DOES LEHMAN'S LAWS APPLY TO OPEN SOURCE SOFTWARE SYSTEMS? A CASE STUDY

Submitted By:

Furqan Shahid

Reg No 110-FAS/MSSE/F06

Supervisor:

Muhammad Usman

Asstt Professor, DCS&SE, IIUI

Department of Computer Science & Software Engineering

Faculty of Basic & Applied Sciences

INTERNATIONAL ISLAMIC UNIVERSITY

ISLAMABAD

International Islamic University
CENTRAL LIBRARY
Islamabad

Accession No. TH-8646

MS
004.62
FUD
1. Network protocols; Computer science
2. Open Systems Interconnection

DATA ENTERED
15/3/13

International Islamic University, Islamabad
Faculty of Basic & Applied Sciences
Department of Computer Science & Software Engineering

Dated: 16-01-2012

Final Approval

It is certified that we have read the thesis, entitled **"DOES LEHMAN'S LAWS APPLY TO OPEN SOURCE SOFTWARE SYSTEMS? A CASE STUDY"**, submitted by **Furqan Shahid Reg No. 110-FAS/MSSE/F06**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the international Islamic university, Islamabad for MS degree in Software Engineering.

Project Evaluation Committee

External Examiner:
Professor Dr Arshad Ali Shahid,
Head of Computer Science Department,
National University of Computer & Emerging Sciences (FAST-NU), Islamabad

Internal Examiner:
Syed Muhammad Saqlain,
Assistant Professor, Department of Computer Science & Software Engineering
International Islamic University, Islamabad

Supervisor:
Muhammad Usman,
Assistant Professor, Department of Computer Science & Software Engineering
International Islamic University, Islamabad

Declaration

I hereby declare and affirm that this thesis neither as a whole nor as part thereof has been copied out from any source. It is further declared that I have completed this thesis entirely on the basis of my personal effort, made under the sincere guidance of my supervisor. If any part of this report is proven to be copied out or found to be a reproduction of some other, I shall stand by the consequences. No portion of the work presented in this report has been submitted in support of an application for other degree or qualification of this or any other university or institute of learning.

Furqan Shahid

110-FAS/MSSE/F06

ABSTRACT

In this thesis I have presented results of my case studies conducted to test the Lehman laws of software evolution on open source software systems. The domain of open source applications on which I have tested these laws is Enterprise Resource Planning (ERP). I tested three of the eight laws on three different open source ERPs. The reason behind this work is that the laws of Lehman, which he postulated on the basis of his thirty years experience, have been found to be disproving on some open source applications, like Linux Kernel, BSD family Kernel, and Nethack etc. It generated certain questions for research community, like, do open source applications really disobey Lehman or these are just few cases which can be considered as exceptional ones? If open source applications really differ from Lehman then what is the reason behind this difference? This type of questions require more effort of research community in the form of study of more and more open source applications (belonging to different domains) and then analysis of the produced results. This thesis is basically a contribution in this work, which can help research community in reaching some conclusions regarding the above mentioned questions. I tested three laws of Lehman, first, fifth and the sixth. According to these laws, software systems continuously change, continuously grow, their growth rate declines, and their incremental growth (amount of growth made in a version) remains in-variant on average. According to my results, the trend of declining growth rate was disproved on two of the three ERPs. The other three trends, however, were proved by all three ERPs included in my study.

ACKNOWLEDGEMENT

All praises and immeasurable thanks to **Almighty Allah.** by Whom mercy, I have achieved this success. Not only this, but all successes of my life are a result of His mercy, Who is The Most Merciful. After Almighty Allah and our beloved prophet **Hazrat Muhammad (Peace Be Upon Him)**, those who deserve most for my thanks, are **my parents**. These are my parents, whose prayers, efforts and sacrifices made me to reach this position. After that I am thankful to **all my teachers**, who taught me at any stage of my life, as they all have played role to make me to reach this position. From my teachers, I want to say thanks especially to **Mr M. Usman, my research supervisor**, for his efforts, guidance and good wishes regarding this thesis. My acknowledgement has no right to be called "complete" until I say thanks to **my wife**, whose prayers, encouragement and cooperation remained with me at each step of my research and thesis. In the end, I want to dedicate this work to my late son, **M.Yahya Furqan**.

Furqan Shahid

110-FAS/MSSE/F06

TABLE OF CONTENTS

Acknowledgement ii

List of Tables v

Chapter 1: Introduction

1.1-Software Evolution 1

1.2-Software Evolution Study 2

1.3-Lehman Laws of Software Evolution 3

1.4-Case Studies for Lehman Laws 9

1.5-Open Source Software Applications; Challenge for Lehman Laws 15

1.6-My Study 16

Chapter 2: Literature Review

2.1- Growth Rate of Linux Kernel [Godfrey 2000] 19

2.2- Growth Rate of Linux and BSD Family Kernels [Robles 2005] 23

2.3- Growth Rate of Linux and FreeBSD [Izurieta 2006] 26

2.4- Growth Rate and Complexity of Nethack [Simmons 2006] 30

2.5- Growth Rate of ARLA [Capiluppi 2004] 35

2.6- Growth, Complexity and Quality of Seven OSS [Xie 2009] 40

2.7- Growth Rate of Thirteen OSS Applications [Herraiz 2006] 46

2.8- Growth and Change Trends of Four OSS [Ali 2009] 52

2.9- Growth, Complexity, and Quality of Linux [Feitelson 2009] 60

2.10- Growth, Change, and Complexity of Nagios [Bonkoski 2007] 68

2.11- Growth, Complexity, and Quality of Firefox [Dong 2008] 75

2.12- Growth and Change Trends of 8621 OSS [Koch 2005] 80

Summary Tables 85

Chapter 3: Research Methodology

3.1-Open Source ERPs Selection 88

3.2-Getting history (old versions) of my studied ERPs 89

3.3-Tools Used 91

3.4-Procedure & Commands 94

Chapter 4: Results

4.1- Measures used in my Study 105

4.2-Graphs/Plots used in my Study 106

4.3-My Observations 107

4.4-Comparison of my Results with Other Studies 123

Chapter 5: Conclusions and Future Work 129

References 131

LIST OF TABLES

Table 1 Summarized description of the studies: [Godfrey2000], [Robles et al 2005], [Izurieta and Bieman 2006] and [Simmons et al 2006]

Table 2 Summarized description of the studies: [Capiluppi et al 2004], [Xie et al 2009], [Herraiz et al 2006] and [Ali and Maqbool 2009]

Table 3 Summarized description of the studies: [Israeli and Feitelson 2009], [Bonkoski 2007], [Dong and Mohsen 2008] and [Koch 2005]

Table 4 Summarized results of my case studies

Table 5 Size measures for Openbravo revisions

Table 6 Size measures for Adempiere revisions

Table 7 Size measures for ApacheOFBiz revisions

CHAPTER 1: INTRODUCTION

1.1 Software Evolution:

Software evolution means the changes made to software after its first delivery/release. Software evolution is a fact that can't be denied [Xie et al 2009]. Evolution is necessary to keep software acceptable for its users. It is because that the software is a model of some part of real world. It is developed against the requirements of real world. And as soon as real world changes, its requirements also changes, and as a result, software begins to in-validate against those changed requirements. If we will not evolve software then it will be continually in-validate against the requirements and a stage will come when it will become totally useless for its users. So in order to keep software acceptable for its users, it must be continually synchronized with the requirements of real world [Leh 78, Leh 97b].

Another source of evolution is the correction of errors in software [Ali and Maqbool 2009]. It is not possible for developers to develop error-free software, first time. There are always certain errors which can only be detected after software is delivered to its users. Although software testing phase is there to control the errors, but testing can only reduce errors, it can not remove 100 percent errors. There would be some errors which will be diagnosed and reported by the users, when they will use software. So software will then be required to be evolved for removal of those errors.

When software is delivered to its users (first release) and users begin to use it, then they detect and report errors. As a result evolution process starts. Errors are corrected, those corrections sometimes generate more errors, and thus more evolution is required. Meanwhile some of the real world requirements change and to accommodate those changed requirements, some more evolution (usually in the form of additions) is required. When those additions are made to software, then some new errors produce which again requires evolution. In this way the evolution process continues, as long as the software is used by its users.

1.2 Software Evolution Study:

Like other disciplines of software engineering, the software evolution has also got attention of researchers and significant work about this discipline can be found in published literature. The study in this area is important in the sense that it can be hoped to be helpful in reducing the cost of evolution [Ali and Maqbool 2009]. Controlling the cost of evolution is a challenge for software development community, especially when it has been discovered that evolutionary cost can be much more than the actual development cost [Sommerville 2005]. It has been discovered in a study that the cost of evolution can be more than 90% of the total software cost [Xie et al 2009].

One of the main focuses of software evolution researchers is to determine the trends of software evolution, so that uncertainty in the characteristics of evolving software can be minimized. Identification of the evolutionary trends can help us in prediction of the future characteristics of our software application and hence we can take certain steps to ensure cost effective evolution of our application. The pioneer of the evolutionary-trends

related work was Meir Manny Lehman, who, on the basis of his thirty years experience, postulated certain trends of evolution. He described those trends in the form of laws. So those trends are known as "laws of software evolution". His effort is admitted to be the biggest participation in the discipline of software evolution. It is difficult to find a study in this area that is without the reference of Lehman laws. The next section describes those laws, along with some of the case studies, which were used as basis of these laws.

1.3 Lehman Laws of Software Evolution:

Lehman initially proposed his laws in 1974, at that time they were three in total. Later in 1980, he added three more laws. Then again in 1996, he added two more laws, resulting in total eight laws [Leh 97b]. These laws are:

1.3.1 Continuous Change (The First Law):

According to the first law, a software system will be continuously changed throughout its life. If this change is resisted, trying to make software stable then its users will begin to feel discomfort from it. They will begin to realize it, unable to satisfy their needs, and this dissatisfaction will gradually increase. This is because software solves a real world problem, in this sense it is a model of some part of real world, and hence, the real world changes continuously, so its model must also be changed to keep it align with its original [Leh78]. The change in software is required for many reasons, sometimes it is for correction of some fault, sometimes for addition of new functionality and sometimes for providing support to new hardware. Need for change doesn't mean that the software was poorly designed, but change is an essential part of software.

1.3.2 Increasing Complexity (The Second Law):

According to the second law, the complexity of a program/software system increases, as it evolves. When a software system is evolved, for addition of new functionality, for example, then the objective of management is not limited to the addition of new features just, but this objective is constrained by some other objectives too. These other objectives include, for example, the time in which change must be completed, the cost limitation for implementing change and the resource consumption limitation for the changed version etc. One of these objectives may be, and in fact should be, the limitation on the structural degradation made by this change. But unfortunately, this objective is rarely considered. It is because this objective has no apparent return. Its effects are not short term. It has long term effects. If it is continuously ignored, as observed in many cases, the program structure will be continuously degraded with each change, and at last, a stage will come when it will become impossible to change the program more. If effort is spend to minimize the structural degradation made by each change, then this situation may be avoided [Leh78], [Leh97a].

1.3.3 Self Regulation (The Third Law):

According to the third law, the software evolution process follows certain predetermined trends. These trends remain same regardless of the type of system/software, the type of organization, its management and the environment in which it is working. These trends are usually uncontrollable. This law can be considered as abstract of the other laws. Each of the other laws represents a trend of software evolution. First law, for example, states that there is a trend of continuous change in software applications. Similarly

second law states that there is trend of continuous increase in complexity of a software application as it evolves. Similarly other laws too describe a trend of evolution, in this way it is an abstract of the other laws.

This law was first formulated when the incremental growth of OS/360 was plotted against the RSNs (Release Sequence Numbers) [Leh78]. It was found that the average incremental growth remains almost same with small positive and negative ripples. The subsequent studies also validated this law like, Logica Plc Fastwire (FW) financial transaction system [Leh97b] and the other studies made under the FEAST project [Leh98a].

1.3.4 Conservation of Organization Stability (The Fourth Law):

According to the fourth law, the average amount of effort spent on each release remains same. It must be noted that effort doesn't mean the man-hours dedicated to the release, but it is a measure of changes introduced by a release. This trend was first observed in the study of OS/360 [Leh78], when, the number of modules handled by each release, were counted and plotted against RSN (Release Sequence Number). Number of modules handled is a sum of three measures, number of modules added, number of modules deleted and the number of modules changed [Lhe98b]. The graph line was observed to be forming regular cycles around the average line with small positive and negative ripples. A release having changes more than the average was found to be followed by a release having changes less than the average by nearly same amount, thus making the average amount of change, same. It was concluded that this behavior was due to the fact that organization wants stability and doesn't permit changes (in a release) more than a specific amount. So this law was named as "Conservation of Organization Stability".

This law was not particularly examined in the subsequent studies made by Lehman at al, however Turski suggested an Inverse Square Model for system growth on the basis of this observation, that effort remains same [Turski96]. This model of Turski was found to be, being validated by the real life applications, which can be considered another evidence of validity of this law.

