Abstract

The changes to software are inevitable as new requirements emerge with the use of the software. The change is necessary in terms of correcting errors, improving the performance and reliability of the system. The changes corresponding to corrective, preventive, adaptive and perfective maintenance leads to evolution of software. Software product is an evolving entity which starts with an initial implementation of a set of requirements and from then onwards software undergoes frequent modifications generally for the purpose of increasing its functionality, performance or in order to adapt to a changed environment. Software evolution refers to the phenomenon of software change and growth. The environment where software has to operate changes over time and software itself must adapt to this changing environment. The software does not change by itself but changed by the development and maintenance teams. In this study the evolution of Mozilla Firefox Open Source Software (OSS) was analysed empirically on latest fifty versions. The applicability of Lehman’s Law of software evolution was investigated on Mozilla Firefox using different software metrics. The Resource Standard Metrics (RSM) software metrics tool was used to get the values of the size and complexity measures of the software. It is evident from the results of the study that the evolution of the Mozilla Firefox for the latest fifty versions was in accordance with Lehman’s laws of continuing change, increasing complexity, continuing growth and declining quality.

1. Introduction

A software system undergoes many changes throughout its lifetime. Software may become useless if it is not changed in response to ever changing user requirements. Software needs to evolve in order to be used for longer period. The evolution of software systems can be generally thought of in terms of kind of changes that are made to the software. However, the overall motivation of evolution is the adaptation but the software changes are categorized as corrections, improvements and enhancements. Corrections are fixes made for coding errors but may also include design, architecture and requirements errors. Improvements can be increase in performance, usability and maintainability of any software system. Enhancements are new features added to the software system and that are generally visible to the users of the system. Most of the software systems experience software evolution over a period of time for adapting the changes. Lehman’s [1] first law of software evolution states that software that addresses the problems in the real world must be continually adapted and changed in order to offer satisfactory services. Functionality is added to improve the quality of the software. Efforts are made to maintain the software system and to keep basic structure as well as the overall architecture of the system intact. But the architecture also has to evolve to accommodate for unexpected events and to adapt to the changes. Lehman et al. concluded in their study that the software systems must evolve otherwise there is a risk of losing market share to competitors [2]. There are various evolution laws to study the evolution of the software system. The laws of software evolution refer to a series of laws formulated by Lehman and Belady [3] in 1974 with respect to software evolution. Lehman categorizes all software systems to three types: S-Types, P-Types and E-Types. E-type software refers to “software solving a problem or addressing an application in the real world” [4]. The laws also suggest that over the period, due to changes and growth, software system become complex and become more and more difficult to add new functionalities to it. The easy availability of source code and change details of open source software has contributed much to the study of software evolution. Lehman concluded that software evolution, which is the continual change of a program, was not a consequence of bad programming but it is required to keep the software up-to-date with the changing operational domain.
Continual software change is needed for the stakeholders’ satisfaction to remain at an acceptable level in a changing world. The various laws of evolution are tabulated in table 1:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Continuing Change (1974)</td>
<td>E-type systems must be continually adapted or they become progressively less satisfactory.</td>
</tr>
<tr>
<td>2.</td>
<td>Increasing Complexity (1974)</td>
<td>As an E-type system evolves, its complexity increases, unless work is done to maintain or reduce it.</td>
</tr>
<tr>
<td>5.</td>
<td>Conservation of Familiarity (1980)</td>
<td>As an E-type system evolves all developers, sales personnel, users associated with it, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive and abrupt growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.</td>
</tr>
<tr>
<td>6.</td>
<td>Continuing Growth(1980)</td>
<td>The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.</td>
</tr>
<tr>
<td>7.</td>
<td>Declining Quality(1991)</td>
<td>The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.</td>
</tr>
<tr>
<td>8.</td>
<td>Feedback System (1996)</td>
<td>E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.</td>
</tr>
</tbody>
</table>

2. Literature Review

Different researchers have different interpretation of the term software evolution. Belady et al. [5] describes software evolution as “the dynamic behaviour of programming systems as they are maintained and enhanced over their life times”. Bennett [6] states “software evolution is concerned with modifying software once it is delivered to a customer”. Arthur [7] differentiate between software maintenance and software evolution, and states that “software maintenance means to preserve from failure or decline whereas software evolution means a continuous change from a lesser or worse state to a higher or better state.”

