A Review of Lehman's Laws in Open Source Software Systems

Taranjeet Kaur[1], Nisha Ratti[2], Parminder Kaur[3]
[1]Research Scholar, [2]Assistant Professor, [3]Assistant Professor
[1,2]Department of computer Science & Information Technology,Rayat Institute of Engineering And Information Technology, Railmajra, SBS Nagar, (Punjab) India
[3]Department of computer Science, Guru Nanak Dev University, Amritsar, (Punjab) India
[1]taranjeet.chd@gmail.com, [2]nisharatti@gmail.com, [3]parminderkaur@yahoo.com

***Abstract*— The characteristic of a software system to easily accommodate changes over the time is referred as software evolution. The study of open source software system is interesting because such systems are evolved in a strict development process .In this paper a survey of past studies related to Open Source Software (OSS) are presented. The objective of this study is to obtain an overview of existing studies related to OSS evolution and Lehman laws of software evolution.**

***Keywords*— Software evolution, software maintenance, open source software, Lehman laws of software evolution**

I. INTRODUCTION

Software evolution reflects "*a process of progressive change in the attributes of the evolving entity or that of one or more of its constituent elements*" [26]. Specifically, software evolution is related to how software systems evolve over time. Such systems are evolved after repeated modifications and results in increasing complexity Software is not prone to wear and tear but still it may become useless if not revised in response to ever changing user requirements. Software needs to evolve in order to be used for longer period. Lehman has done extensive research on evolution of large and long lived software. Lehman's laws of software evolution [22], based on the empirical study, indicate that continuous change and growth is required for keeping the software long-lived. The laws also suggest that over the period, due to changes and growth, software system becomes complex and it becomes more and more difficult to add new functionalities to it. The objective of this paper is to systematically select the published literature and provide the overview of OSS evolution. Section 2 differentiates between software evolution and software maintenance. Lehman's laws are discussed in section 3. Section 4 establishes the relation between OSS and Lehman's laws. Section 5 tries to discuss the reasons for not making any change in these laws.

II. SOFTWARE EVOLUTION VERSUS MAINTENANCE

Software maintenance is defined in IEEE standard 1219 [20] as the modification of the software product after the delivery to correct the faults, to improve the performance or other attributes, or to adapt the product to different modified environment. **Software maintenance** is different from the maintenance of physical goods because it normally leads to a changed or improved software system. Physical goods maintenance seeks to restore the item or entity as close as possible to factory condition. Software doesn't wear out but it "degrades" through suboptimal "quick fixes" and through an increasing number of unfulfilled requirements. Nontrivial software isn't free from defects: some maintenance is always required as long as the software is in use [11]. **Software evolution** is also different from the evolution of physical goods. It's mainly concerned with changes in a software system over versions or releases of the same system, while physical goods tend to evolve over "generations of products" (that is, there's little or no evolution of a specific physical product) [11]. Swanson [19] has categorized software maintenance into three categories: **corrective, adaptive,** and **perfective**. **Corrective** maintenance refers to the changes that fix bugs in the code base. **Adaptive** maintenance is changes that allow a system to run within a new technical infrastructure. **Perfective** maintenance are any other enhancements intended to make the system better, such as adding new features, boosting performance, or improving system documentation. According to this categorization, corrective and adaptive maintenance tasks do not alter the outward semantics of the system, while perfective maintenance includes a wide variety of possible changes to the system. Some later taxonomy adds a fourth category: Preventive maintenance is changes made to ease future maintenance and evolution of the system, such as reorganizing internal dependencies to improve cohesion and coupling. While the terms "evolution" and "maintenance" are often used interchangeably with respect to software, there are important semantic differences between them. Firstly, Evolution subsumes the idea of essential change that maintenance simply does not connote. Maintenance suggests preservation and fixing, whereas evolution suggests new designs evolving from old ones. Second, maintenance is usually considered to be a set of planned activities; one performs maintenance on a system. Evolution, on the other hand, concerns whatever happens to a system over time; in addition to the planned activities, unplanned phenomena often manifest themselves also.

III. LEHMAN LAWS

M.M.Lehman [21] has initially proposed three laws of evolution, introduced in the year 1974. Later five more laws of software evolution were added in the year 1980 and 1996 respectively [22, 23]. Before laws were defined, Lehman has also classified programs into three classes: S, P, and E. The law of software evolution refers to E-type softwares because

they change according to the environment. The laws of software evolution are:

A. *Continuing Change (1974):* E-type systems should continuously be changed in order to be used for longer period.
B. *Increasing Complexity (1974):* The complexity of an E-type system increases unless some preventive maintenance is done to control it.
C. *Self Regulation (1974):* Evolution process of E-type system is self regulatory. This means that growth rate of is regulated by the maintenance process.
D. *Conservation of Organizational Stability (1980):* Evolution process of software conserves the organizational stability. The work rate of an organization evolving a large software tends to remain constant.
E. *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.
F. *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.
G. *Declining Quality (1996):* The Evolution process causes decline in the quality of evolving software.
H. *Feedback System (1996):* E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.

