Study of Lehman's Laws and Metrics during Software Evolution

Baljinder Singh, Pawan Luthra

Department of Comp. Science
S.B.S State Technical Campus
Ferozepur, India

Abstract

This paper presents the results of study conducted on 18 versions of Freemind and 12 versions of jFreeChart released over the period of 14 years. We measured Object Oriented Metrics and studied the changes in the measured values over different releases of two medium sized software developed using Java. We also investigated the applicability of Lehman's Laws of Software Evolution on Object Oriented Software Systems using different measures. We observed that the reflection of some of the laws namely law 1 (Continuing Change), law 2 (Increasing Complexity), law 6 (Continuing Growth) and law 7 (Declining Quality) have direct relevance to the computed metrics and were easily determined using the metrics. But the relatedness of law 3 (Self Regulation), law 4 (Conservation of Organizational Stability), law 5 (Conservation of Familiarity) and law 8 (Feedback System) to open source software system was hard to determine and will require more empirical studies with relevant data.

Keywords: *Software Maintenance, Software Evolution, Software Metrics, Lehman's Laws of Software Evolution, Versions, open source software.*

I. Introduction

Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment. This definition reflects the common view that software maintenance is a post-delivery activity: it starts when a system is released and encompasses all activities that keep the system operational. The characteristics of a software system to easily accommodate changes over the time is referred as software evolution [1,2]. Software needs to evolve in order to be used for a longer period.

In recent years many businesses have developed a strong interest and are dependent on software generated information. They are driven and controlled by computers and software that may have been operational for several years. Now specification and design of such systems requires assumptions about the intended application and its operational domain. These, in turn, will be reflected in the software. Subsequently, installation and operation of the system together, be continually updated to maintain their validity and adapt to changed circumstances.

This paper is organized as follows: Section 2 contains the background, giving brief interpretation of Lehman's Laws of software evolution and Object Oriented Metrics. Section 3 explains the methodology used for data collection. Section 4 provides the brief introduction of the two case studies. Section 5 presents the analysis and interpretation of Lehman's Laws. Section 6 contains the related work. Section 7 presents conclusion and future work.

II. BACKGROUND

This section contains two subsections. The first one gives a brief interpretation of Lehman's Laws of Software Evolution and the second one presents the meaning of object oriented metrics used in the study.

A. *Lehman's Laws of Software Evolution*

i. Continuing Change (1974):

E-type systems should continuously be changed in order to be used for longer period or they become less satisfactory. The required change may be called for in response to change in environment, as a bug fix exercise or as a preventive maintenance activity or any other activity leading to change.

ii. Increasing Complexity (1974):

The complexity of an E-type system increases unless some preventive maintenance is done to control it. Increase in complexity may arise due to number of changes or due to addition of more functionalities leading to more interaction.

iii. Self Regulation (1974):

Evolution process of E-type system is self regulatory. This means that growth rate is regulated by the maintenance process. There is a balance between what is desired to be changed and what can actually be achieved. In order to have a smooth evolution process, the limitations on growth rate should be accepted.

iv. Conservation of Organizational Stability (1980):

Evolution process of software conserves the organizational stability. The work rate of an organization evolving large software tends to remain constant. This means it is hard to change the staffs who have been working on evolving software. The average global effective rate in evolving software tends to remain constant over product lifetime.

v. Conservation of Familiarity (1980):

The familiarity with evolving E-type software is conserved. A huge change that might cause lack of familiarity of staff members involved with the evolving software is avoided. For small changes the familiarity of software is easily achieved by the personnel involved with the software. Hence the average incremental growth remains constant as the software evolves.

vi. Continuing Growth (1980):

The functional content of E-type systems must be continually enhanced in response to user feature request in order to maintain user satisfaction over its life period.

vii. Declining Quality (1996):

The Evolution process causes decline in the quality of evolving software.

viii. Feedback System (1996):

According to this law, the evolution process constitutes multi–level, multi agent feedback system. For open source software the law seems to be true since feature request and reporting of bug comes from user community.

