A CRITIQUE BASE SOLUTION ON LEHMAN'S LAW

Khurram Shehzad[1], Maqbool Uddin Shaikh[1]
[1]Faculty of Computer Sciences, Preston University, Islamabad, Pakistan
Corresponding Author: arizon45@yahoo.com

ABSTRACT: Software evolution activities has cost and value for an organization. Unfortunately, software evolution processes are still treated in traditional ways though it varies ethnically and organization to organization. Lehman had devoted his lo ng life time to reform software processes. He presented most admirable work in the history of software evolution processes. Unfortunately his work is still not considered significant by software industry precisely in the Asian industry of software development. Further, Capability Maturity Models which appear after going crucial reforms are mainly related from management point of view. Here our scope is not a part of discussion on CMMI models, yet its existence cannot be ignored. In this paper, we have discussed in detail the history of software evolution processes and its reforms which have been reformed by Lehman in terms of Lehman's Law. Later a proposed solution is presented to overcome the limitations of Lehman's Laws for software evolution processes. These proposed reforms are done to overcome the limitations related to the development of software evolution processes. In this paper we are presenting a work flow model, which presents the dependency and solution against complexity issues are presented.

Keywords: Lehman's Law, Critique, software evolution, changing business model.

INTRODUCTIO

Software is a piece of programming instructions, which executes exactly what it is coded for. Software development practices are done under a software development life cycle (SDLC) model. There are ambiguities such as poor requirements, design flaws, trends like tools and technologies; and usage because of requirements changes are main causes of change in software. In view of Sommerville, the process of enhancing or modifying software is called process of software evolution [3]. "*Evolution is an essential property of real-world software*" [1]. Bennet P. Lientz in his study of the Maintenance Computer Applications Software expressed that "As needs change, criteria for satisfaction change" [2].

"*Software evolution is the term used in software engineering (specifically software maintenance) to refer to the process of developing software initially, then repeatedly updating it for various reasons*" [1]. E.B. Swanson initially identified three categories of maintenance which are corrective, adaptive, and perfective [2]. Preventive as a fourth category of maintenance which was introduced in 1980 [2]. When, Why, Who, How; such questions raised on change request. Sommerville pointed out some reasons for change request [3] which are the followings:

a) Incipient requisites appear during utilization of software.
b) The industry milieu revolutionizes.
c) Disappointment of programming either by equivocal requisite or by designer's oversights.
d) New gears integrated or required by the framework.
e) The framework obliges change in part of execution or unwavering quality.

In 1974, Lehman [5, pp 3-4] expressed that software engineers were getting to be progressively keen on surveying their benefit measures as far as every day source line of code (SLOC). He presented three laws related to software evolution process in light of his analysis or observations, which are listed as;

1. Law of continuing change
2. Law of increasing entropy
3. Law of statistically smooth growth

After having four years industrial software experience during 1974 - 1978 Lehman further introduced two more laws of software evolution that are listed as;

1. Law of continuing change
2. Law of increasing complexity
3. Law of statistically regular growth
4. Law of invariant work rate
5. Law of incremented growth limit

Lehman further, reformed software evolution laws in 1980 by applying a new Specified, Problem solving and Evolutionary(SPE) Scheme. Consequently different researches conducted research in different timing to validate these laws. Pirazada [5, pp 6, 8-10, 18-19 and 21] and other researchers [6], [12] concluded that the most Lehman Laws are not valid. If found it is validated then it would be a specific type of projects, like large project. Other short type projects don't prove a good implementation of Lehman Laws [7], [9]. During 1996 - 2006 a revised form of Lehman Laws [5, pp-11] were presented which are listed below:

L1: Law of continuing change.
L2: Law of increasing complexity.
L3: Law of self regulation.
L4: Law of conservation of organizational stability.
L5: Law of conservation of familiarity.
L6: Law of continuing growth.
L7: Law of declining quality.
L8: Law of feedback system.

According to Guowu [11] source code metrics and project information or defect information is used to analyze software growth, characteristics of software change(s) and software quality. Guowu [11] found in his study that Laws L1, L2, L3, and L6 are confirmed and in the remaining Laws L4, L5, L7, and L8 evidence to the contrary or precise operational definition is required. According to Dewayne E. Perry [4] dimensions of software evolution are Domains, Experience and Process. Further, the sub dimensions are feedback, experimentation, understanding, methods, organization and technology. These sub dimensions provide a wide variety of causes for evolution in software systems. These dimensions are interrelated in various ways and interact with each other in a number of surprising ways.