1.3.5 Conservation of Familiarity (The Fifth Law):

This law was first formulated during the study of OS/360, when it was observed that each release was increased in size, by a fixed amount. If a release showed un-usual increment in size then that un-usual increment was adjusted by its subsequent releases by either zero increment or even decrease in size. It was concluded that this trend was due to the reason that users of a software system can absorb only a fixed amount of enhancements. Thus if a release introduces larger amount of enhancement, a negative feedback generates and as a result, the subsequent releases either don't enhance or decrease in size to balance the effect of that larger amount of enhancement [Leh78].

At that time, Lehman concluded that the amount of enhancement (incremental growth) remains same throughout life of a software system. But in later studies, he discovered that this amount varies with the passage of time. At first, he couldn't conclude that either this amount increases or decreases. Like in the study [Leh97b], he stated that amount of incremental growth may increase with time because of improvement in programming technology, or it may decrease because of increasing complexity of software or it may remain same because of balance between these two factors. But in his later studies, he reached the conclusion that this amount of incremental growth decreases with the passage of time [Leh98b], [Leh2001].

In this way this law is, basically combination of two things, one, that evolutionary trends don't permit un-usual incremental growth. If such growth occurs then it will be adjusted by subsequent releases. And second, that the amount of incremental growth decreases as the software gets older.

1.3.6 Continuous Growth (The Sixth Law):

According to the sixth law, software systems continuously grow throughout their life. At first glance, it appears that this law is same as first law, means, the law of continuous change. But in fact it is different from that. Growth is a type of change made to cover those features of real life system which were not included in the previous releases [Leh97a]. If we adapt software to accommodate changes in those features of real life, which are already covered by the software system, then this type of change will not be considered as growth. Growth is related to the enhancements in the software.

A software application is a model of some real world application. And because real world has an unbounded number of attribute, so the application, being a part of real world, too has an unbounded number of attributes. But we have to develop software for a bounded set of requirements within the constraints of budget and time limitations. So it is not possible to automate a real world application fully. Thus we can say that every software system is incomplete. The attributes/features of application which we exclude from requirements, causes users dissatisfaction. So sooner or later, users begin to realize the need to cover those attributes too. This results in an un-ended process of software growth. [Leh97a, Leh2001]

1.3.7 Declining Quality (The Seventh Law):

According to the seventh law, the quality of a software system decreases as it evolves. The main reason of decline in quality is the structural degradation of software with its evolution, as described in the second law. It is fact that every change in software degrades its structure. The dependencies among its components increase in an un-normalized way. Components interfaces become more complex. The overall architecture loses its integration. This structural degradation makes it more difficult to change the software. Thus developers feel more difficulty in making changes in that software, which means that the quality of software declines with the developer's point of view. Moreover, when software is structurally degraded then any change in it causes to produce many errors and faults. And hence its quality also decreases with the user's point of view [Leh2001].

1.3.8 Feedback Systems (The Eighth Law):

According to the eighth law, the software systems are feedback control systems, and their evolutionary characteristics are controlled by the user's feedback. This law can be considered as a summary of the other seven laws. Each of the other laws describes an evolutionary trend for software applications, which is usually not controllable. This law states that those trends are basically a result of user's feedback. This law was basically concluded from the existence of regular cycles in the "amount of changes" and "incremental growth" graphs. The positive ripples were considered to be a result of positive feedback whereas the negative ripples were considered to be a result of negative feedback.

1.4 Case Studies for Lehman Laws:

Lehman suggested the laws of evolution on the basis of his more than thirty years experience. He was pioneer of the software evolution studies. He, on the basis of his contribution in this field, is known as "father of software evolution". His laws are based upon a number of case studies. These case studies are spread over thirty year period, from 1971 to 2001. During this time, he also conducted a project FEAST (Feedback, Evolution and Software Technology), with industrial collaboration, to study the evolutionary trends of software systems. This project let him to study a large variety of medium to large sized software systems, belonging to different application domains, working in different organizational environments and having different evolutionary periods. This project was consisting of two phases which were identified as FEAST/1 and FEAST/2. The first phase, FEAST/1, commenced on October 1996 and terminated in September 1998. And the second phase, FEAST/2, commenced on April 1999 and terminated in March 2001. Lets look at few of those systems, studied by Lehman, for the formulation of his laws. I have particularly focused on those parts of case studies which are relevant to the first, fifth and the sixth laws. Because my study includes these three laws only.

1.4.1 OS/360 [Leh 97a, Leh 97b]:

This software system was one of those which were studied by Lehman in the very beginning of his work means during 1970. But Lehman included its results in his later

studies too, like, [Leh 97b, Leh 98a], for the sake of comparison of later results with the older ones. Here are given two plots of this system:

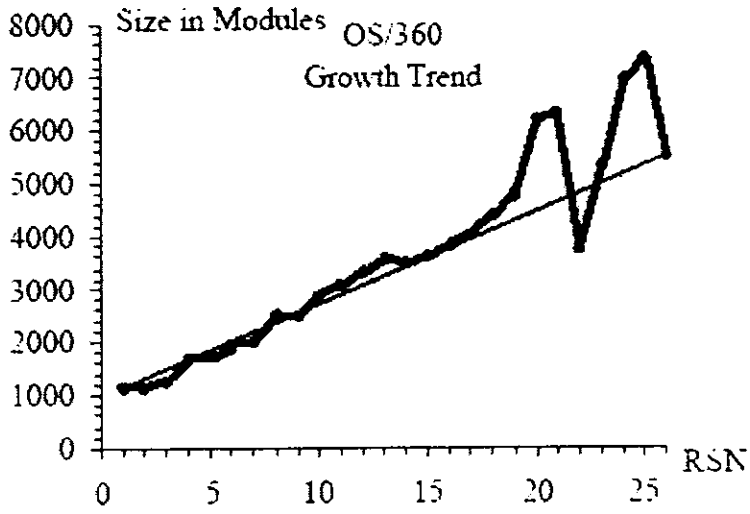


Figure 1: OS/360: Total size (in number of modules) plotted against releases [Leh et al 97 b]

In this figure, the total size of application (measured in number of modules) was plotted against the release sequence numbers (RSN). The upward movement of graph indicated continuous change and continuous growth trends. The linear increment in size has also been shown by the relatively thin line. It can be seen that actual increment was exactly following the linear increment but with small ripples (positive or negative). So it indicated that the incremental growth (amount of increment in each release) had remained same throughout the period. This linear trend made Lehman to conclude initially that the incremental growth remains same throughout the life of application. But, on the basis of later studies, Lehman changed this statement.

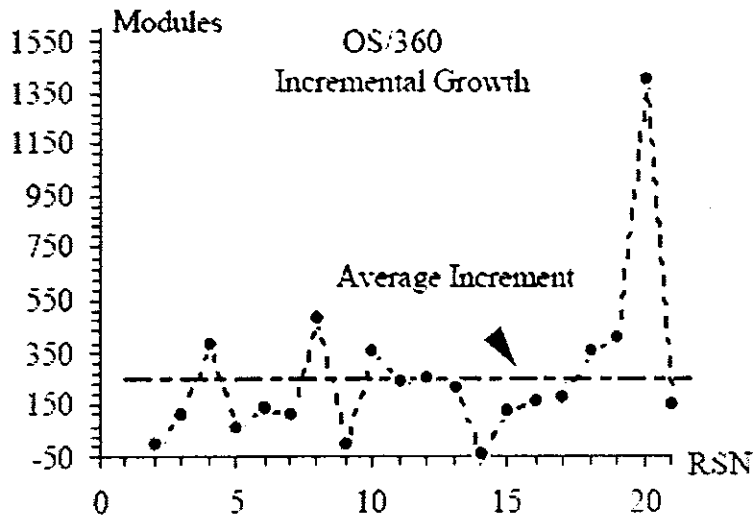


Figure 2: OS/360: Incremental growth (in number of modules) plotted against releases [Leh et al 97 b]

In this figure, the incremental growth (number of modules increased in each release) has been plotted against release sequence numbers. Here the straight horizontal line is showing average incremental growth. On the basis of this plot, Lehman concluded that size of each release increases by a fixed amount. And any un-usual increase in size will be followed by releases with zero or negative increment, to compensate that un-usual increase in size.

1.4.2 Logica FW (FastWire) [Leh 97b, Leh98a]:

This system was studied as part of FEAST project. It was a financial transaction system. At the time of study, it was eight years old. Lehman gathered data about its latest five years history. The data was consisting of number of modules in each release. Lehman preferred to use the measure, number of modules, in all of his case studies, unlike some other researchers, who preferred LOC. But Lehman said that number of modules can give more consistent picture as compared to LOC. Because LOC depends on the

programmer's practice whereas number of modules measure is independent from that of programmer. Moreover he said that he had repeated some of his case studies using the LOC measure and found the same results. Thus LOC and number of modules will act in the same way when used in the evolutionary studies of software systems.

In this study again, Lehman plotted total release size (number of modules) against the release sequence numbers (RSN) and found the same results as were found in the study of OS/360. This system was belonging to far different domain than that of OS/360. OS/360 was an operating system whereas this system (logica FW) was a banking transaction system. Moreover there was more than twenty years gap between both studies. So the repetition of results gave much confidence to Lehman regarding his conclusions, which he made more than twenty years ago, in the form of laws of evolution. The plot is shown by the given figure:

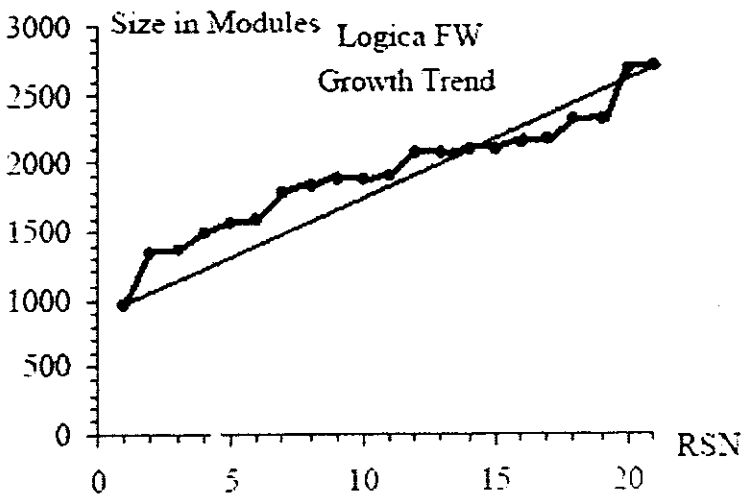


Figure 3: Logica FastWire: Total size (in number of modules) plotted against releases

[Leh et al 97 b]

The graph is showing continuous change and growth trends. However the linear increment in size (shown by the relatively thin line) has shown that the actual increment is not exactly following the linear increment. It is appearing that actual increment has slowed down with the passage of time. This observation made Lehman to change his statement of constant incremental growth throughout the life of application. He concluded that the amount of incremental growth varies with the passage of time, but he was still not sure that it will increase or decrease.

1.4.3 ICL VME Kernel and Lucent Technologies Real Time System [Leh 98a, Leh98b]:

These two systems were also studied as part of FEAST project. First one of these, Virtual Machine Environment (VME) was an operating system, developed by a UK based company, International Computers Limited (ICL). It was developed for the ICL's manufactured mainframe computers. The second system was a real time system developed by Lucent Technologies. Lehman gathered data about more than ten releases of each of these systems. Like his previous studies, he performed his analysis on the basis of "number of modules" measure. So he collected number of modules in each release and plotted them against the release sequence numbers (RSN). Both applications showed same type of results. The results for Lucent Technologies application are given here:

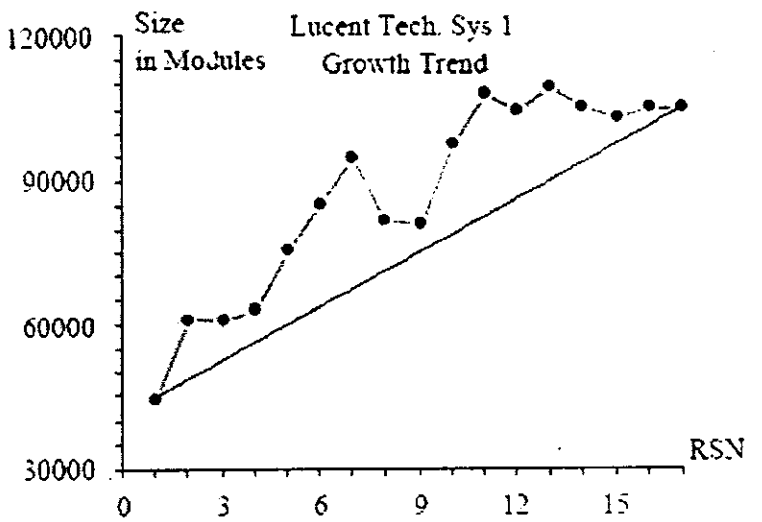


Figure 4: Lucent Tech rela time system: Total size (in number of modules) plotted against releases [Leh et al 98 b]

It can be seen that like OS/360 and Logica FW, the Lucent Technologies application also showed continuous growth (and hence continuous change) trend. The thin line, in the graph, is showing linear growth. And we can observe that the actual growth is lying totally above the linear growth. It is an indication of the fact that growth rate has been decreased with the passage of time. Thus the incremental growth (amount of growth in each release) had not been remained same. But it was decreased with time. This observation made Lehman to conclude that incremental growth (and hence growth rate) decreases as the system gets older. So he refined his fifth law, which initially stated that incremental growth remains in-variant throughout the application life. After refinement, Lehman's fifth law was changed as "the incremental growth and growth rate decreases with the passage of time".