IV. OPEN SOURCE SOFTWARE EVOLUTION AND LEHMAN LAWS

Lehman's well known eight laws of software evolution address the fundamental concepts underlying the dynamics of software evolution. Free software complies exceptionally well with Lehman's laws, although both the laws and the free software development process emerged independently: free software is continuously changed (law I), complexity increases noticeably (law II), and the self-regulation of the evolution process is obvious (law III). Many researchers have performed empirical studies to validate the Lehman laws on open source softwares by analyzing their different releases over their evolution period. Kalpana Johari and Arvinder Kaur [4] in their research paper has analyzed different versions of two open source software i.e. Jhotdraw and Rhino released through evolution process, by applying package level, class level and method level metrics, the main objective of the study was to examine the applicability of Lehman laws. Some laws have a direct relevance to computed metrics whereas for some of laws they did not find the direct relevance to software metrics used. They observed that the reflection of some laws namely law 1, law 2,and law 6 was easily determined but the relatedness of law 3, law 4 and law 5 to open source software system are hard to determine and will require more empirical studies with the relevant data. Chris J Arges [8] evaluate the Lehman laws for an open source software projects; Linux Kernel. Godfrey and Tu studied the Linux kernel evolution. Chris has given the view that Linux kernel showed a super-linear growth rate in lines of code and that the fourth Lehman's law did not apply. The fourth Lehman law states that development is constant and independent of the resources devoted to it. Then he has given the view that if open source E-type software violates Lehman's fourth law and compares its results with Godfrey and Tu's study [5] . The Linux kernel [24] showed super-linear growth as well as some other open source projects. However, many open source projects exhibited just linear growth and supported Lehman's fourth law.

V. WHY DID THE LAWS OF SOFTWARE EVOLUTION UNDERGO SO MANY MODIFICATIONS DURING THEIR EARLY LIFE, BUT HAVE NOT CHANGED IN THE LAST 15 YEARS?

The last modification of the laws of software evolution dates from 1996 [12], while the main works about the validity of the laws were published later. However Lehman did not change the laws because of the early empirical findings. He adapted the laws to the new development practices that evolved during the eighties and the nineties. He also took into account new concepts, such as feedback, to adapt the formulation of the laws [15]. To address the problem of different software development, maintenance and releasing practices, Lehman coined the SPE scheme, modifying the laws, and making it explicit that they are only valid for a particular type of software, i.e., *E*-type. SPE stands for the classification of programs given by Lehman i.e. S-type ,P-type, E-type software. S stands for Specification, P stands for problem solution and E stands for evolutionary type. Later on, he stated that P-type software are subset of E-type. So the classification is made limited to two types i.e. E-type and S-type Perry proposed the notion of domains in software evolution [16]. Evolutionary behavior could differ for software in different domains. The need for domains is very similar to the reasons that led Lehman to propose the SPE classification scheme: not all software evolves in the same way. As a matter of fact, the SPE scheme was recently updated [17] to clarify the differences between software evolution categories. The case of libre software is a paradigmatic example of a new domain that has been ignored in the definition of the laws and the SPE scheme. Prior to the appearance of the Internet, no software was developed by teams distributed across the globe, without regularly meeting in person, and communicating just through electronic channels. This environment is very different from the software engineering practices common during the times of the first versions of the laws. However, although Lehman updated the SPE classification scheme [17], he did not incorporate the notion of domain into the laws, or tried to adapt them to different software engineering practices. The laws kept exclusively referring to *E*-type software. Thus, as a summary, we can conclude that:

—During the first decades, the laws did not evolve as an answer to new empirical findings questioning their validity.

—The main modifications in the laws were due to changes in software development and maintenance practices and standards. The only source of modifications is publications from Lehman and collaborators. Proposals of modifications by other authors [14] were explicitly rejected [15].

—In modern publications by Lehman and collaborators, they have not taken into account recent studies about the validity for libre software. The most recent publication about the laws

[18] still contains the same formulation that was published 10 years before [12].

IV. Conclusions

In this paper, a study is performed on published mater and found that Some Lehman laws i.e. law 1,6,8,5 and 3, have a direct relevance to computed metrics whereas for some of laws i.e. law2,4,7, they did not find the direct relevance, will require more empirical studies with the relevant data. Software evolution research is still a young field and it continues to change its focus and even the underlying concepts. We know that software must evolve but still learning how to model this evolution, particularly in the increasingly complex environments in which the software is designed and deployed. There are many open questions in the road ahead including the validity of software evolution laws on the OSS, Which strategies, data and techniques can be used to validate the laws of software evolution? According to the reported results, which kind of software projects, and under which conditions, fulfill the laws of software evolution?