In Section I introduction is presented. In Section II critiques on Lehman's Law is discussed. In Section III proposed work is presented. In section IV proposed solution implementation workflow has discussed and finally in section V conclusion and future work is presented.

CRITIQUE(S) ON LEHMAN'S LAW

A. Software evolution, software evolve-ability and related Lehmans Laws were studied and discussed by several researchers [5], [6] and they pointed that:

L1: Continuing Change

"*An E-type system must be continually adapted, else it becomes progressively less satisfactory in use*" [5, pp. 11] and users concerns should always be consider accepting the change.

L8: Feedback System

"*E-type evolution processes are multi-level, multi-loop and multi-agent feedback systems."* [5, pp. 11].

During this study, we have argued that L8 orderly placed far away. L8 should have been the first to be stated. In software evolution, feedback comes first to get and go further. Feedback loops create continues development of a software product until user satisfaction. Sources of feedback can be stakeholders, the application domain, the environment in which the system deployed and the self-feedback in the form of logical errors. Developers must be keenly aware that if their software system does not respond positively then over time the system will be seen as increasingly less appealing by its user base. So new positions of L8 would be as L1 and old L1 should become L2.

L2: Increasing Complexity

"*As an E-type is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity*" [5, pp. 11] and

L7: Declining Quality

"*Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E- type system will appear to be declining*" [5, pp. 11].

The laws L2 and L7 are inversely proportional to each other because increased in complexity causes decrease in quality. This implies that the changes required for system evolution makes the system more complex and lower its quality [7].

L3: Self Regulation

"*Global E-type system evolution is feedback regulated.*" [5, pp. 11].

Over time any measurements of the system will follow a well-defined trend, which ripples effects the direction of business trends. System has to consider these regulations. This shows that

L4: Conservation of Organizational Stability

"*The work rate of an organization evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime*" [5, pp. 11]. The study of laws L3 and L4, finds that L4 is corollary of L3.

L5: Conservation of Familiarity

"*In general, the incremental growth i.e. growth rate trend of E-type systems is constrained by the need to maintain familiarity*" [5, pp. 11]

The users should be aware of changes in software so that the risk of losing understandability of software may reduce [8].

L6: Continuing Growth

"*The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.*" [5, pp. 11]

Since growth means adding new functionalities to software and change often means adding new code [7]. This concluded that L6 can be seen at least in part to be corollary of L1. Lehmans Laws have sometimes been criticized for lack of a solid empirical foundation [9].

B. Recent software evolution processes can be further categorized in following;

1. The emergence of software architecture

The rise of interior edges and subsystem limits by and large may cause unpredictability for a substantial programming framework [10]. Practically speaking, multifaceted nature computation of a substantial programming framework is judged by utilizing straightforward whole of the extent of the segments [10]. This methodology is misdirecting the unpredictability measures. Many-sided quality expanded by utilizing outer parts e.g. Commercial Off The Shelf (COTS). Case in point, gadget drivers now involve more than 60% of the source code in the Linux part source appropriation [10]. Since drivers speak with whatever remains of the framework by means of a moderately slender interface, so it is not clear that their inward multifaceted nature has much bearing on the many-sided quality of whatever remains of the framework and the other way around [10].

2. The de-monolithization of software systems

The current era of software development is a very different landscape [1]. Systems are embedded within an ecosphere of peer components for example libraries, frameworks, run-time environments, virtual machines and services; which in turn may be local, mobile, distributed, and location dependent [12]. Creating such a framework is not about composing source code. All it is going to comprehend accessible administrations, assessing security concerns, researching appropriated execution and pointing out organization points of interest [12]. A significant part of the complexities of current programming improvement don't appeared in conventional programming measurements [11]. One must assess conceivable parts, administrations, use and convey their frameworks inside a proper runtime environment [11]. Empirical models of the size, complexity, and development efforts of software systems need to explicitly recognize that not every important factor can be measured easily [11].