1.5 Open Source Software Applications; Challenge for Lehman Laws:

Lehman formulated his laws on the basis of in-house applications. For all of his case studies, he selected commercial applications. He didn't test his laws on any open source application. His laws, however, were tested by other researchers for open source applications, and, some of those laws were found to be disproved in case of open source software applications. Once it was discovered that some of the Lehman laws don't hold true for open source applications, it gave a new direction to researchers. Many researchers were attracted towards this issue and they started to test Lehman laws on open source applications. The discoverer of this issue was Godfrey, who, first time discovered that Linux (a well known and commonly used open source operating system) wasn't following some of Lehman laws [Godfrey and Tu 2000]. He found that the growth rate of Linux had been increased during its past 96 releases. It was in contradiction with the fifth law of Lehman. Later, in 2005 and 2009, the evolution of Linux was again studied by two different researchers [Robles et al 2005], [Israeli and Feitelson 2009] and they also found same trends as were found by Godfrey in 2000. In both studies the growth rate of Linux was found to increasing, which was in accordance with the findings of Godfrey, and hence, was contradictory with the fifth law of Lehman. In 2005, Koch studied 8,621 open source applications [Koch 2005]. It was a very large set which included applications of nearly all varieties, small sized, medium sized, large sized, successful, unsuccessful etc. Moreover the applications were taken from different domains. Koch also found the fifth law of Lehman to be disproved, because many applications (especially large scale) were found to be evolving without any decrement in their growth rates.

In 2006, Simmons studied the evolutionary trends of an open source game, Nethack, and found that Nethack was not following second and fifth laws of Lehman [Simmons et al 2006]. He found that certain complexity measures for Nethack were decreased with the passage of time, which was in contradiction with the second law of Lehman. It was also found that Nethack's growth rate was not decreased during the past 23 years, which disproved the fifth law of Lehman. In another study, in the same year (2006), Herraiz observed 13 different open source applications [Herraiz et al 2006]. He found that only three of those thirteen applications were following the fifth law of Lehman. Means only three were there with decreasing growth rates. The remaining ten were evolving with either in-variance growth rate or with an increasing growth rate. Thus ten of thirteen applications were not obeying the fifth law of Lehman.

In 2009, Xie studied seven different open source applications [Xie et al 2009]. He studied large number of releases of all of those seven applications and observed their complexity and change rates. He found that all seven applications were increased in their complexity, which proved the second law of Lehman. However, he found fifth law of Lehman to be disproved, because none of the seven applications showed decrement in its growth rate.

1.6 My Study:

Validity of the Lehman laws on open source applications is still a question for researchers. It is nearly sure that some of the Lehman laws, in their original, don't hold for open source applications, and hence they need refinements. Although few studies have proposed refined versions of these laws [Dong and Mohsen 2008], but those haven't

been accepted at a large scale. And research community feels need of more case studies in this area, so that it can be clearly identified that, where and how much, the evolutionary trends of open source applications deviate from the laws of Lehman [Robles et al 2005], [Izurieta and Bieman 2006]. Such findings can be expected to prove helpful in the refinement of these laws. So I decided to conduct case studies to check validity of Lehman laws on open source Enterprise Resource Planning (ERP) systems. I have found no study in the published literature in which Lehman laws were tested on open source ERPs. I have decided to test first, fifth and the sixth laws of Lehman on three different open source ERP systems. The law which was first time found to be disproving on open source applications, was the fifth one [Godfrey and Tu 2000], and most (nearly all) of the later studies included this law in their work. So I particularly selected this law for my study. I also included the first and the sixth laws in my work because these two laws are very close to the fifth one, and same evolutionary measures can be used to test the all three. In order to strengthen my results, I decided to study three ERP systems rather than the only one.

CHAPTER 2: LITERATURE REVIEW

Introduction:

In this chapter, I have described the studies in which Lehman laws were tested on open source applications. Lehman proposed his laws on the basis of his case studies performed on commercial applications. He didn't even considered just one of the open source applications. And because open source applications differ from those of commercial ones in their very nature, so it is not un-natural to think that they (open source) may not be proving Lehman laws which are purely basing on closed source applications. This was the reason which caused researchers to check validity of these laws on open source applications. Here are given some of those studies. I have described each study under four headings:

Software applications:

Under this hiding, a brief introduction of those software application(s) has been given which were observed in that study.

Research methodology:

Under this heading, the methodology used by the study has been explained, which includes, what source was used to get applications (releases), what measures/metrics were calculated and what tools were used to perform the study.

Observations:

Under this heading, it has been stated that what type of evolutionary trends were observed in the studied application(s).

Conclusions:

Under this heading, the conclusions regarding the proof or disproof of Lehman laws have been described. That which law was found to be proved and which was found to be disproved.

At the end, a table, summarizing all these studies, has been given. In that table, each study is summarized using a framework consisting of ten elements, which are:

1. Software applications studied
2. Domain of applications
3. Type of applications
4. Evolutionary period covered/ number of versions observed
5. Measures/metrics used
6. Tools used
7. Source of releases used
8. What characteristics of the applications considered
9. Observations
10. Conclusions

2.1 Growth Rate of Linux Kernel [Godfrey and Tu 2000]:

Godfrey and Tu examined the evolution of Open Source Systems to observe that either their evolution holds the Lehman Laws of software evolution or not. Means either their evolution follows the trends of commercial in-house developed (closed source)

applications or not. For this purpose they observed two open source applications, the Linux operating system kernel and the VIM text editor. They presented their findings and conclusions about the Linux operating system in the study [Godfrey and Tu 2000].

2.1.1 Software Application:

Linux was originally written by Linux Torvalds, but then it were updated and enhanced by hundreds or thousands of developers. It was basically developed to run on Intel 386 platform, but later on it was ported to a large number of different architectures. Its first release was launched into market in March 1994. After that, a large number of versions have been released. Its releases are of two categories, development releases and the stable releases. The development releases contain the untested enhancements which are for experimental purpose. But the stable releases contain the bug fixes made to the experimental enhancements. The release numbers consist of three parts, middle of which is an indicator of the type of release, an odd number indicates a development whereas an even number indicates a stable release. At the time of study of Godfrey (early 2001), the total 369 development and 67 stable versions had been released.

2.1.2 Research Methodology:

Godfrey studied 96 releases in total, 62 of which were development whereas 34 were stable releases. The source of versions was the Linux Kernel Archives website. They examined, basically, the growth pattern of Linux by measuring the growth made in each release. For this purpose they used three different ways to examine the growth:

- The total size of the kernel, calculated as a compressed file
- The LOC of all source files was counted using the Unix command "wc-1"

- Using an awk script to make all blank lines and comments ignored, while calculating LOC
- Using a program "exuberant ctags", to count number of global functions, variables and the macros in different releases

2.1.3 Observations:

In order to observe the growth trends of the Linux, they plotted the size of different releases against time. Although different type of size measures were used as mentioned above, the total release size as a compressed file, the LOC of all source artifacts, the number of global functions, the number of variables and the number of macros. All of these measures were plotted, and all told the same story, as shown by the figures 5 and 6.

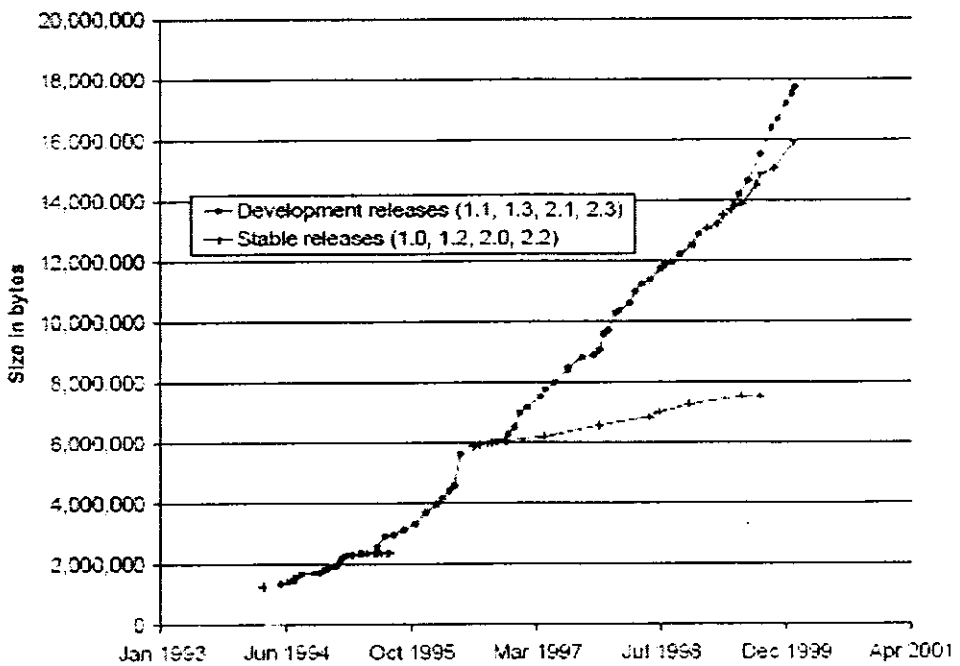


Figure 5: Total Size of Release (In Compressed Form) Plotted Against Time [Godfrey and Tu 2000]

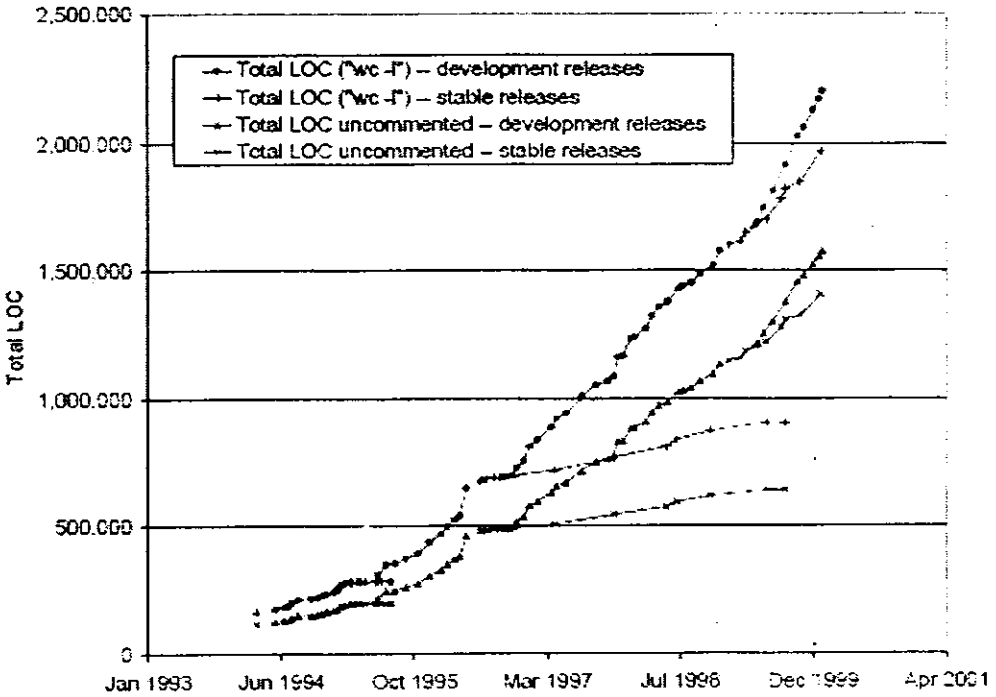


Figure 6: Total LOC of Release Plotted Against Time [Godfrey and Tu 2000]

It was observed that Linux has grown at a super-linear rate. Means its growth has speed up with the time. The growth rate has been increasing with the passage of time.

2.1.4 Conclusions:

Thus it has been clearly observed that Linux growth pattern contradicts with the Lehman laws. According to Lehman and his fellows, the growth rate of an evolving software application slows down because of increasing complexity [Leh97a] and [Turski96]. But the Linux growth rate has been increasing. Similarly it also contradicts another Lehman law according to which the work rate remains in-variant throughout the life of an evolving application. The super-linearity of Linux growth clearly indicates a continuous increase in the work rate. It can be summarized that the Lehman's Fourth and Fifth laws have been disproved in this study.

2.2 Growth Rate of Linux and BSD Family Kernels [Robles et al 2005]:

Robles et al studied the evolutionary trends of large libre software systems to determine either their evolutionary trends are same as those, observed by Lehman in the studies of commercial in-house applications, or different from them. Libre means free and open source applications. They studied applications having large sizes, being used by a large user community and participated by a large community of developers. The basic focus of study was to determine either these large open source applications follow Lehman's laws or not.