References

[1] Godfrey. M. W. and German, D. M. (2013), On the evolution of Lehman's Laws. J. Software. Evolution. and Proc.. doi: 10.1002/smr.1636

[2] Robles.G, Amor.J, Barahona.G.J and Herrariz.I, The evolution of the laws of software evolution. A discussion based on a systematic literature review.ACM Computing Surveys, Vol. 1, No. 1, Article 1, Publication date: June 2013.

[3] Lehman M.M., Laws of Software Evolution Revisited. In proceedings of he 5th European Workshop on the Software Process Technology, London, U K 1996

[4] Johari K and Kaur A, Effect of software evolution on software metrics: An open source case Study, ACM SIGSOFT Software Engineering Notes September 2011 Volume 36 Number 5.

[5] Godfrey W and Q. Tu. Evolution in open source software: A case study. In Software Maintenance, 2000. Proceedings. International Conference on, pages 131{142. IEEE, 2000.

[6] Xie.G, Chen.J, Neamtiu.I, Towards a Better Understanding of Software Evolution: An Empirical Study on Open Source, SoftwareProc. ICSM 2009, Edmonton, Canada

[7] Godfrey M.W.,German D.M. The Past, Present, and Future of Software Evolution Proc. ICSM 2008.

[8] Arges.C.J, Linux and Lehman- Literature Review of Open Source Evolution Analysis,

[9] Dang.Y and Mohsen.S, Does Firefox obey Lehman Laws of Software Evolution,

[10] Bennett.K.H and Rajlich.V.T, "Software Maintenance and Evolution: A Roadmap," Proc. Conf. Future of Software Eng., ACM Press, 2000, pp. 73–87; doi: 10.1145/336512.336534.

[11] Mens.T, Guehéneuc.Y.G, Ramil.J.F, D'Hondt.M, "Guest Editors' Introduction: Software Evolution," IEEE Software, vol. 27, no. 4, pp. 22-25, July-Aug. 2010, doi:10.1109/MS.2010.100

[12] Lehman MM, Perry DE and Turski WM, Why is it so Hard to Find Feedback Control in Software Processes? Inv. Pres., Proc. 19th Australasian Comp. Sc. Conf., Melbourne, Australia. Jan. 31 - Feb.2, 1996

[13] Lawrence.M.J 1982. An examination of evolution dynamics. In Proceedings of the International Conference on Software Engineering. IEEE Computer Society Press, Tokyo, Japan, 188–196.

[14] Pirzada.S.S 1988. A statistical examination of the evolution of the UNIX system. Ph.D. Dissertation.Imperial College. University of London.

[15] Lehman, Meir M. and Ramil, Juan F. (2003). Software evolution: background, theory and practice. Information Processing Letters, 88(1-2) pp. 33–44.

[16] Perry D.E., Staudenmayer N A ,People, organizations, and process improvement, , LG Votta Software, IEEE 11 (4), 36-45

[17] Cook.S, Ji.H and Harrison.R, Software Evolution and Software Evolvability, unpublished manuscript, University of Reading, UK, 2000.

[18] Lehman .M.M, Ramil.J.F: Software evolution - Background, theory, practice. Inf. Process. Lett. 88(1-2): 33-44 (2003)

[19] Lientz, B.P. and Swanson, E.B., Software Maintenance Management, A Study Of The Maintenance Of Computer Application Software In 487 Data Processing Organizations. Addison-Wesley, Reading MA, 1980. ISBN 0-201-04205-3

[20] id., Programs, Cities, Students, Limits to Growth?, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaug.l Lect. Ser., vol 9, 1970, 1974, pp. 211 - 229. Also in Programming Methodology, (D Gries ed.), Springer, Verlag, 1978, pp. 42 – 62

[21] id, On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle, J. of Sys. and Software, v. 1, n. 3, 1980, pp. 213 – 221

[22] Lehman M M, Perry D E and Turski W M, Why is it so hard to find Feedback Control in Software Processes? , Invited Talk, Proc. of the 19th Australasian Comp. Sc. Conf., Melbourne, Australia, 31 Jan - Feb 2 1996. pp. 107-115

[23] Israeli A. ,Feitelson D.G., The Linux kernel as a case study in software evolution, Journal of Systems and Software, Volume 83, Issue 3, March 2010, Pages 485-501.

[24] Cook S, Harrison R, Lehman M.M., Wernick P, "Evolution in software systems: foundations of the SPE classification scheme",Journal of Software Maintenance and Evolution: Research and Practice, Volume 18, Issue 1, January/February 2006, Pages: 1–35,

[25] Madhaji, N.H.,Fernandez-Ramil, J. Perry, D Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006.