3. Freeware development and rapid (Agile) development methodologies

Current software development approaches such as agile approach, do not resemble the old-school model like waterfall [10]. Software development has new trends towards freeware or integrating customized COTS module to form a complete project whereas, significant resources contribute to implement core functionality for project(s) [8]. Industrial developers may also take existing open source codebases to create a customized suite to their particular needs and released to the community [12]. Developer's contributions and measurement of characteristics of product is still answerable [12].

4. Emerging Trends in Uses of Software

Lehmans Law L8: Feedback System perceived that product frameworks are installed inside different situations that can give criticism being developed of advanced programming variant [9]. Programming frameworks developed by changing necessities inside an advancing social and specialized environment [11]. Besides, when frameworks are created and conveyed, the clients from different hierarchical environment impacts future improvement patterns [10]. Lehman's verifiable exhortation, think as a framework build as opposed to a mathematician whose hypothesis can control truths to accomplish coveted objectives [12]. It is an actuality programming framework includes science in some sense. Its development is not only a question of controlling it until it performs a set of coveted usefulness [13]. The substances of programming improvement methodologies owe much to weights that lie outside of the formal advancement ancient rarities [14].

PROPOSED WORK

In our work some concerns about Lehman's Laws are presented, which may be considered as rehabilitation for software evolution. These concerns are:

1. Based on Software Development Life Cycle (SDLC) models, workflows and documentation [14] the Laws presented by Lehman are not in proper order as shown in Fig 1. In view of critiques discussed in section II, we have proposed a solution, which rearranges the order of flow, as shown in Fig 2.
2. Lehman did not present influence or dependency between software and its organizational environment [15]. Our given solution will resolve this concerned deficiency.
3. Software Evolution Process begins after deployment of first release of software product [2]. Users should verify their concerns and provide feedback. L1: Continuing Change law should not be first step to consider. L8: Feedback System should be the first step to consider as shown in Fig 2.
4. The Law L1: Continuing Change should be the second step in software evolution process as shown in Fig 2.
5. For Law L2: Increasing Complexity Lehman considers source line of code (SLOC) as a complexity measure. It is observed that SLOC cannot be a true measure of complexity [6]. The reason of this according to our experience and understanding the number of lines written for a software programming code to develop a function or a task can vary developer-to-developer, organization-to-organization, programming language to programming language and programming style. We believe that an architectural complexity or modular interaction can be a measure of complexity rather than SLOC. This is because SLOC depicts dependency or development of modules by application. Consider entities as nodes and relationship as edges it may be useful to measure complexity. Therefore, L2: Increasing Complexity should be at third position, which is shown in Fig 2.
6. In development and evolution of a software product, developers do unit testing for validation and end users do verification [8]. Due to this reason, L7: Declining Quality should be fourth to consider as shown in Fig 2.
7. After implementation and having quality check, software solution of organization reforms its ambiguities or flaws [6]. That is how software grows in terms of maturity as shown in Fig 2. Therefore Law L6: Continuing Growth should be at fifth position to consider.
8. Once software is developed, frequent changes may occur due to many other reasons. Advancement in software structure and functionality leads to standardization. Achieving standards is regulating software itself [7]. Therefore, Law L3: Self Regulating should be at sixth place as shown in Fig 2.
9. When software evolved due to the changes provided by the stakeholders, it became self-regulated under the growth of organization [8]. This applies new reforms in organization stability, which is about L4. Therefore, L4 should be at seventh position as shown in Fig 2.

10 Organization becomes synchronized with software, after evolving software according to user's requirements. End user of software product is able to provide feedback after familiarization of software [8], which is L5: Law of Conservation of Familiarity. Therefore, L5 should be at eighth position in software evolution process as shown in Fig 2.

Fig 1 below shows mapping of existing Lehman's Laws its evolution in environment. The reformed workflow based on software development lifecycle (SDLC) is shown in Fig 2, which shows mapping of revised ordered Lehman's Laws by us on the same software and its environment.