2.2.1 Software Applications:

Robles studied Linux kernel, BSD family kernels and eighteen other widely used open source software applications. The BSD family kernels are considered alternates to the Linux kernel. There are total three kernels included in this family, FreeBSD, NetBSD and OpenBSD. All these kernels have been derived from the UNIX operating system. The other eighteen applications all belong to different domains and have a reasonable evolutionary history.

2.2.2 Research Methodology:

They got different snapshots/versions of their applications from the publically available repositories on Internet. Linux however, being not available on public repositories, had been got from the Linux website. For all other applications including the BSD family kernels, the public CVS repositories were available, from where different snapshots of each application were got. The monthly snapshot of each application was downloaded from the date when repository was established till the time of study (April 2005). The

different versions of Linux were got from its website where these were available in the form of compressed files. The size of each release was measured using a tool "SLOC-Count". This tool detects all code files as well as the language in which those files were written and then calculates the source lines of code (SLOC) ignoring the comments and the blank lines. Robles et al studied 580 releases of Linux in total. For all other applications, four years or more evolutionary period was studied.

2.2.3 Observations:

In order to observe the growth rate of their studied applications, Robles plotted the total size of each version (measured in uncommented lines of code) against the time, taking time along x-axis and release size along y-axis. For Linux, the graph was super-linear, thus showing that Linux had grown at a super-linear rate (growth rate increasing with time) as shown by the figure 7. These results align very well with those found by Godfrey nearly five years ago [Godfrey and Tu 2000]. Besides the un-commented LOC, they used two other measures too, the total release size (in compressed form) and the number of files in release. Like LOC, these two measures were also plotted against time, and the same results (super-linear growth) were observed. In case of BSD family kernels, none of the kernel showed super-linear growth trend, instead they all showed a linear growth trend (in-variant growth rate) as shown by the figure 8. Only FreeBSD showed super-linear growth trend till 2000, but after that it also changed to linear one, resulting an overall linear trend. From the other eighteen applications, sixteen of them showed a growth rate linear or close to linear.

2.2.4 Conclusions:

Most of the studied applications clearly showed an in-variant growth rate. It clearly contradicts with the fourth Lehman law, according to which work rate remains in-variant throughout the life of a software application. If work rate remains in-variant then growth rate must decrease, as stated by Lehman and Turski [Leh97a], [Turski96]. According to the second law, the complexity of a software system continuously increases as it evolves. Increase in complexity means, more effort (work rate) will be required to maintain growth rate. But if effort remains same then growth rate will ultimately decline. So an invariant growth rate shows that effort (work rate) increases, due to which system has maintained its growth rate. It contradicts with the findings of Lehman.

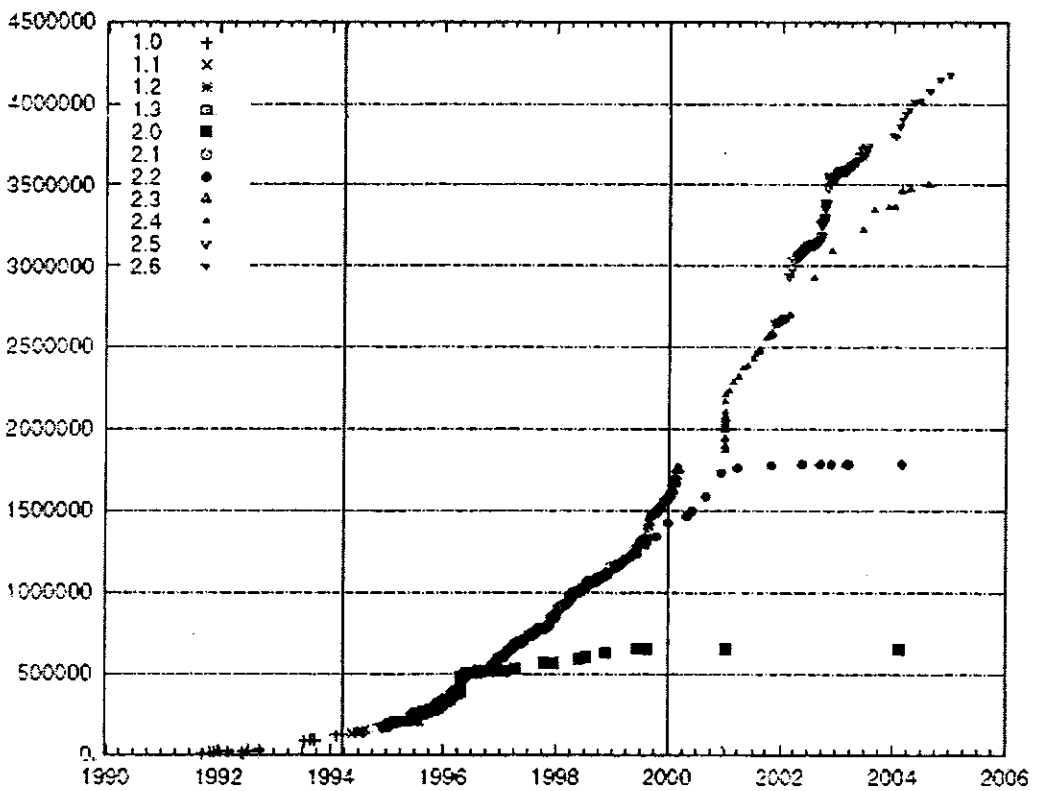


Figure 7: Total LOC of Linux releases plotted against time [Robles et al 2005]

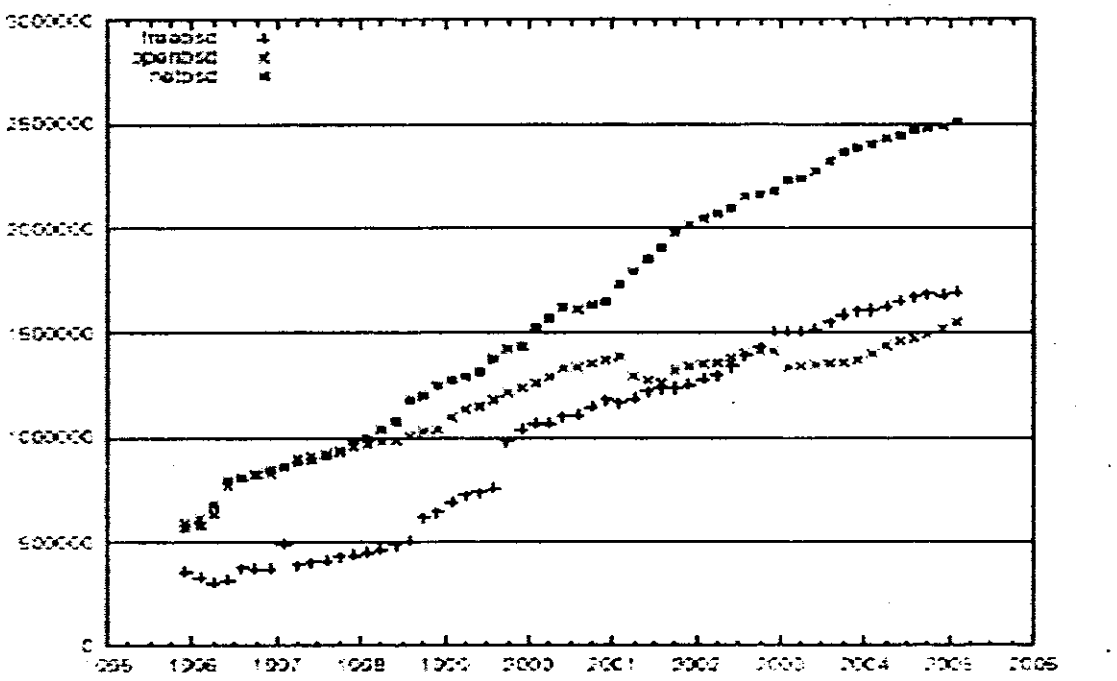


Figure 8: Total LOC of "BSD family Kernels" releases plotted against time [Robles et al 2005]

2.3 Growth Rate of Linux and FreeBSD [Izurieta and Bieman 2006]:

Izurieta and Bieman conducted their case studies to determine either open source systems evolve at a rate of commercial in-house systems or differ from that. They were basically exited for this work by the conclusions of Godfrey about the OSS evolution rate [Godfrey and Tu 2000]. They too were interested in the testing of Lehman's laws on open source systems, that either those laws hold for open source systems or not. Their basic focus was to examine that either open source systems grow at super-linear rate or not.

2.3.1 Software Applications:

Like Godfrey, Izurieta also studied a very popular and widely used open source operating system, i.e. Linux. However, they didn't think just one system sufficient, so they expanded their observation to another widely used and large size open source operating system, the FreeBSD. Both these operating systems are considered alternate for each others. However Linux supports more hardware devices than the FreeBSD. Both of these systems follow the following type of setup, in which a small team of core developers leads a large community of committers (developers) for development of new code. Later on, a committer may be included in the core development team.

2.3.2 Research Methodology:

They observed only stable releases of their studied applications. It is a common trend in open source applications to release a newer version with un-tested code for experimental purpose. That type of release, which contains experimental and un-tested code, is called a development (in case of Linux) or a current (in case of FreeBSD) release. However these types of releases are followed by stable releases which are mainly focused on bug fixes and correction of errors found in their predecessor development/current releases. Izurieta and Bieman did not include development or current releases in their study, but they purely focused on the stable releases. They got FreeBSD from its publically available CVS (Concurrent Versioning System) repositories. CVS is a system used to maintain code history. It maintains a history of changes made to source code files to enable us to retrieve previous versions of the code files. Linux, however, doesn't have such repository, so it was obtained from its web site. They observed 127 releases of Linux and 34 releases of FreeBSD, in total. They calculated different type of measures for their systems, including, Lines of Code (LOC), number of files, number of directories

and the total release size measured in Kbytes. To calculate these measures, they used the UNIX command "wc-l" as well as a shell script. Like Lehman, Izurieta plotted size measures against release numbers instead of time. It is important to remind that Godfrey plotted his measures against time rather than the release numbers. Thus in this aspect this study is following Lehman rather than Godfrey. As stated above, Izurieta got various versions of Linux from its website, where they were available in separate form. FreeBSD, however, was got from publically available CVS repository, so they downloaded the entire CVS tree of FreeBSD. Then by using CVS commands, they checked out different releases of FreeBSD. And then, at the end, they used UNIX command "wc-l" and the shell script to calculate different size measures, as indicated above.

2.3.3 Observations:

To observe the growth pattern of their studied applications, Izurieta and Bieman plotted different size measures of their applications against application's release numbers, taking release numbers along x-axis and size measures against y-axis. The different size measures which were plotted, include, total release size (measured in Kbytes), total LOC, total number of C files, total number of C++ files, total number of H (header) files, total number of Make file, total number of scripts and the total number of directories. From all these plots/graphs, they included just one in this study, that is, the total release size in Kbytes, which is shown by Figures 9 and 10. However they have said that all these plots showed same type of results. According to all these plots, both applications, means Linux and FreeBSD, have grown at a linear rate (in-variant growth rate). Although at

some points, a large increment in size could be observed, but the dominant growth trend is the linear one.

2.3.4 Conclusions:

According to Lehman, the software systems grow at a linear or sub-linear rate. Means either their growth rate remains same or it declines. If we examine growth rate on a short span of time, it will be constant on average, but, if we examine it on long time span, it will be decreasing [Leh98b]. Lehman quoted these findings under the fifth law of software evolution, i.e. "conservation of familiarity". Because both Linux and FreeBSD have grown at a linear rate, therefore they follow/prove the Lehman's fifth law of evolution. In this way, this study has proved that Lehman laws of evolution holds for open source applications too.