B. *Object Oriented Metrics used the study*

In this section we present the definition of package level, class level and method level metrics used in the study.

i. LOC – Lines of Code

LOC includes all the lines of source code, except blanks and comments.

ii. NOC – Number of Children

Number of children of a class *C* is defined as the number of first level subclasses of *C*.

iii. CBO – Coupling Between Object Classes

CBO for class *C* stands for the number of other classes that *C* is coupled. A class is coupled to another if one uses other's methods, attributes or one is inherited from the other.

iv. RFC – Response For a Class

RFC of a class C is the cardinality of the set of methods that belong to C or is invoked by methods of C.

v. DIT – Depth of Inheritance

Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be maximum length from the node to the root of the tree.

vi. LCOM – Lack Cohesion in Methods

The LCOM metric of a class is the count of the sets of methods in a class that are disjoint with respect to members of a class being accessed by them. The original definition of metric is presented at [3].

vii. WMC – Weighted Methods per Class

The WMC metric for a class is the sum of complexities of its methods. The complexity of an individual method can be measured as cyclomatic complexity. We have used cyclomatic complexity variant of WMC.

viii. NOM – Number of Methods

The NOM metric for a class defines the number of methods defined in the class. The count of static method is defined using NSM and the number of overridden methods is defined using NORM.

ix. NOA – Number of Attributes

The NOA metric for a class defines the number of attributes in that class.

x. NPM – Number of Public Methods

The NPM metric for a class defines the number of public methods in a class

xi. MI – Maintainability Index

In [4] Oman and Hagemeister has defined maintainability index in detail. A higher MI value indicates better maintainability. Various components of MI are LOC, Halsted volume, Cyclomatic Complexity [5] and percentage of line of comments.

III. METHODOLOGY USED FOR DATA COLLECTION

The software used in the study are open source software therefore, the availability of source code for different versions to conduct the study with said objective was never a problem. The computation of different metrics was done using a metrics measurement tool. The average measures of the various metrics for releases of FREEMIND and JFREECHART are presented in Tables 1 and 2 respectively. The revision details of FREEMIND and JFREECHART are available at [10] and [11] respectively.

IV. CASE STUDIES

This section contains a brief introduction to the software used in the study, namely FREEMIND and JFREECHART. All the data related to different versions is available on the internet [10] [11]. The choice of systems is mainly guided by the limitations of metrics measuring tools and the availability of revision details and source code.

A. *Case study 1: FREEMIND*

A mind mapper and at the same time as easy-to-operate hierarchical editor with strong emphasis on folding. These two are not really two different things, just two different descriptions of a single application. Often used for knowledge and content management [8].

The first version has been released on 26 June, 2000. Since then 18 versions [10] has been released. The last version has been released on 12 April, 2014. Table 1 gives the details of release of FREEMIND over the period.

Table 1. Details of releases of Freemind

Versions	No of Classes	LOC	AV CBO	AV LCOM	AV RFC	AV MI	AV CC
0.0.2	48	2578	5.52	0.29	21.74	104.27	1.45
0.0.3	50	2770	5.82	0.22	24.53	100.07	1.48
0.1.0	69	3330	4.77	0.25	20.96	110.05	1.61
0.2.0	61	3372	4.9	0.3	24.03	107.97	1.68
0.3.0	73	3713	4.12	0.3	22.25	109.76	1.64
0.3.1	73	3756	4.18	0.3	22.37	109.55	1.65
0.4	80	4427	4.9	0.32	25.22	104.5	1.72
0.5	82	5363	5.7	0.34	29.97	99.57	1.94
0.6	91	5728	5.66	0.35	27.09	97.45	1.85
0.6.1	92	5671	5.51	0.34	27.2	98.4	1.83
0.6.5	104	5386	4.93	0.31	25.66	100.88	1.78
0.6.7	107	5951	4.99	0.3	28.05	101.02	1.9
0.7.1	107	6460	5.19	0.29	29.87	101.92	1.84
0.8.0	100	5384	6.2	0.43	23.31	101.79	1.99
0.8.1	99	5369	6.22	0.42	23.43	102.52	2
0.9.0	115	7952	7.22	0.45	20.92	91.58	2.05
1.0.0	118	8807	7.52	0.43	22.69	89.12	2.08
1.0.1	117	8220	7.38	0.42	21.74	90.86	2.02

B. *Case study 2: JFREECHART*

JFreeChart is a free (LGPL) chart library for the Java platform. 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 [9]. The JFreeChart project was founded fifteen years ago, by David Gilbert. Today, JFreeChart is the most widely used chart library for Java with version 1.0.19 reaching more that 450,000 downloads to date. The project continues to be managed by David Gilbert, with contributions from a diverse community of developers.

