ISSN:0975-9646

Software Metrics Evaluation: An Open Source Case Study

Sandeep Kaur[#1], Navjot Kaur[#2]

[#]*Department of CSE*
GIMET Amritsar, Punjab

***Abstract*— It is essential for any software to evolve so as to be used for large time period. It is necessitate to evolve software in order to do changes like adaptive, corrective, preventive maintenance In this paper we are presenting the results of study conducted on different versions of an open source software i.e. JStock. We calculated Object Oriented Metrics and investigated the changes in the measured values over different versions software which is developed in Java. Moreover we examined the applicability of Lehman's Law of Software Evolution on chosen software using different measures plus statistical analysis of the chosen metrics. We originate that Lehman's laws associated with increasing complexity and continuous growth are buoyed by the statistics and calculated metrics measure.**

***Keywords*— Laws of software evolution, software metrics, software complexity, open source**

I. INTRODUCTION

Software is not liable to wear and tear nevertheless it could turn out to be unusable if it is not swotted in retort to always changing user's needs. It is necessary to evolve software so that it can be used for a longer period of time. Lehman et al has made wide exploration on the evolution of big and extensive survived software [1].Basically Lehman's laws of software evolution specify that incessant change and development is must for any kind of software to be long-lived. The laws also represents that software becomes more and more complex plus it become more difficult to add new features to it over the time period i.e. due to changes and growth. No doubt there are more than decades ever since the laws being projected however there are very less experiential studies that provision the validating of laws to software systems. The free availability of source code of an open source software is delivering a support towards the study of software evolution.

Basically the work offered in this paper is based on the study of multiple versions of open source software i.e. JStock. The software used for study have been written in Java. The main aim of the paper is to observe the applicability of Lehman's laws towards object oriented software system. We have calculated the object oriented metrics, proposed by Chidamber and Kemerer[2] , for the software. The calculated metrics statistics for different versions has been used as the root for investigating the laws of software evolution. Another reason of the proposed study is to categorise structural metrics which could facilitate to discrete more than one aspirant version of a software system when shaping the contents of an open source release. The paper is organized as follows.

Section 2 represents the background history of software evaluation followed by related work presented in section 3. Section 4 presents a concise overview of software metrics used in this study .Section 5 gives an introduction to open source software. Section 6 presents the study of software evaluation over different versions of software component. Section 6 represents the analysis of statistical description for various metrics and the last section presents conclusions and future scope.

II. BACKGRUOND & HISTORY

A. Lehman's law of Software Evolution [1]

- Continuing Change (1974): The E-type systems ought to be unceasingly changed so as to be used for a longer period of time. The essential changes might be entitled for in response to change in environment, as a bug fix exercise, preventive maintenance activity etc. leading to change.
- 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.
- Self-Regulation (1974): Evolution process of E-type system is self-regulatory. This means that growth rate of 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.
- Conservation of Organizational Stability (invariant work rate) (1980): Evolution process of software conserves the organizational stability. The work rate of an organization evolving a large software tends to remain constant. This means it is hard to change the staff who has been working on evolving software. The average global effective rate in evolving software tends to remain constant over product lifetime.
- 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.

- 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.
- Declining Quality (1996): The Evolution process causes decline in the quality of evolving software.

III. Related Work

This section describe the various work done in the field of software evaluation. The forerunner work in the field of software evolution has been done by Belady and Lehman[1]. 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 in [7][8].

Israeli [9] used 810 versions of the Linux kernel, released over a period of 14years, to characterize the system's evolution, using Lehman's laws of software evolution as a basis. They investigate different possible interpretations of these laws, as reflected by different metrics that can be used to quantify them.

Stephen Cook et al [10] proposed refinement of SPE which was expected to provide a more productive basis for developing testable hypotheses and models about possible differences in the evolution of E- and P-type systems than provided by the original scheme.

IV. Metrics Used In This Study

This section represents the metric used in this study for software evaluation and statistical description. The details of metrics that we investigated are the following:

1) *LOC:* Line of code is the size related metric. It is a count of total number of lines in the selected elements that contain characters except white space and comments. [4]

2) CBO: "Coupling between Object Classes" is the measure of all those classes with which the given class is coupled [2].

3) RFC: "Response for Class" metric for a class is the measure of number of methods that can be invoked in response to a message received by an object of the class [2].

4) NPM: "Number of public methods" of a class represents the count of total number of public methods in a class[2].

5) LCOM: "Lack of Cohesion in Methods" is the class level metric that count the set of methods in a class which is disjoint with respect to members of a class being accessed by them[2].