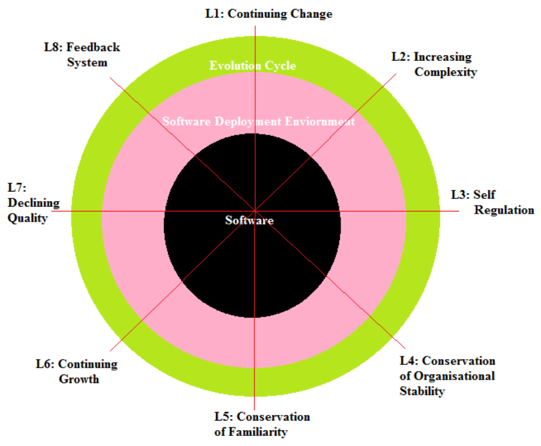
L1: Continuing Change
L8: Feedback System
Evolution Cycle
L2: Increasing Complexity
Software Deployment Enviornment
L7: Declining Quality
Software
L3: Self Regulation
L6: Continuing Growth
L4: Conservation of Organisational Stability
L5: Conservation of Familiarity

Fig 1: Basic Flow based on Lehman's Law

Table 1: Old and New formation Comparison Of Lehman's Laws

Ser #	Old Position	New Position	Reason
1	L1	L2	Changes comes with feed back
2	L2	L3	Complexity is after change
3	L3	L6	Till software grows
4	L4	L7	After self regulation software in stable state
5	L5	L8	Usage of software make them familiar
6	L6	L5	Growth comes with change
7	L7	L4	Increase in complexity gives low quality
8	L8	L1	Evolution comes after usage of software, usage enables users to provide feedback

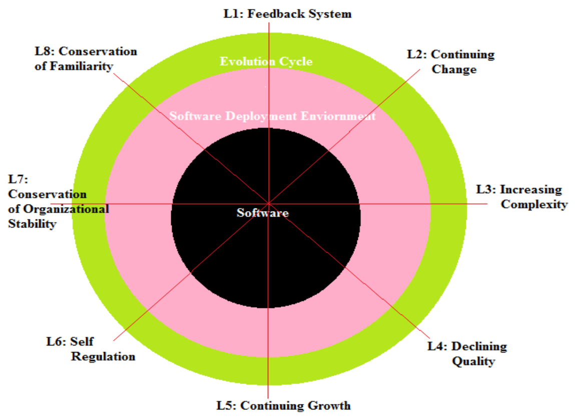
L1: Feedback System
L8: Conservation of Familiarity
Evolution Cycle
L2: Continuing Change
Software Deployment Enviornment
L7: Conservation of Organizational Stability
Software
L3: Increasing Complexity
L6: Self Regulation
L4: Declining Quality
L5: Continuing Growth

Fig 2: Reformed Flow

PROPOSED SOLUTION - IMPLEMENTATION WORKFLOW

To conduct software evolution process successfully it needs to have a 'workflow record'. The main reason is to carry out details of the evolution process and show the complete flow starting from feedback until familiarization of changes in software. This workflow should provide a solution and track to consider evolution process. Lehman in his research or industrial work did not present any workflow for the evolution laws.

Fig. 3 introduces the workflow or lifecycle model of our presented work. This workflow is based on SDLC. In traditional waterfall model [3 pp. 100], SDLC starts from requirement and leads to implementation. In modern agile practices [3 pp. 300] like requirement changes process are considered on different level of consideration. For this reason, we consider basic workflow of SDLC and consider it for advance practices

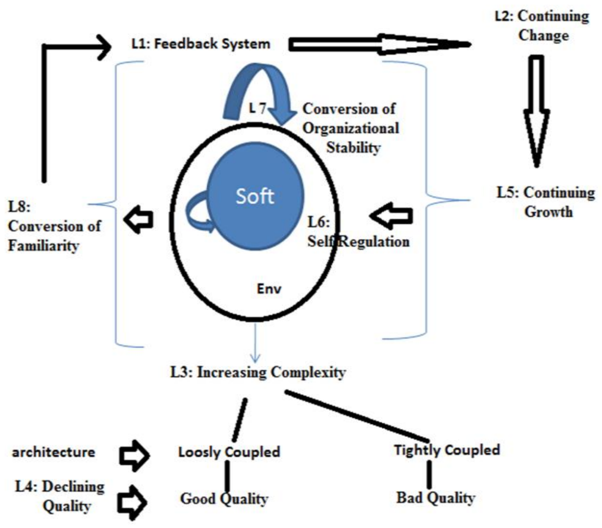

Fig 3: Software Evolution Architecture

The L6: Self-regulation of software is the process a enhancing of software product in itself by comparing services provided in different iterations. Software iteration process continues till the software complies with requirements and got stabilized. This type of software refers to stabilized business workflow that is known as L7: conservation of organizational stability. L3: increasing complexity of software is based on architecture. Strong coupling will raise L4: declining quality. To get good quality of software we have to consider loosely coupling and strong cohesiveness. Continuous working with software gives an ease of usage. It gets familiar with the product, which is L8: conservation of familiarity.