It supports bar charts, pie charts, line charts, time series charts, first version 0.5.6 has been released on 1 Dec, 2000. Since then 57 versions [11] has been released. The last version 1.0.19 has been released on 31 July, 2014. The later versions are extremely stable and reliable. Table 2 gives the details of release of JFREECHART over the period.

Table 2. Details of releases of Jfreechart

Versions	No of Classes	LOC	AV CBO	AV LCOM	AV RFC	AV MI	AV CC
1.0.0	49	8414	10.47	0.44	52.92	101.69	2.09
1.0.9	46	10205	12.91	0.57	69.93	93.15	2.43
1.0.10	46	9039	13.24	0.48	63.15	96.3	2.23
1.0.11	45	10287	13.91	0.53	71.38	96.05	2.37
1.0.12	45	10096	14.27	0.53	70.19	95.68	2.25
1.0.13	47	11187	14.23	0.54	77.9	95.15	2.31
1.0.14	43	9795	14.74	0.52	71.96	99.37	2.26
1.0.15	43	9533	15.07	0.52	77.38	99.52	2.07
1.0.16	43	9668	15.05	0.53	75.22	100.4	2.08
1.0.17	43	9768	14.95	0.53	75.11	100.03	2.09
1.0.18	44	9756	14.66	0.5	73.6	100.22	2.08
1.0.19	44	9753	14.66	0.5	73.6	100.22	2.08

V. ANALYSIS AND INTERPRETATION OF FREEMIND AND JFREECHART

In this section, we analyze the Lehman's laws of software evolution based metrics value for various versions of FREEMIND and JFREECHART. Some of the laws have direct relevance to the computed metrics while others did not have direct relevance to the metrics.

A. *Law 1: Continuing Change*

According to this law, evolving software has to adapt to the changing environment or it will become progressively less satisfactory i.e. the software has to continually change in order to be used for longer period of time. Software may change in response to bug fixing activity or in response to change in the environment. It is difficult to differentiate between growth and change. The change in the usage environment may include the addition of some functions or classes, which will increase the size of software leading to growth.

In case of FREEMIND, the version 0.8.0 includes a rewrite of all internal data structures and classes, which makes it faster than version 0.7.1. version 0.8.1 fixes a bug leading to small changes.

Similarly, in case of JFreeChart, version 0.6.0 adds new plots including scatter plot, stacked bar charts and 3D bar charts. It adds improved pie chart. New properties to control spacing on bar charts are added. JFreeChart panel now incorporates buffering, and popup menu. Java docs are revised. It fixed numerous bugs from version 0.5.6. In version 0.8.0 all the category plots are now controlled through the one class (Category Plot) with plug-in renderers. It added a Resource Bundle for user interface items that require localization.

Both softwares involve addition of new classes and bug fix activities which indicates that Law 1 holds for the above two open source software.

B. *Law 2: Increasing Complexity*

According to this law, the complexity of software tends to increase over releases unless measures are taken to keep the complexity in check. It is generally perceived that growth increases complexity, but if the appropriate c hanges are made, the evolution process might not support the law. Therefore it becomes difficult to determine whether the law is reflected by evolution process or not.

In case of object-oriented systems, the complexity of the software system or subsystem can be determined using coupling between the objects (CBO), response for a class (RFC), weighted method per class (WMC) and lack of cohesion of methods (LCOM). The measures of CBO, RFC, WMC and LCOM as complexity measures for object oriented software systems have been validated through empirical research by Gyimothy, Ferenc and Siket [13].

In case of FREEMIND, the increase in complexity can be viewed in table 1 where we see an increase in the values of CBO, RFC, WMC and LCOM. Figure 1, Figure 2, Figure 3 and Figure 4 depict CBO, RFC, WMC and LCOM respectively for FREEMIND.