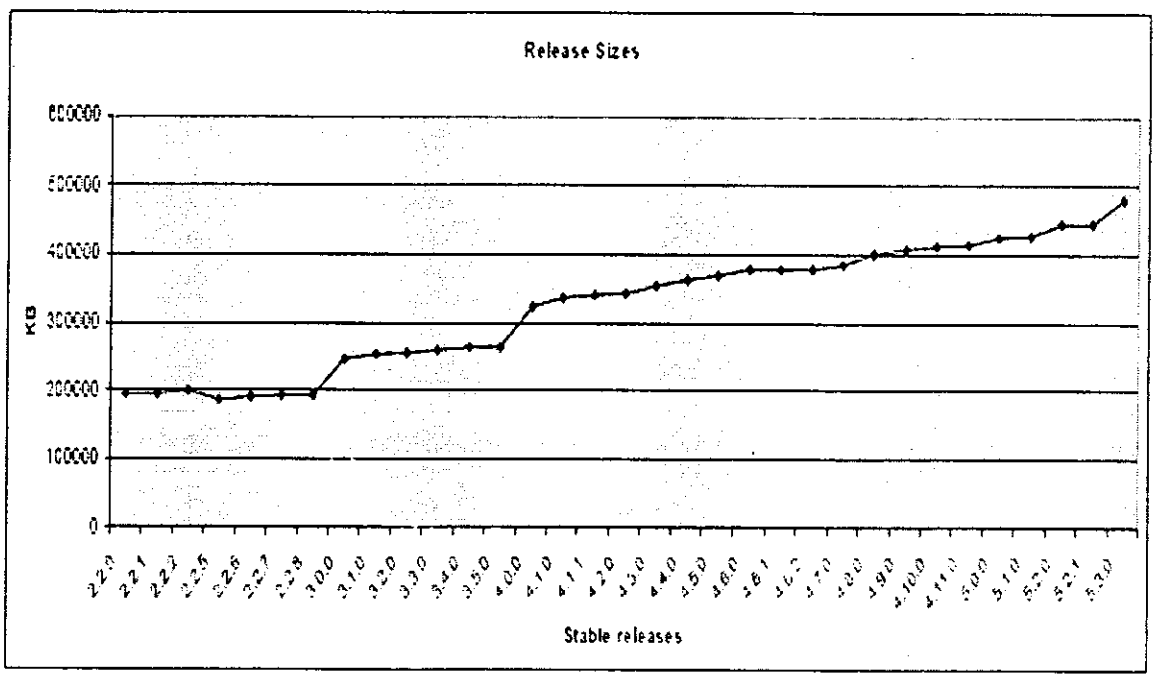


Figure 9: FreeBSD total release sizes plotted against release numbers [Izurieta and Bieman 2006]

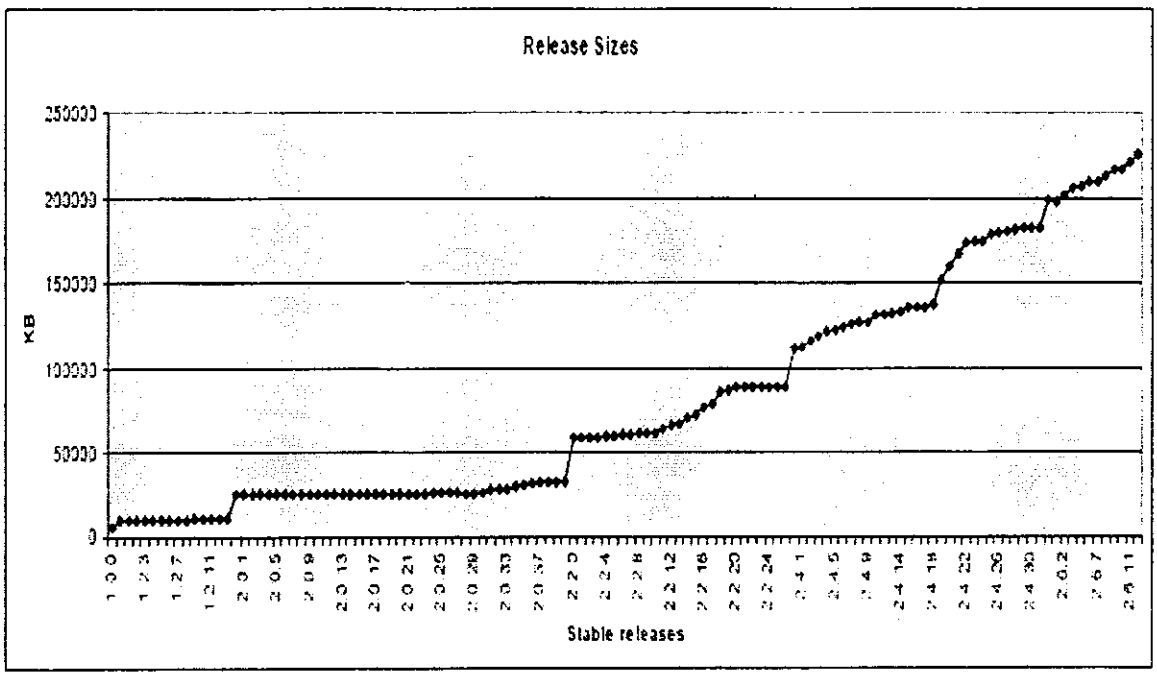


Figure 10: Linux total release sizes plotted against release numbers [Izurieta and Bieman 2006]

2.4 Growth Rate and Complexity of Nethack [Simmons et al 2006]:

Simmons et al performed software archeological study of evolutionary trends of an open source system. Software archeological studies are basically aimed at studying/observing characteristics of existing software applications to gain knowledge about different aspects of software systems. So Simmons and his fellows studied evolutionary history of an open source application to gain knowledge about evolutionary trends/characteristics of open source software systems. The evolutionary characteristics which were observed by Simmons include, the growth rate, the modularity, the complexity, the ratio of the module complexity to the module volatility for changes and finally the ratio of the comment lines to the LOC. They wanted to observe that either the measures related to growth rate,

modularity and complexity increase or decrease during the life of an open source application. Moreover, they observed that either the ratio of the comments increases as the LOC increases or not. Some of these measures are related to the Lehman's laws. Thus, in this way, Simmons tested some of the Lehman's laws on an open source application in this study.

2.4.1 Software Applications:

The open source software application, which was observed in this study, is a widely used large size open source game, the Nethack. This application, at the time of study, had an evolutionary history spread over more than a decade. Its popularity among user community can be realized by this fact that this game was downloaded more than 20,000 times during the month of December, 2005. Moreover, during this month, its link was hit more than 277,770 times. This application has been developed in C language. Its first version was released in the mid of 1980. And the latest version, at the time of study, was released in 2003. Its versions are categorized into three generations, early, middle and latest. During the early generation period, this application had no organized development team. The whole development was a result of contribution of individual developers spread all over the world. However in 1998, in the start of middle generation period, a team of core developers was organized. At the time of study, the strength of developers in the core development team was 11.

2.4.2 Research Methodology:

Simmons examined all the releases of this application (Nethack), starting from the earliest one (mid 1980) to the latest one (2003) with respect to the time of study.

However for calculation of different metrics they selected 12 such releases, which introduced major enhancements or which were aimed at major bug fixes. Simmons used the GQM (Goal Question Metric) approach for studying this application. There were total five goals of study:

1. Does the application's growth rate decline as suggested by Lehman?
2. Does the modularity of code increase, because of improving programming standards?
3. Does the complexity of code increase as suggested by Lehman?
4. Is the module having more complexity is also more volatile for change?
5. Does the ratio of the comment lines increase as the LOC increases?

It can be clearly seen that two of the goals (first and the third) are directly related to the Lehman's laws and hence they basically test Lehman's laws on the studied application. Simmons, then, decided a number of metrics to be calculated for evaluation of the above goals. The metrics, related to the first and third goals, include, the total release size, the LOC, the executable LOC (eLOC), the comment lines (cLOC), the number of functions, the McCabe complexity and the Halstead complexity. The applications source code was downloaded from the website sourceforge.net, where it was available in compressed form. In order to calculate the above mentioned metrics, a commercial tool "understand for C" was used. The whole metric data was then loaded to Microsoft Excel sheet for generation of graphs.

2.4.3 Observations:

To observe the growth rate of the application (Nethack), Simmons plotted the total release size, measured in bytes, against the release number. The result was a clear linear

growth as shown by figure 11. Linear growth means an in-variant growth rate. To strengthen the results, Simmons further plotted other growth related measures against the release numbers, which include, the total LOC, the executable LOC and the number of functions in a release. All these plots verified the above results, means all these showed a linear growth trend. These plots are shown by figures 12 and 13. As stated above, Simmons used two measures for observing complexity, the McCabe cyclomatic complexity measure and the Halstead complexity measure. The McCabe cyclomatic complexity measure counts the number of independent execution paths in the code, whereas the Halstead measure counts the number of operands and operators in the code. It is interesting to note that the application's (Nethack's) complexity was found to be decreasing with respect to the first measure, whereas it was found to be increasing with respect to the second measure. It means that the complexity of the application has decreased with respect to the number of execution paths, whereas it has increased with respect to the number of operators and operands.

2.4.4 Conclusions:

The evolutionary characteristics of Nethack do not obey Lehman's fifth and second laws. According to the findings of Lehman, the growth of a software application is sub-linear, if we observe it over a long time span [Leh97a] and [Turski96]. It means that growth rate of an application decreases /declines, if it were observed over a long span of time. Lehman has quoted this observation under the fifth law of software evolution. But in case of Nethack, the growth is not sub-linear, instead it is clearly linear, although it has been observed over a long time span (from mid 1980 to 2003). It means that the growth rate of Nethack is not declining but it remains in-variant. In this way it clearly

contradicts with the Lehman's fifth law. Similarly, Nethack's evolutionary trends contradict with Lehman's second law. According to the second law of Lehman, the complexity of a software application increases, as it evolves. But in case of Nethack, it was observed that the complexity of application, with respect to the number of execution paths, has decreased instead of increasing. It is clearly contradictory with the findings of Lehman.

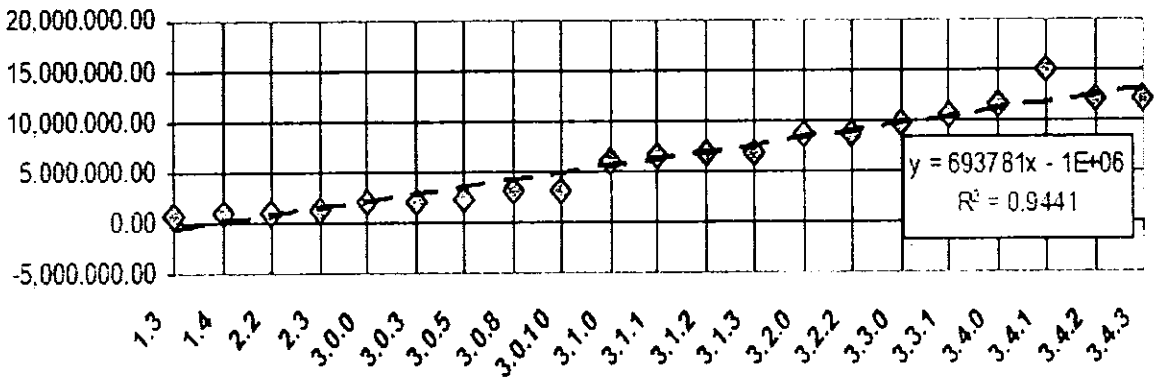


Figure 11: The total release size plotted against release numbers [Simmons et al 2006]

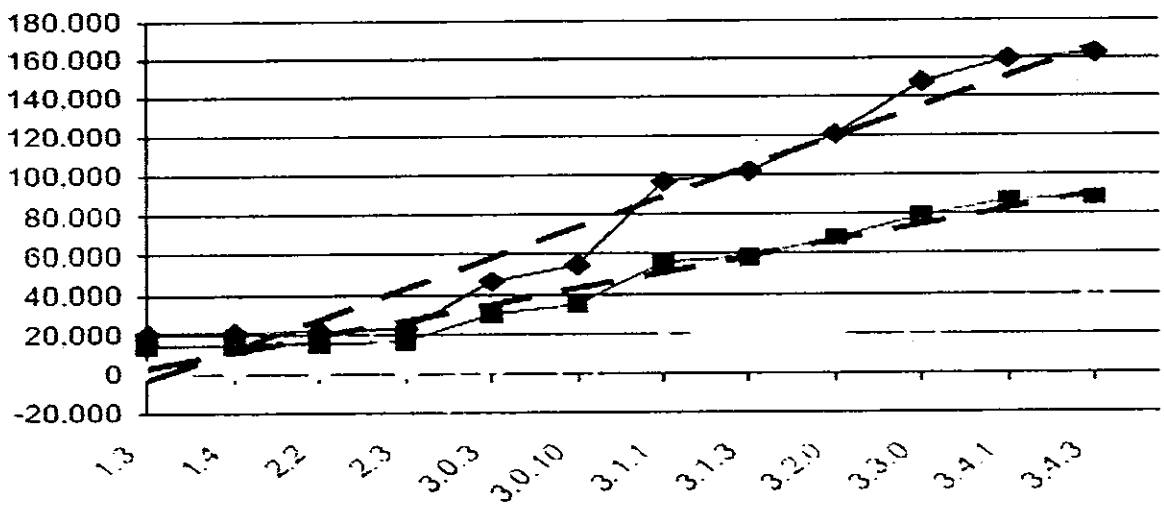


Figure 12: The total LOC (represented by black squares) plotted against release numbers [Simmons et al 2006]

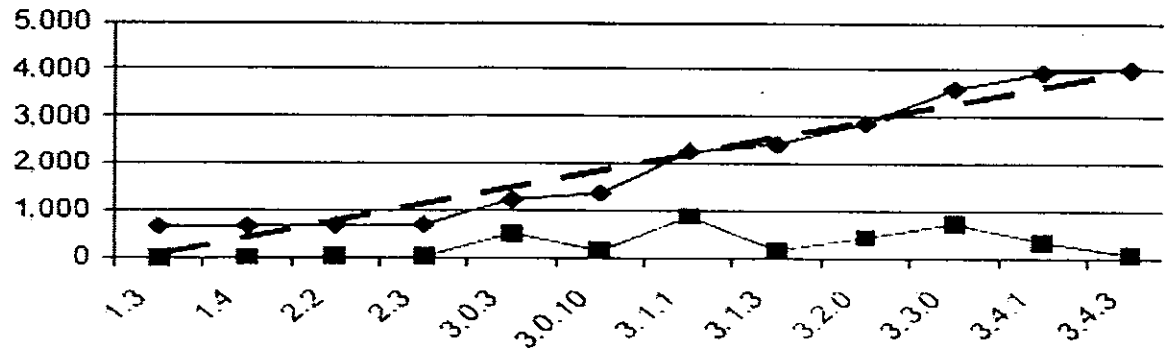


Figure 13: The total number of functions (represented by black squares) plotted against release numbers [Simmons et al 2006]

2.5 Growth Rate of ARLA [Capiluppi et al 2004]:

Capiluppi et al studied the growth of code components and changes in code structure of an open source distributed file system. They believed that these observations are helpful in the complexity evolution of an open source application. They basically have observed the effect of growth in code components on the structure of code as well on the arrival rate of new developers in an open source system. For this purpose they assumed three hypotheses and tested them against their system. Those hypotheses are:

Hypothesis 1: The code of an application (number of files) grows with the passage of time.