Familiarization of software product enables users to analyze and predict ambiguities. This familiarity enables users to provide a feedback, which is L1: feedback system.

Feedback based on ambiguities leads to fixation of bugs or fulfilling of changes. This process considered as in terms of software changes L2: continuing change. Continues changes reformed the software product into acceptance criteria until its growth accomplish the needs of users, which is L5: continuing growth.

Continuing development until user demands fulfilled, software regulates itself. This is L6: Self-Regulating law. This completes the life cycle of evolution process of software with in an environment.

CONCLUSION AND FUTURE WORK

In this paper, authors have presented different perspective of Lehman laws, activities and disciplines. Software evolution is not as mature as it should be. This work is a big contribution in enhancement of

software evolution by Lehman's Laws.
This mutiny in evolution process of Lehman Laws can be validated by the industry through surveys and actual implementation. Such implementation will lead to new trends in Software Evolution Process.

REFERENCES

[1] en.wikibooks.org.[Online]. http://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Deployment/Evolution, Oct 25, 2012 [25, Aug, 2014]

[2] Bennet.P.Lientz, Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations, 1st ed., E Burton Swanson, Ed. Boston, United States: Addison-Wesley Pub (Sd), August, 1980.

[3] Ian Sommerville, Software Engineering, 7th edition, Ed. United States, Boston: addison-wesley, 2004, ch. 21, pp. 488 - 511.

[4] Dewayne E Perry, "Dimensions of Software Evolution," in International Conference on Software Maintenance, Chicago, USA, April 2004, pp. 296–303.

[5] Israel Herraiz, Gregorio Robles, and Jesús M González-Barahona, "The evolution of the laws of software evolution. A discussion based on a systematic literature review," in ACM Computing Survey, United States, 2013, p. 28.

[6] Stephen Cook, He Ji, and Rachel Harrison, "Software Evolution and Software Evolvability," in University of Reading, UK, 2000.

[7] Robbes Romain and Michele Lanza,"Change- based software evolution," in ERCIM Workshop on Software Evolution, Lille, France, 2006, pp. 159–164.

[8] Steven P. Reiss, "Evolving Evolution," in Principles of Software Evolution, Eighth International Workshop on IEEE, Lisbon, Portugal, 2005, p. 4.

[9] Meir M Lehman, et. al., "Metrics and Laws of Software Evolution - The Nineties View," in Software Metrics Symposium, 1997. Proceedings. Fourth International IEEE, Albuquerque, NM, USA, 1997, pp. 20-32.

[10] Bennett H Keith and Václav T. Rajlich, "Software Maintenance and Evolution: a Roadmap," in Proceedings of the Conference on the Future of Software Engineering, ACM, Limerick, Ireland, 2000, pp. 73-87.

[11] Michael W Godfrey and M German Daniel, "On the Evolution of Lehman's Laws," Evolution and Process, vol. 25, no. 7, pp. 663-780, July 2013.

[12] Guowu Xie, Jianbo Chen, and Iulian Neamtiu, "Towards a Better Understanding of Software Evolution: An Empirical Study on Open Source Software," in IEEE International Conference Software Maintenance, ICSM 2009. Edmonton, Alberta, Canada, 2009, pp. 51-60.

[13] Wu Jingwei, et. al. "Exploring Software Evolution Using Spectrographs," in Reverse Engineering, 2004. Proceedings. 11th Working Conference on. IEEE, Delft, Netherlands, 2004, pp. 80-89.

[14] (2014,Oct.)sdlc.ws.[Online]. http://www.sdlc.ws/category/models/

[15] Goran Mustapic, et. al. "Influences between Software Architecture and its Environment in Industrial Systems – a Case Study," MRTC Technical Report, http://www.idt.mdh.se, 2004.