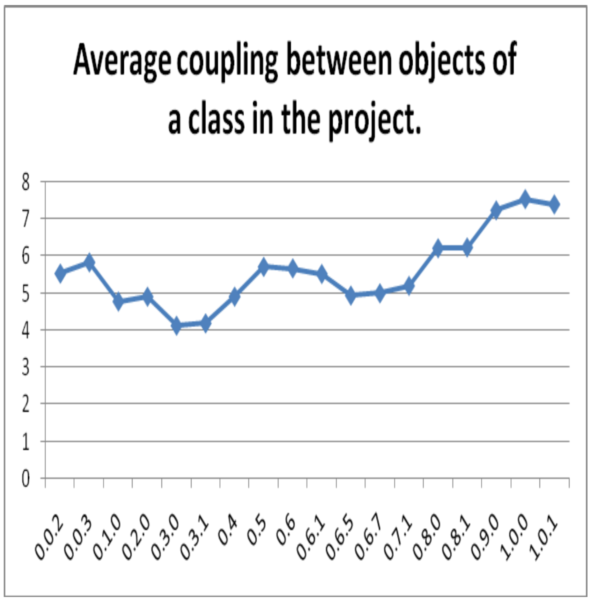

Figure 1. FREEMIND coupling between objects

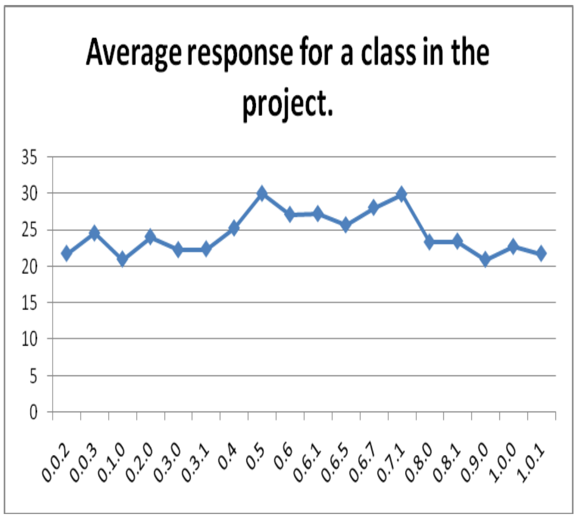

Figure 2. FREEMIND response for a class

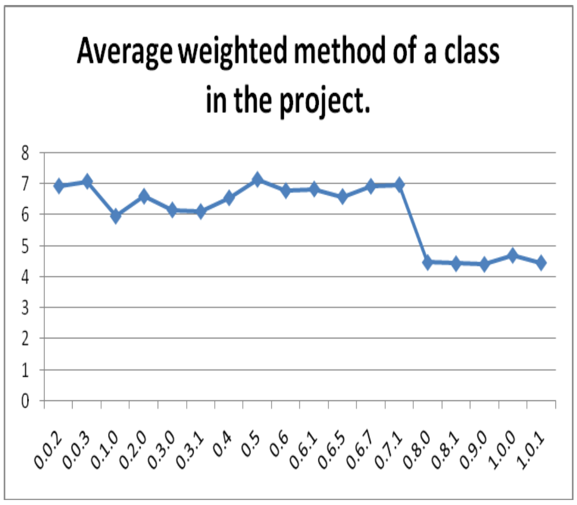

Figure 3. FREEMIND weighted methods of a class

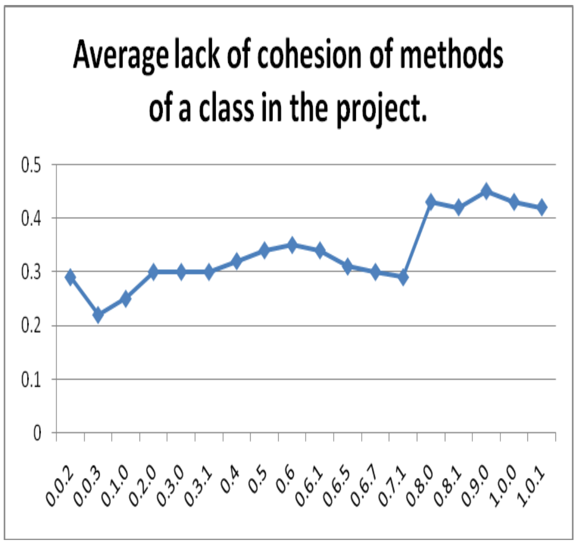

Figure 4. FREEMIND lack of cohesion of methods

In case of JFREECHART, the increase in complexity can be viewed in table 1 where we see an increase in the values of CBO, RFC, WMC and LCOM. Figure 1, Figure 2, Figure 3 and Figure 4 depict CBO, RFC, WMC and LCOM respectively for JFREECHART. Similar observations were also made in [13]. Considering the law to be reflected by FREEMIND and JFREECHART.