Hypothesis 2: There is a relationship between changes in code structure (folder structure) and the code growth.

Hypothesis 3: There is a relationship between arrival rate of new developers and the code growth.

They further explained each of the three hypotheses into the sub-hypotheses and then tested each of those sub-hypotheses on their studied application to find conclusions. First

one of the above three hypotheses is clearly related to the sixth law of Lehman. Not only related, but it can be said that it is exactly what Lehman's sixth law says. So it can be said that while testing this hypothesis, Capiluppi basically tested the sixth law of Lehman. Similarly, for testing this first hypothesis, they plotted different size measures like, LOC, SLOC and KB etc, of the studied application against the release numbers. These graphs, on one side proved the hypothesis, and on the other side showed the growth rate of the application, which is directly related to the fifth law of Lehman. Thus those graphs proved a source of testing the fifth law of Lehman.

2.5.1 Software Applications:

Capiluppi et al studied an open source distributed file system, ARLA. Distributed file systems can be said backbone of the internet technology. It can be used to enable users to access files/data stored/placed at different computer systems, at their own PC's. ARLA has been written in C language. Its first version was released in Feb 1998 and was labeled as "0.0 release". The latest version, at the time of study, was 0.35.12, which was released in Feb 2003. In this way Capiluppi covered an evolution period of ARLA spread over five years. During this five years period, 62 versions were released in total. 35 of them were major whereas 27 were minor releases.

2.5.2 Research Methodology:

As stated above, Capiluppi explained each of the three hypotheses in the form of many sub-hypotheses. The first hypothesis has been explained as following sub-hypotheses:

Sub-hypothesis 1: The LOC of application increases with releases.

Sub-hypothesis 2: The SLOC of application increase with releases.

Sub-hypothesis 3: The size of source files measured in KB increases with releases.

Sub-hypothesis 4: The number of source files of application increase with releases.

Sub-hypothesis 5: The number of source folders of application increase with releases.

They studied 62 versions of ARLA in total. These versions were released during a period of five years (1998 to 2003). They extracted data about different versions of the application from a database created by them, in their previous work [Capiluppi 2003]. That database contains data about 400 different open source applications. The source of information used in that work was the publically available CVS repositories of this application. In order to test these hypotheses, they used the measures, LOC (total lines of code including comments, blank lines etc), SLOC (source lines of code, LOC which are executable), KB (total size of all source files in kilo bytes), number of source files and the number of source folders. For LOC, they identified all source files (files having extension .c or .h) and counted number of lines of all of those files. For SLOC, they created a parsed file against each source file by eliminating comments and blank lines from it. The parsed files were containing only the executable code. So number of lines of parsed files gave the SLOC measure. To calculate these measures, they used UNIX commands, as well as the "xscc" awk script.

2.5.3 Observations:

In order to observe code growth, Capiluppi plotted LOC against releases. The graph clearly showed an upward trend as shown by Figure 14. Means the first sub-hypothesis was true, that the LOC of an application increases with releases. Moreover this graph showed a linear trend. Means the rate of growth was in-variant. In order to test the other sub-hypotheses, they plotted, one by one, each of these measures against releases, the

SLOC, the size of source files in kilo bytes, the number of source files and the number of source folders. All these graphs showed same type of results as were shown by the first plot, means an upward trend. The graphs for size (in KB) of source files, number of source files and number of source folders have been shown by figures 14 and 15 respectively. Thus all of the sub-hypotheses were found to be true. The application was found to be growing with respect to all size measures. Similarly, these graphs also showed a clear linear trend, except the last one, means number of source folders. Although it can be said linear to some extent, but it was not as clear as were the other ones. Thus all these graphs too showed an in-variant growth rate.

2.5.4 Conclusions:

ARLA has found to be following Lehman's fifth and sixth laws. According to the sixth law, an application continuously grows throughout its life. ARLA, when examined using a number of different size measures, was found to be growing. Thus it proved the Lehman's sixth law of evolution. Similarly, according to the fifth law, an application grows at an in-variant rate, if we observe its growth over a short time span. In this study, ARLA's growth was observed over a five years period and it was found to be growing at an in-variant rate, which is in accordance with the law.

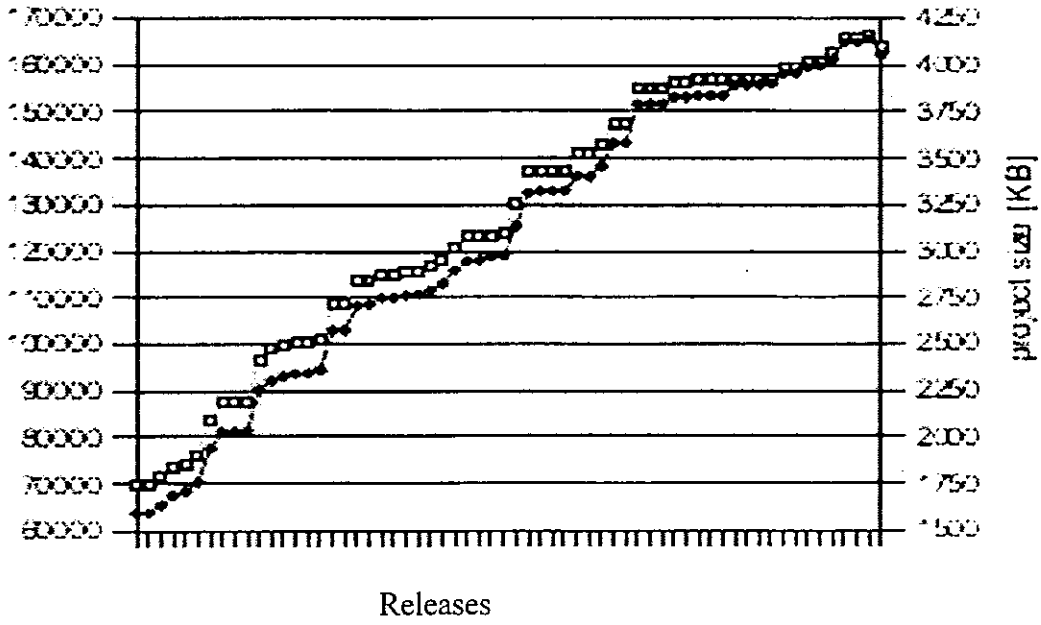


Figure 14: The LOC and total size of source files (in KB) plotted against releases [Capiluppi et al 2004]

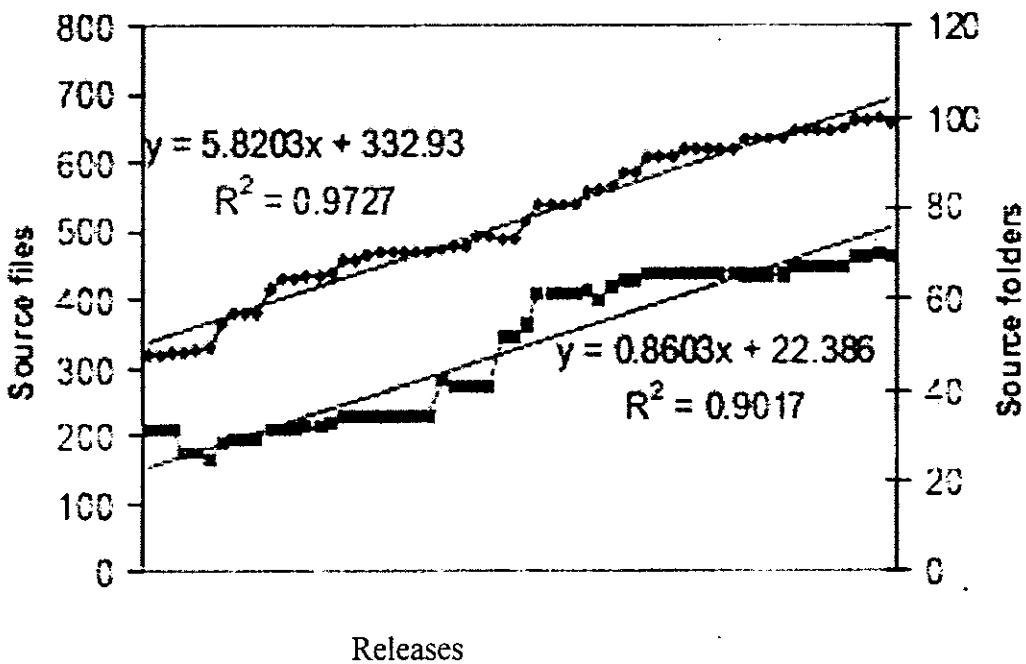


Figure 15: The total number of source files and total number of source folders plotted against releases [Capiluppi et al 2004]

2.6 Growth, Complexity and Quality of Seven OSS [Xie et al 2009]:

Xie et al studied evolutionary history of seven different types of open source software applications. The aim of study was two fold, one, to conclude that either these projects/applications validate the existing evolutionary trends (especially those proposed by Lehman) or not. And second to observe more evolutionary trends which have not been quoted still by any other researcher. Xie et al believe that observing evolutionary history of software application, with the aim of finding evolutionary trends, can help in reducing the cost of evolution, which sometimes, according to the past studied, equal to the 90% of the total software cost. It is a very comprehensive study, which used many different types of software measures, as well as tested all of the eight laws of Lehman. It, along with testing the existing evolutionary trends (like of Lehman), proposed new evolutionary trends as well as, first time, gave a clear distinction between growth and change measures.

2.6.1 Software Applications:

Xie et al studied total seven software applications which are all open source applications. Here is a brief introduction for each of those applications:

Samba: It is a Client-Server interoperatablility tool that enables clients having operating system other than windows, like UNIX, to interact with the windows server. This study covered 15 years evolutionary period of Samba (1994 to 2009) and observed its 89 official releases.

Sendmail: It is an Email transfer tool that uses different methods for mail delivery. This study covered 15 years evolutionary period of Sendmail (1993 to 2008) and observed its 57 official releases.

BIND: It is a commonly used DNS (Domain Name System) server. This study covered 9 years evolutionary period of BIND (2000 2009)) and observed its 168 official releases.

OpenSSH: It is network security tool that makes to avoid data from hijacking while travelling on network. It encrypts data before transmitting it on the network path, so that hijackers may not read it. This study covered 9 years evolutionary period of OpenSSH (1999 to 2008) and observed its 78 official releases.

SQLite: It is an SQL database engine existing in the form of software library. This study covered 8 years evolutionary period of SQLite (2000 to 2008) and observed its 172 official releases.

VSFTPD: It stands for Very Secure File Transfer Protocol Daemon. It is an FTP server used by Linux. This study covered 8 years evolutionary period of VSFTPD (2001 to 2009) and observed its 60 official releases.

Quagga: It is a tool suit used for development of software routers. This study covered 5 years evolutionary period of Quagga (2003 to 2008) and observed its 29 official releases

In this way, this study has covered 69 years evolutionary period (aggregate/sum of evolutionary periods of all seven applications) and observed 653 official releases in total.

2.6.2 Research Methodology:

Xie et al downloaded different versions of their studied applications from the public repositories of those applications. Many of those applications have both server and client suits. But they considered just the server part of applications and ignored the client part

as well as the test programs. The inclusion of client part and test programs will just result in increased values of measures, it have no impact on the trends of evolution. Means the trends will remain same to those which are observed in case of pure server suits. They merged all the source code of a release in one file using the CIL merger tool. Then they calculated LOC of that file to get the total LOC of release. Moreover, they used two tools for analysis of the source code, the ASTdiff (Abstract Syntax Tree difference) tool and the RSM (Resource Standard Metrics) tool. They developed ASTdiff themselves, but the RSM is a commercial tool. ASTdiff compares syntax trees of the source code files written in C language. It calculates a number of metrics for those files, including, changes in types (structures), changes in data types and definitions of global variables, changes in functions signatures and bodies, types (structures) added, global variables added, functions added, types (structures) deleted, global variables deleted, functions deleted etc. The RSM tool calculates cyclomatic complexity of the code.