6) Classes: This metric symbolizes all the available classes in the selected project. Classes, are user defined data structure, consists of data members and methods that exemplify the deeds of class in object oriented programming. [3]

7) Class Size: It is the fraction of the number of lines of code divided by the total number of classes.[3]

8) Complexity: It is useful for discovering how complex the code is? As classify by McCabe (1976), this is a metric which relies on a graph hypothesis that represents the total number of linearly free path in a related graph [4]. Mc.Cabe describe complexity as

V(G)= e-n+2, Where V(G) is cyclomatic complexity of particular flow graph G., e= total number of edges in graph G and n= total number of nodes in graph G.

V. Case Study

In this paper an evaluation has been illustrated by using software JStock from open source repository http://sourceforge.net/projects/jstock/. Basically the JStock [5], an open source software, purely coded in java. This has been chosen due to high downloading rate. Multiple versions of the JStock software has been downloaded and after that the metrics are calculated and arranges in tabular form to further perform operations.

B. JStock- Free Stock Market Software[5]

JStock is free stock market software for many countries. It provides Stock watchlist, intraday stock price snapshot, Stock indicator editor, Stock indicator scanner and Portfolio management etc. Free SMS/email alert supported. We download multiple versions for our case study from repository http://sourceforge.net/projects/jstock/ and perform software evaluation and statistical analysis among different metrics of each version.

VI. Analysis Of Evaluation Of JStock

In this section we examine the Lehman's laws of software evolution with the help of several metrics which are computed for multiple versions of JStock. From the evaluation it is concluded that some of the laws have a direct relevance to the computed metrics whereas for some of the laws we did not find direct relevance to software metrics. Software is not liable to to wear and tear nevertheless it could turn out to be unusable if it is not swotted in retort to always changing user's needs. It's necessary to grow for any software in order to be used for large era Lehman et al has done extensive study on the development and evolution of vast and broad lived software [1].

Lehman's laws of software evolution states that incessant transformation and development is mandatory for maintaining the software long-lasting. According to Lehman's laws of software evolution the continuous change and growth is highly required for maintaining the software for large period of time. These laws also explains that as the time passes away, it become more complex for the addition of different functionalities to software component due to changes and growth. Hence it turn out to be more necessary to investigate whether the open source code permits speedy evolution or not.

The figure 1 shows the growth curve of different metrics for the different versions of JStock i.e. open source software. In the below graph it can be seen that software has grown up in size can be resolute by perceiving the variation of size metrics over consequent releases. Figure 1 represents the linear growth curve for each calculated metrics for different versions. Growth of evolving software can be calculated by observing variance in the size metric in

terms of continuing growth, RFC in terms of feedback, CBO, LCOM, classes, class size etc. as a functional metrics.

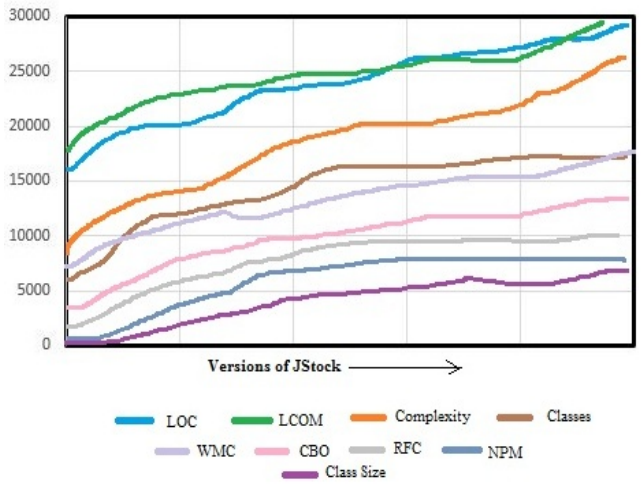

FIG 1:- GROWTH CURVE OF DIFFERENT METRICS FOR DIFFERENT VERSIONS OF JSTOCK

VII. MEASUREMENT AND CATEGORIZATION OF OPEN SOURCE CODE QUALITY

Software with high quality is always a foremost need of user. Always there is a trend of utilisation of software component in software application development because of its obvious benefit of low cost, great quality and lesser time for the development of applications etc. Hence it is necessity for the open source code to measure its quality. We selected to examine source code metrics because the code quality is resolute due to lot of elements & measuring it is far from trivial [6].We select these metrics as we can easily calculate these metrics by virtue of different metric tools.