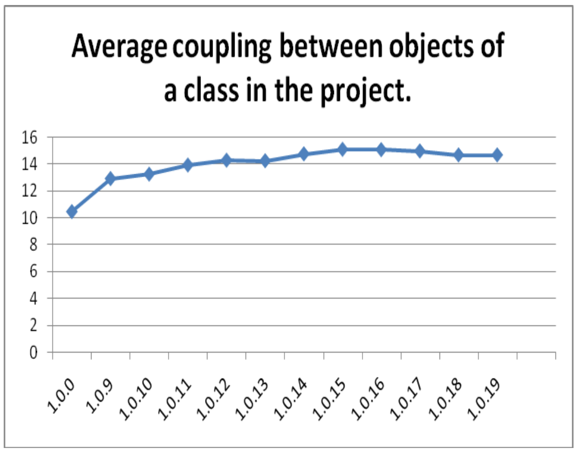

Figure 5. JFREECHART coupling between objects

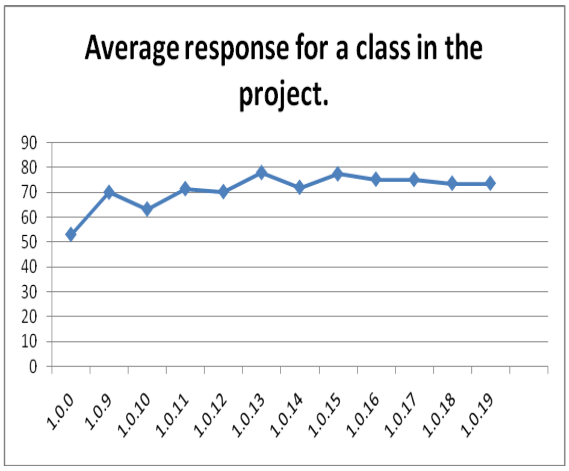

Figure 6. JFREECHART response for a class

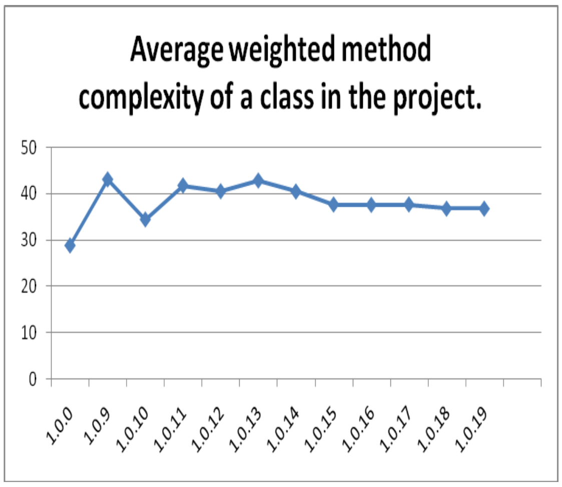

Figure 7. JFREECHART weighted method complexity

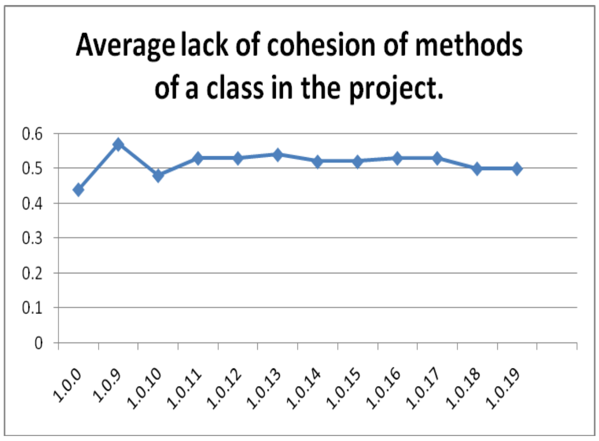

Figure 8. JFREECHART lack of cohesion of methods

C. *Law 3: Self Regulation*

According to this law, the software evolution process is self regulating. This means that growth rate is regulated by the maintenance process. There is a balance between what is desired to be changed and what can actually be achieved. In order to have a smooth evolution process, the limitations on the growth rate should be accepted.

The observations made from Table 1 and Table 2 for FREEMIND and JFREECHART shows a steady increase in size. There are no evidences of huge changes or sharp increases in the size of the code, but still more empirical study is required to find the relevance with Lehman's laws.

D. *Law 4: Conservation of organizational stability*

According to this law, the average global rate of activity on an evolving system is invariant over the product lifetime. Determining the average global rate of activity for open source software is difficult if not impossible. The overall process that results in development of software involves community effort and this community generally grows for open source software. This law is not related to the software metrics.