2.6.3 Observations:

In order to observe evolutionary trends of their studied applications, Xie et al plotted different size and change measures against the time. The first graph was to plot cumulative number of changes, made to types/structures, global variables and functions, against time. The graph clearly showed an upward trend, means the modifications/changes are continuously happening. In other words, the application is continually changing. The graph, shown in Figure 16, is just for Samba, but other six applications have also shown same type of results. They observed that most of the changes were related to the functions and very few were related to the global variables and types/structures. For this purpose, they distributed changes among the three factors,

functions, types and global variables. The result clearly indicated that most of the changes were belonging to functions. The graph is shown by Figure 17. They also observed that additions are more common than deletions as the graph in Figure 16 clearly indicates. It was also noted that interface changes are much less frequent than the implementation changes. Means mostly changes belong to the implementation rather than the interface of the application. In order to test the law of self regulation (3rd law), they plotted incremental growth against release numbers. Lehman derived this law when he plotted the incremental growth of OS/360 against release numbers [Leh 78]. He observed ripples (small positive and negative adjustments) in the graph, so he concluded that software adjusts its size itself. Hence he concluded that the evolution process is self regulatory. Xie et al, too, plotted incremental growth against release numbers and observed ripples in it. The graph for OpenSSH is shown by Figure 18, however remaining six applications, too, showed same type of results. Figure 18 also showed that the incremental growth was neither in-variant nor it was declining. To strengthen this observation, Xie et al further plotted number of functions added to each release against the release numbers. The results, as shown by Figure 19, were same to those observed in Figure 18. Means the incremental growth was neither in-variant nor it was declining. Instead, incremental growth was found to be increasing. In order to test the law of continuing growth (6th law), Xie et al plotted LOC of each release against the release numbers. While calculating LOC, they ignored comments and empty lines. The graphs, as shown by Figure 20, were continuously moving upward. Thus it clearly indicated continuous growth in application size. This observation was strengthened by a previous graph (shown by Figure 16), which showed that additions are more common than

deletions. Larger number of additions will ultimately result in continuous increment/growth in the software size.

2.6.4 Conclusions:

All the applications were found to be continuously changing, with respect to functions (their bodies and signatures), structures and global variables, so the law of continuous change (first law) has been verified. Similarly, all applications were found to be increasing in their sizes (LOC, number of functions, number of structures, number of global variables), so the law of continuous growth (sixth law) also verified. The incremental growth, when plotted against release numbers was found to be having small positive and negative adjustments (ripples), which is in accordance with the law of self regulation (third law). Thus the third law has also been verified. The incremental growth was neither found to be in-variant nor it was decreasing, which indicated that applications were growing at a super-linear rate. It means their growth rate was increasing with time, instead of decreasing or remaining in-variant. It contradicts with the law of conservation of familiarity (fifth law), thus the fifth law has not been verified.

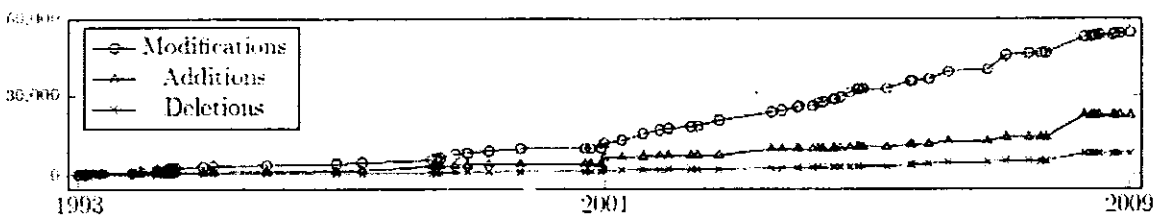


Figure 16: Cumulative number of changes plotted against time (for Samba) [Xie et al 2009]

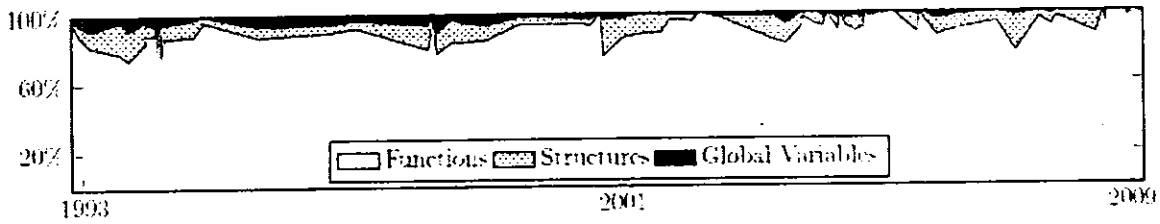


Figure 17: Changes distributed among functions, structures and global variables (for Samba) [Xie et al 2009]

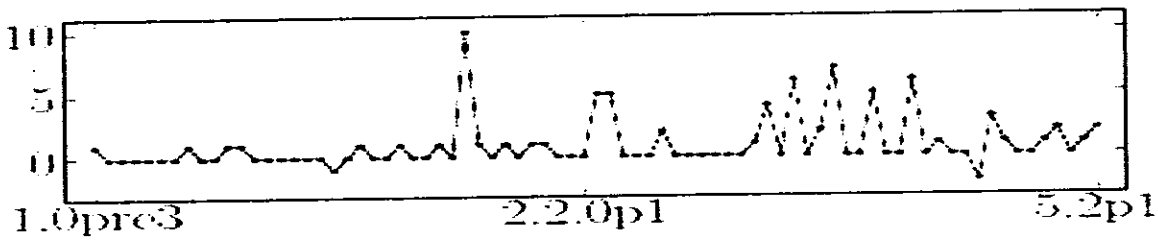


Figure 18: Incremental growth (measured in modules) plotted against releases (for OpenSSH) [Xie et al 2009]

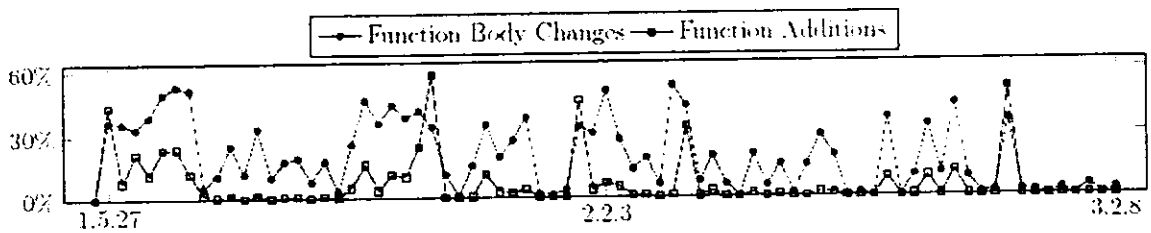


Figure 19: Incremental growth (measured in functions) plotted against releases (for Samba) [Xie et al 2009]

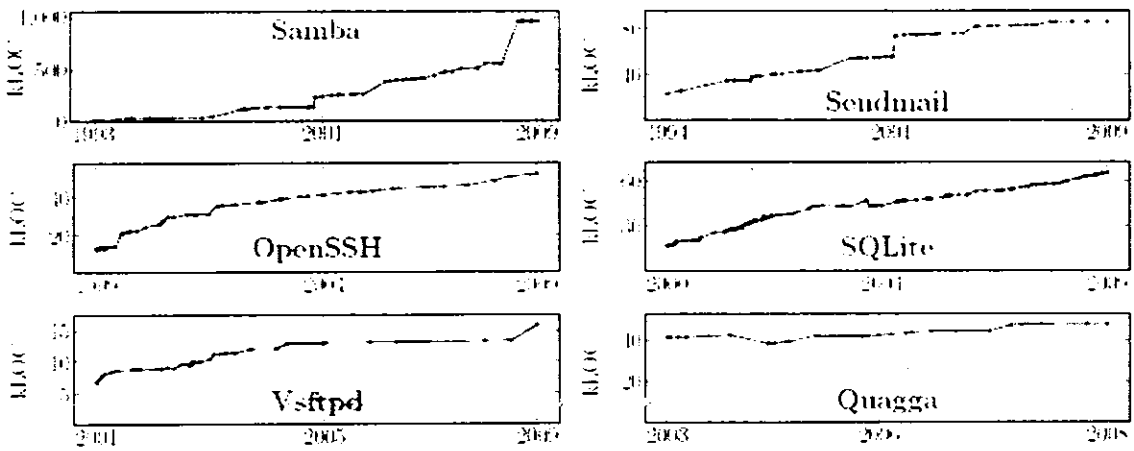


Figure 20: LOC plotted against time (for six applications) [Xie et al 2009]

2.7 Growth Rate of Thirteen OSS Applications [Herraiz et al 2006]:

Herraiz et al studied growth patterns of some open source applications using two types of metrics, LOC and number of modules (source files). Their study was basically aimed at comparing these two types of metrics. It was because, the studies which disproved Lehman laws for open source applications, like [Godfrey and Tu 2000] and [Robles et al 2005], used LOC or SLOC for measuring growth. Whereas Lehman, when suggested laws, used the metric, number of modules (source files). So they thought to use these two types of metrics on same projects/applications in order to observe that either they produce same results or their results differentiate from each other. Besides this they also tested Lehman laws on their studied applications. It was obvious that when they will calculate those size metrics for their studied applications, they must observe that either those applications have grown at rate of Lehman or not.

2.7.1 Software Applications:

They studied total thirteen applications, all of which are open source in nature. Ten of which are basically packages included in Debian GNU/Linux. Debian is perhaps the largest software application of the world consisting of more than 229 millions LOC. Most of the open source applications in the world are written for Debian GNU/Linux. Thus it can be said that Debian is a representative of the whole community of open source applications and its trends can be said to be the trends of most of the open source applications. Herraiz et al included ten largest packages of Debian in their study, whose names along with the evolutionary period are as under:

1. Amaya: 8 years
2. Evolution: 7.5 years
3. Kaffe: 7 years
4. Prc-Tools: 5 years
5. Python: 8 years
6. Wine: 7.5 years
7. wxWidgets: 2.5 years
8. XEmacs: 7.5 years
9. XFree86: 8 years
10. Linux: 13 years

Besides these ten packages of Debian, they included three BSD family kernels in their study, FreeBSD, OpenBSD and NetBSD. The evolutionary period covered for each of these kernels is as under:

11. FreeBSD: 12 years
12. OpenBSD: 7.5 years

13. NetBSD: 13 years

2.7.2 Research Methodology:

They, first of all, defined the criteria for selection of applications to be studied. For this purpose, they decided to include only those applications for which a CVS repository was available. Secondly they decided to include those applications which have a reasonable evolutionary history, so that some conclusions can be made about their evolution trends. Thus they decided to include applications having evolutionary history spread over thirty months or more. Finally they decided to study applications which are large in size. So they selected all applications meeting this criteria, except the Linux kernel, which was not meeting one point of criteria, that is, its repository had not been maintained using a CVS server. But they included it in their study because prior studies of Godfrey [Godfrey and Tu 2000] and Robles [Robles et al 2005] included it, and they wanted to compare their results with prior studies. They downloaded periodic snapshots of the applications from their publically available CVS repositories, suing a tool GlueTheos. Twelve of their studied applications were maintained in the form of CVS repositories, so they got their snapshots. One application, however, was not maintained using CVS server, which is Linux. Linux is available in the form of versions at its website in compressed form. So they downloaded those compressed versions and decompressed them to get the code. For each of the other twelve applications, all six months snapshots were downloaded from the date of first commit till the date of study. Then they calculated two types of measures for different snapshots/versions of each of those applications and stored results in a database for future use and result interpretation. In

order to calculate these measures, means LOC and number of modules, a tool SLOCCount was used.

2.7.3 Observations:

In order to compare the two metrics, SLOC and number of modules, Herraiz et al, first of all correlated both metrics against each other, taking number of modules along x-axis and SLOC along y-axis. The result was a clear linear graph, in case of all thirteen applications as shown for Linux by Figure 21.