The table 1 signifies the relationship between quality criterion and corresponding metrics. Table 1 clearly represents that the functionality the system can be calculated by metrics response for class, coupling between objects and number of public methods. Deployablity of the system can be checked by measuring class size and complexity metrics. The metrics lines of code and classed are used to measure the scalability characteristic of quality.

TABLE I
RELATIONSHIP BETWWEN QUALITY CRITERIA AND METRICS

Quality Criterion	**Associated Metric**
Functionality	RFC, CBO,NPM
Scalability	LOC,classes
Deployability	Complexity, class size

Table 2 presents descriptive statistics for software metrics calculated for an open source software. Descriptive Statistic is the discipline of quantitatively describing the main features of a collection of information or quantitative description itself.

In our research work, source code or quality metrics are calculated for huge no. of multiple versions of Jstock and arranged in tabular form. For the statistical analysis, each metric was considered as a random numerical variable.

Statistical analysis is a science of collecting, exploring and presenting large amount of data to discover underlying patterns and trends. Each module of an application has been measured and the mean value of each metric has been computed crosswise an application. Descriptive statics across all application is given in table 2.

For each metric, the minimum, maximum, mean, standard deviation and median values are calculated. In some belonging, awfully varying values have been discovered, but this is normal given the wealth of open source examined and the large number of peoples that have been concerned in the development of the software. For LOC the maximum value is extremely high. Standard deviation for lines of code and classes are also high and alternatively it leads to high value for response for class, NPM, coupling between objects etc. The achieved result for complexity are seemed to be generally far-away from the formal choice, but this is most likely be as an effect of poor classification and understanding of the impact of metric on a quality of code. The below table 2 give the complete representation of all the factors of statistical analysis i.e. minimum, maximum, mean, standard deviation and media for all the source code or quality metrics.

VIII. CONCLUSIONS

In this paper we offered the study centered on open source software. The applicability of Lehman's laws of software evolution towards open source software and statistical analysis were studied in the light of number of metrics.

Our aim was actually to explore the benefits that source code evaluation and statistical analysis of metrics can make available to open source and provide clues for further empirical research. This study has opened up more opportunities for research in the field of software evolution and quality measurement.

ACKNOWLEDGMENT

I would like to gratefully and sincerely thank to Ms Navjot Kaur for giving his valuable time for guidance in research work and support during the research period. Finally, I honestly thank to my parents, family, and friends, who provide the advice and financial support. The product of this research paper would not be possible without all of them.

TABLE III
STATISTICAL ANALYSIS OF CALCULATED METRICS

	Minimum	Maximum	Mean	Standard Deviation	Median
LOC	19537	27756	22948	2878.8	21327
RFC	417	458	444	13.56	446
CBO	71	76	74.4	1.58	74
NPM	16	20	19.11	1.36	20
Classes	172	246	202.3	28.601	183
Class Size	31.06	33.03	31.94	0.72	32.04
Complexity	1.96	50	30.32	16.44	26

REFERENCES

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

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

[3] Sanjay Kumar Dubey, Amit Sharma & Dr. Ajay Rana, "Comparison Study and Review on Object- Oriented Metrics", Global Journal of Computer Science and Technology, Volume 12 Issue 7 Version 1.0 April 2012

[4] Amit Sharma, Sanjay Kumar Dubey "Comparison of Software Quality Metrics for Object-Oriented System", IJCSMS International Journal of Computer Science & Management Studies, Special Issue of Vol. 12, June 2012

[5] http://sourceforge.net/projects/jstock/

[6] Ioannis Stamelos , Lefteris Angelis, Apostolos Oikonomou & Georgios L. Bleris "Code quality analysis in open source software development", Info Systems J (2002) 12, 43–60, published in Wiley Online Library

[7] Lehman, M.M. 1980. Programs, life cycles, and laws of software evolution. In Proceedings of the IEEE (Special issue of Software Engineering., 68,9, 1060 – 1076.

[8] Lehman, M.M. 1996. Laws of software evolution revisited. In Software Process Technology, ser. Lecture Notes in Computer Science. 1149, 108-124. http://dx.doi.org/10.1007/BFb0017737

[9] Israeli, A. and Feitelson, D.G. 2010. The linux kernel as a case study in software evolution. Journal of Systems and Software. 83, 3, 485 – 501.

[10] Cook, S., Harrison, R., Lehman, M.M. and Wernick, P. 2006. Evolution in software systems: foundations of the spe classification scheme. Journal of Software Maintenance and Evolution: Research and Practice.18, 1, 1-35.