E. *Law 5: Conservation of Familiarity*

According to this law, the familiarity with evolving E-type software systems is conserved. This allows conserving familiarity for developers and maintainers. For small changes the familiarity of software is easily achieved by the personnel involved with the software. Hence the average incremental growth remains constant as the software evolves. This law is not related to the software metrics.

F. *Law 6: Continuing Growth*

According to this law the functionality provided by the software should continually grow so as to provide user satisfaction over longer period. Growth can be interpreted as increase in the size of code or increase in the functionality being provided by the software. Growth in size can be determined by observing line of code (LOC) over subsequent releases. A similar approach was used by Lehman. We computed and compared LOC and number of classes for different releases of FREEMIND and JFREECHART.

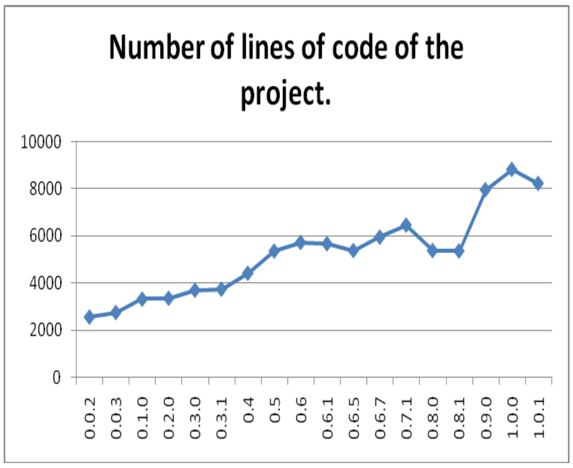

Figure 9. Number of lines of code for FREEMIND

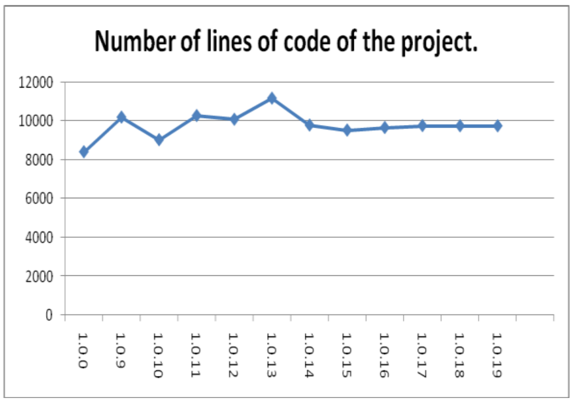

Figure 10. Number of lines of code for JFREECHART

G. *Law 7: Declining Quality*

According to this law, the quality of evolving software will decline unless some substantial efforts are made to improve it. The law is somewhat similar to Law 2. Increase in complexity itself is an indicator of declining quality. In case of Freemind and JFreeChart the law is clearly reflected in the line of law 2. Further decrease in the Maintainability Index (MI) value is an indicator of declining quality.

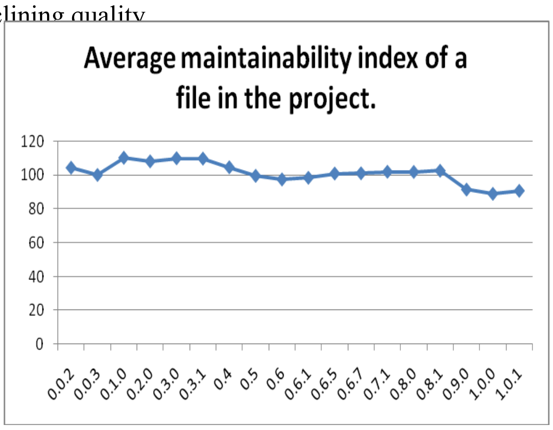

Figure 11. FREEMIND maintainability index

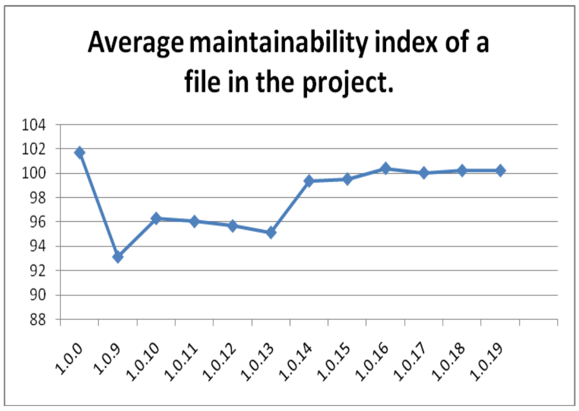

Figure 12. JFREECHART maintainability index

H. *Law 8: Feedback System*

According to this law, the evolution process depends on feedback system. For open source software the law seems to be true since feature request and reporting of bug comes from user community. The existence of feedback system is true for Freemind and JFreeChart, but this being multi-level involving multi-agent is hard to determine