Cook et al. studied the evolution of a real time telephone switch software system of a German telecommunications firm. They examined the evolution of the system over 18 months of data and argued that they found support for the laws of evolution, citing continuing change, increasing entropy and total change not being uniform [8]. Perry [9] concluded that corrections, improvements and enhancements are the three dimensions of software evolution. These dimensions of evolution provide a wide variety of sources of evolution for software systems. All these dimensions are interrelated in various ways and interact with each other in a number of unanticipated ways as well. Not only do they provide direct sources of evolution, but indirect sources as well.

In the area of Open Source System (OSS) the first case study by Godfrey et al. [10] analysed the evolution of Linux. In their study they expected OSS systems to exhibit a growth rate similar to their industrial counterparts as having growth rate an inverse square function. They concluded that the
evidences from this study indicated that Linux is growing at a super-linear rate which contradicts the empirical studies of commercial systems.

Paulson et al. [11] analysed three open source projects and closed source systems. They have used a linear approximation and have not found any differences in growth behaviour between open and closed source software projects.

Robles et al. [12] concluded that the fourth law of software evolution, ‘conservation of organizational stability’ which implies constant incremental effort, possibly does not apply to the large OSS.

Ramil [13] et al. have suggested that the number of files in terms of sum of the different files added, modified and deleted between two subsequent releases can be taken as a rough indicator of the effort applied to evolve a system from one release to another release.

Barahona et al. in their study shown the results of the evolution of the stable releases of Debian over nine years. They analyzed and presented the evolution of the size in terms of lines of source code, number and size of their packages, the changed and unchanged packages, the use of programming languages, and the dependencies between packages [14].

3. Objective of the Study

The main objective of this study is to observe the evolution phenomenon in Mozilla Firefox OSS and empirical investigation of Lehman’s law. The computed metrics value for different releases of Mozilla Firefox has been used as the basis for examining the laws of software evolution.

4. Scope of the Study

In order to achieve the objectives of the study, the source code of the data source was required. The data source Mozilla Firefox used in the study is open source software therefore the availability of source code for fifty different versions were not difficult.

5. Research Methodology

In this study the evolution of Mozilla Firefox Open Source Software (OSS) will be analysed empirically on latest fifty versions. The Mozilla Firefox is an E-type system as it addresses an application in the real world. The evolution of the E-type system can be studied by using the Lehman’s Laws of evolution. The applicability of Lehman’s Law of software evolution will be investigated for Mozilla Firefox using different software metrics. The researcher computed the size and complexity related metrics using a software tool called RSM. The RSM tool was downloaded from the website http://msquaredtechnologies.com/m2rsm/. The computed metrics value for different releases has been used as the basis for examining the laws of software evolution in context to Mozilla Firefox.

6. Analysis

The Lehman’s laws of software evolution in terms of several software metrics related to size and complexity for latest fifty versions of Mozilla Firefox released were analysed in the following manner.

Law 1: Continuing Change

The first law postulates that a program must continually adapt to its environment otherwise it becomes progressively less useful. According to this law the evolving software has to adapt to the changing environment. The change in usage environment may include addition of some functions which will increase the size of software leading to growth.

Figure 1. Lines of Code Growth in Firefox Releases

Figure 1 and Figure 2 showed the system growth for different measures i.e. change in lines of Code (LOC) and change in number of functions. It is evident from the results in Figure 1 and Figure 2 that there is an increase in the LOC and number of functions. This growth in terms of LOC and number of functions is assumed to be a result of constant need for new functionality.

