementation & Management", 2004.

[16] Ismail Ari, "Quantitative Analysis of Open Source Software Projects", 2001.

[17] W.W.Ming, Y.D.Lin, "Open Source Software Development", In Proceeding of the International Symposium on Digital Object Identifier, 2001.

[18] T. koponen, V.Hotti, "Open Source Software Maintenance Process Framework", In Proceeding of 5th Workshop on Open Source Software Engineering, 2005.

[19] T.koponen, Heli Lintula, V.Hotti, "Exploring the Maintenance Process through the Defect Management in the Open Source Projects-Four Case Studies", In Proceeding of International Conference on Software Engineering, 2006.

[20] Timo Koponen, "Evaluation of Maintenance Processes in Open Source Software Projects through Defect and Version Management Systems", University of Kuopio, 2007.

[21] T.Koponen T, V.Hotti, "Defects in Open Source Software Maintenance - Two Case Studies: Apache and Mozilla", In Proceedings of International Conference on Software Engineering 2005.

[22] T.Koponen , V.Hotti "Evaluation Framework of Open Source Software", In Proceedings of The International Conference on Software Engineering Research and Practice Vol. II, pp. 897-902, 2004.

[23] "Documentation of JFreeChart", *http://sourceforge.net/projects/jfreechart/files/2.%20 Documentation.*

[24] "Packages of JFreeChart",*www.jfree.org/jfreechart/api/javadoc/org/.../packagesummary .html.*

[25] "JHawk", *http://java-source.net/open-source/testing-tools/j-hawk.*

[26] "JHawk licensing options", *http://www.virtualmachinery.com/jhawkprod.htm.*