VI. RELATED WORK

Some of the prior work that motivated us include the case study of HoDoKu and HSQLDB by Gagandeep Singh and Hardeep Singh [14] through the evolution study of 17 versions of HSQLDB (HyperSQLDataBase) and 10 versions of HoDoKu (a Sudoku generator/solver/analyzer).

The pioneer work in the field of software evolution was done by Belady and Lehman [15]. They conducted the study of 20 releases of OS/360 operating system. The study led them postulate the laws of software evolution. The laws were further developed and published by Lehman.

Another work that closely relates to our is the case study of JHot Draw and Rhino by Kalpana Johari and Arvinder Kaur[16]. Through the evolution study of 13 versions of JHot Draw and 16 versions of Rhino, their work highlighted the relatedness of Laws of software evolution.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, a study is performed on two open source software namely FREEMIND and JFREECHART. The applicability of Lehman's laws of software evolution to open source software was studied through number of object oriented metrics. We observed that the reflection of some of the laws namely law 1 (Continuing Change), law 2 (Increasing Complexity), law 6 (Continuing Growth) and law 7 (Declining Quality) have direct relevance to the computed metrics and were easily determined using the metrics. But the relatedness of law 3 (Self Regulation), law 4 (Conservation of Organizational Stability), law 5 (Conservation of Familiarity) and law 8 (Feedback System) to open source software system was hard to determine and will require more empirical studies with relevant data. The major contributions of this work are:

1. Considering the aspects of software evolution, the study has opened up new paths for process modeling and improvements.

2. The study helps in basic understanding of the behavior of open source software systems.

3. The validity of the study can be ascertained as the source is available on the internet

REFERENCES

[1] Lehman, M. M., 1980. "Lifecycles and the Laws of Software Evolution", Proceedings of the IEEE, Special Issue on Software Engineering, 19:1060-1076.

[2] Lehman, M. M., 1984. "Program Evolution", Journal of Information Processing Management, 19(1):19-36, 1984.

[3] Chidamber, S.R. and Kemerer, C. F. 1994. A metrics suite for object oriented design. *IEEE Transaction on. Software Engineering*. 20, 6, 476–493

[4] Oman, P., and Hagemeister, J., 1994. Construction and testing of polynomials predicting software maintainability. In Journal of Systems and Software, vol. 24(3), pp. 251-266

[5] McCabe, T.J., 1976. A Complexity Measure. In IEEE Transactions on Software Engineering, vol. 2(4), pp. 308-320

[6] Alkhatib, G., 1992. "The Maintenance Problem of Application Software: An Empirical Analysis", Journal of Software Maintenance – Research and Practice, 4(2):83-104.

[7] Artur, L. J., 1998. "Software Evolution: The Software Maintenance Challenge", John Wiley & Sons, New York, NY.

[8] FREEMIND.[Online].Available:http://sourceforge.net/projects/freemind/

[9] JFREECHART.[Online].Available:http://sourceforge.net/projects/jfreechart/

[10] FREEMIND.[Online].Available:http://sourceforge.net/projects/freemind/FREEMIND revision

[11] JFREECHART.[Online].Available:http://sourceforge.net/projects/jfreechart/JFREECHART revision

[12] ~~http://www.codeswat.com~~

[13] Gyimothy,T., Ferenc, R. and Siket. I/. 2005. Empirical validation of object oriented metrics on open source software for fault prediction. IEEE transaction on Software Engineering. 31, 897-910.

[14] Gagandeep Singh, Hardeep Singh, 2013. Effect of software evolution on metrics and applicability of Lehman's law of software evolution. ACM SIGSOFT Software Engineering Notes 38(1): 1-7

[15] Belady, L.A. and Lehman, M.M. 1976. A model of large program development. IBM Syst. J. 15, 225-252.

[16] Johari. K., Kaur. A., 2011. Effect of software evolution on software metrics: an open source case study. ACM SIGSOFT Software Engineering Notes 36(5): 1-8