Figure 2. Growth of No. of Functions in Firefox Releases
Law 2: Increasing Complexity

The second law state that as software evolves, its complexity also increases, unless effort is made to prevent it. It is generally perceived that growth in the LOC adds to increase in complexity but if the appropriate changes are made then the evolution process might not show increase in complexity. It is evident from the Figure 3 that there is an increase in the complexity of the Firefox from 3.5.17 and its successive versions. But after the versions 5.0 there is gradual decline in the complexity of the successive versions which can be supported by the fact that in case of Mozilla Firefox efforts are being made to make the changes in such a manner that evolution process does not increase the complexity.

![Figure 3. Growth in Cyclomatic Complexity of Firefox Releases](image)

Law 3: Self Regulation

The law of self regulation suggests that the evolution of large software systems is a self-regulating process, i.e., the system will adjust its size throughout its lifetime leading to a steady trend. From Figure 1 it is observed that growth curve in terms of LOC shows an abrupt increase in size between version 3.5.19 to 3.6.14 and then again from version 3.6.23 to 4.0B5. In Figure 2 the same behaviour is observed when considering the number of functions as metric for system size. Therefore it may be concluded that the law of self-regulation is not confirmed for the evolution of Mozilla Firefox as the growth is not showing a steady trend.

Law 4: Conservation of Organizational Stability

In OSS projects developers commonly collaborate with one another for the development of the software and they are free to modify and redistribute the source code. In this development scenario a central organization in the name of core group or steering committee exists. It plays an important role in providing general development guidelines to the community of developers and users and nurturing collaboration across the community. However, such an organization does not control how an individual developer should modify the source code. Spontaneous collaboration activities between developers ensure the delivery of a system’s first release and then sustain the future evolution of the system. The development process in case of OSS involves community effort and the organizational structure is decentralized. In case of Mozilla Firefox which is an OSS it is evident that the evolution law of organizational stability is not followed.

Law 5: Conservation of Familiarity

This law postulates that the incremental growth or growth rate trend of E-type systems is constrained by the need to maintain familiarity. Both the developers and users need to maintain familiarity with the system as it evolves between the releases. However from the results in Figure 1 and Figure 2 it is observed that increase in LOC and number of functions from version 3.5.19 to 3.6.14 and then again from version 3.6.23 to 4.0B5 is abrupt hence the familiarity with the software is not conserved. So in case of Mozilla Firefox the law of conservation of familiarity is not followed.

Law 6: Continuing Growth

The law states that the functionality provided by the software should continually grow so as to provide user satisfaction over longer period. The increase in the functionality being provided by the software can be interpreted as increase in the size of code. To judge the growth size of the software the growth of the various size metrics namely LOC, number of functions over subsequent releases was evaluated. Growth of evolving software, in terms of functionality, can be measured by observing change in the LOC and change in the number of methods. It can be observed from the Figure 1 and Figure 2 there is a noticeable change in the size metrics of Mozilla Firefox versions. It is concluded that the evolution pattern of Mozilla Firefox is correlated to law of continuing growth.

Law 7: Declining Quality

According to the law, the quality of evolving software will decline unless some substantial efforts are made to improve it. Increase in complexity itself is an indicator of declining quality. In the Figure 3 it is clear that there is an increase in the complexity of Mozilla Firefox over the successive versions. So this increase in the complexity is an indicator of the declining quality.

Law 8: Feedback system

According to this law an evolutionary process must constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base. Agents can be corporate managers, process managers, product managers, software architects, software developers, software testers and users. The loops are the code reviews and the level is the parallel versions. For open
source software the law seems to be partially true in the sense that since feature request and reporting of bug comes from user community. However the multi-loop, multi-agent feedback systems in case of Mozilla Firefox are not true.

7. Conclusion and Future Scope

The paper presented the study of the software evolution for fifty latest versions of Mozilla Firefox. The applicability of Lehman’s laws of software evolution to OSS Mozilla Firefox was investigated empirically using the size and complexity related metrics. It is evident from the results that the Mozilla Firefox shows an evolution pattern in accordance with evolution laws namely law 1, law 2, law 6 and law 7. But the relatedness to law 3, law 4, law 5 and law 8 in case of Mozilla Firefox was not found. In future more empirical studies with more number of versions over a long period in term of number of years can be carried out.

References