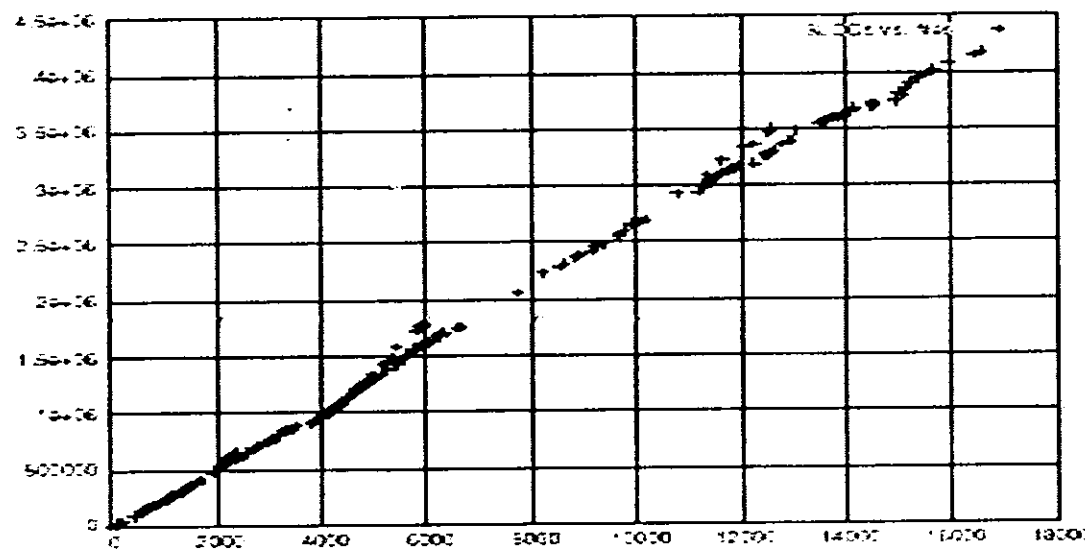


Figure 21: Number of modules against SLOC for Linux [Herraiz et al 2006]

In order to observe the growth pattern of these applications, they plotted version size against time, taking time along x-axis and version size along y-axis. Since versions size was measured by two metrics, SLOC and number of modules, so they plotted two graphs for each application, one for SLOC and second for number of modules.

The results of these plots were mixed. For most of the applications (six of thirteen), the result was super-linear, where as for four applications, it was linear. And for three

applications, it was sub-linear. The plots for FreeBSD and Linux have been shown by figures 22, 23 and 24.

The applications along with the nature of their growth graphs are as under:

- Amaya: Linear
- Evolution: Sub-linear
- Kaffe: Super-linear
- Prc-Tools: Super-linear
- Python: Linear
- Wine: Linear
- wxWidgets: Super-linear
- XEmacs: Sub-linear
- XFree86: Sub-linear
- Linux: Super-linear
- FreeBSD: Linear
- OpenBSD: Super-linear
- NetBSD: Super-linear

2.7.4 Conclusions:

The linear graph of SLOC against number of modules clearly showed that both metrics are equal, means both generate same type of results. Thus it doesn't matter, what type of metric we use for observing growth pattern, because it doesn't change results. Both metrics will show same type of growth trends. If some study has used SLOC, then it can be said that same type of results would be observed, if we repeat analysis using different metric, means number of modules. Moreover the super-linear graphs of the six of

thirteen applications showed that most of the open source applications grow at a super-linear rate; means their growth rate increases with time rather than decreasing. Thus the general growth trend of open source applications is different from those of commercial applications, which, according to Lehman, follow a sub-linear growth trend. Sub-linear means, their growth rate decreases with time. In this study, it was observed that most applications were growing either at an increasing or a steady rate. Only three were there, which were found to be loosing their growth rate. Thus the results of this study clearly contradict with the findings of Lehman.

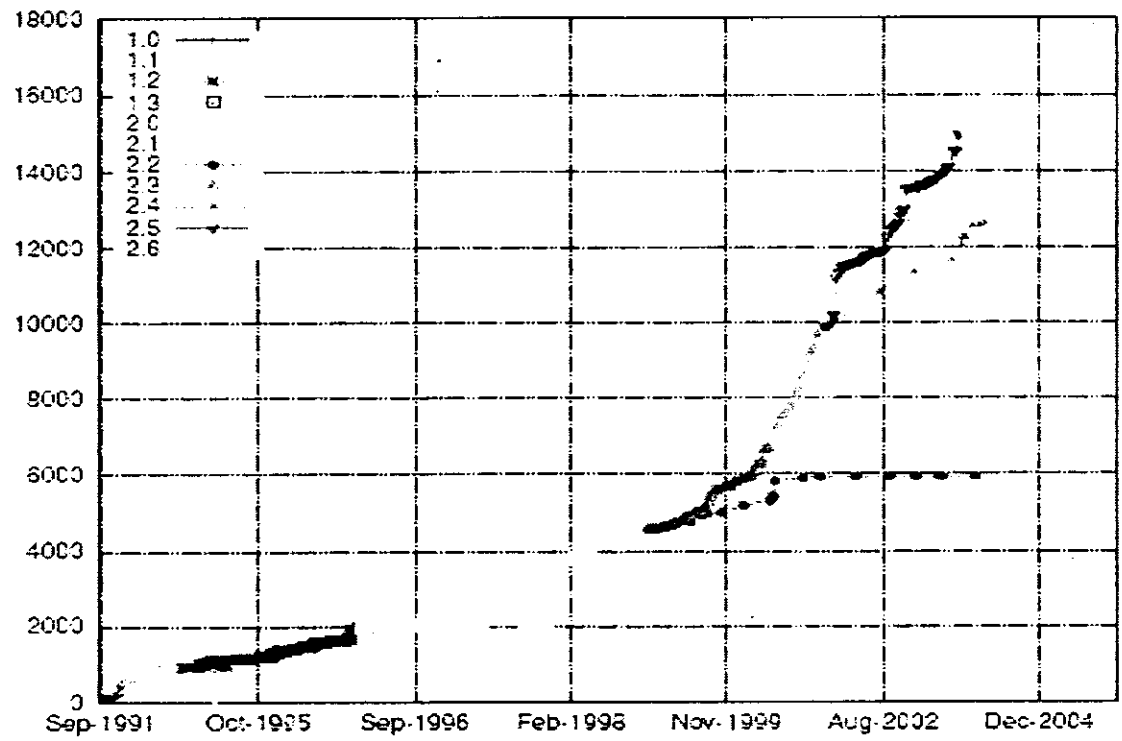


Figure 22: Number of modules plotted against time for Linux [Herraiz et al 2006]

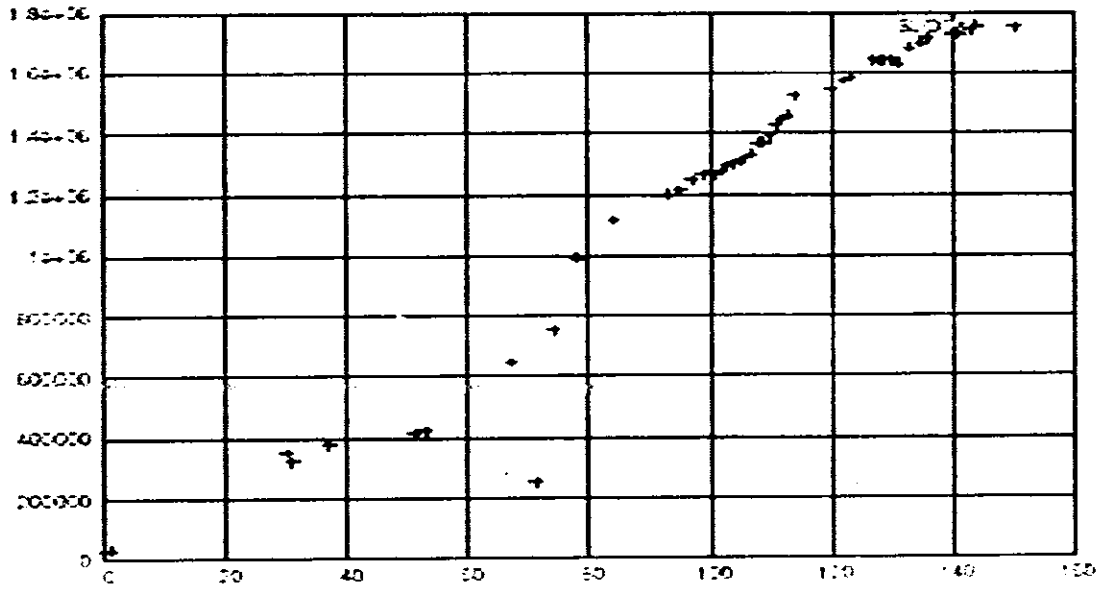


Figure 23: SLOC plotted against time for FreeBSD [Herraiz et al 2006]

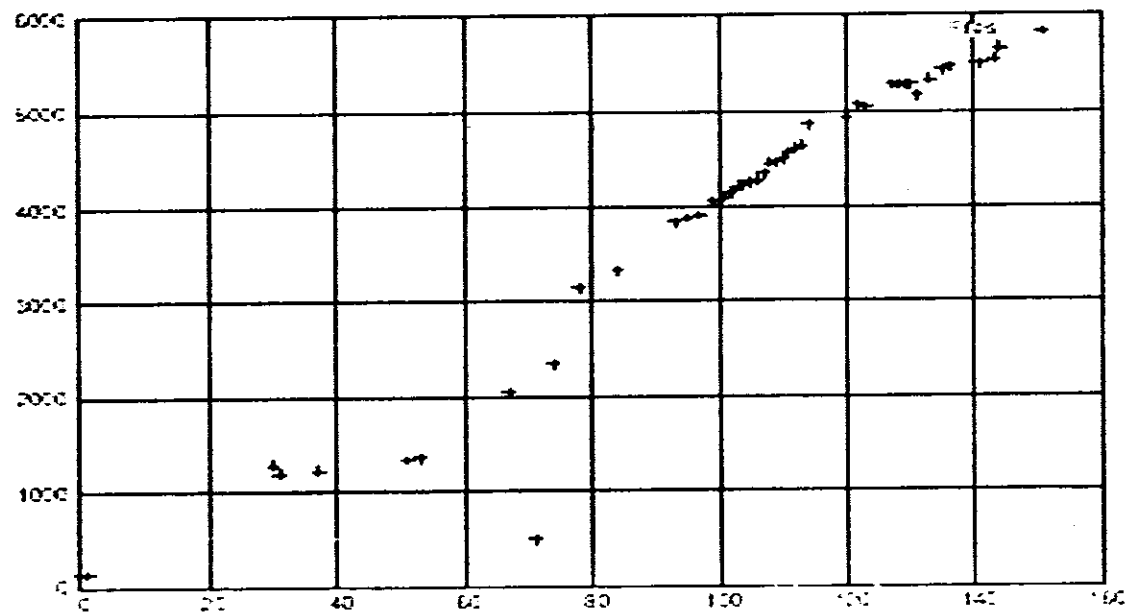


Figure 24: Number of modules plotted against time for FreeBSD [Herraiz et al 2006]

2.8 Growth and Change Trends of Four OSS [Ali and Maqbool 2009]:

Ali and Maqbool studied evolution of small scale open source software systems using different type of measures. They were feeling that the measure used to observe

evolutionary behavior plays an important role regarding results. It is possible that different measures show different evolutionary behaviors of an application. They performed their analysis using the measures used by Lehman, i.e. number of modules. Then they repeated their analysis using other measures, introduced purely by themselves and compared both results. At the end they concluded that different measures represent different aspects of the application and their results may vary from each other. Therefore selection of an appropriate measure has much significance in the study of evolutionary behavior of software applications.

2.8.1 Software Applications:

They studied four different open source applications which are all small scale applications. Here is a brief introduction of all those applications with the number of versions studied / observed of each one:

KTorrent: It is a light open source application used for exchange of data. This study covered 45 releases of KTorrent.

GNOME: It acts as a desktop for users and developers of Linux. It is very user friendly. This study covered 168 releases of GNOME.

Konversation: It acts as an IRC client. It is used for KDE desktop environment. It is too very user friendly. This study covered 13 releases of Konversation.

Evince: This application is used as a document viewer which allows viewing documents of many different formats. This study covered 46 releases of Evince.

2.8.2 Research Methodology:

Ali and Maqbool believe that just observing number of modules in a release is not sufficient to conclude something about evolutionary trends. But it should also be focused that how many modules have been added, how many deleted and how many modified. Just number of modules can't give us a true picture of evolutionary trends. We can see more insight, if we consider the three counts (modules added, deleted and modified) separately, instead of just total number of modules. They strengthen their idea with the help of an example. Suppose a release, say R1, has ten modules and the next release, say R2, has twenty modules. If we consider just number of modules, we will say that 10 modules have been added to release R2. But we don't know the inner picture. It is not necessary that just 10 new modules would be added to release R2, but there are many possibilities. Like, it is possible that 15 new modules would be added, but 5 old modules of release R1 would be deleted. In this way the net module count for release R2 will also be 20. Thus if we focus on module count just, then we will lose inner information and will not be able to determine the exact evolutionary trends. It's the reason they didn't rely solely on total number of modules in a release. But they also counted three different measures, number of modules added in a release, number of modules deleted from a release and number of modules modified of a release. This enabled them to see more deeply into the evolutionary trends of their studied application and they pointed out certain those trends which were not possible to observe by counting just total number modules.

2.8.3 Observations:

In order to observe the evolutionary characteristics of these four applications, Ali and Maqbool first used the measure used by Lehman, i.e. number of modules (number of

source files) in each release. By using this measure, they plotted two types of graphs for each application; one for total releases size (total number of modules in release), and second for modules increment (module difference between release i and i-1) in each release. Both of the measures were plotted against release numbers. The result of the first plot (total release size in modules versus release number) showed that most applications were growing at a linear rate, as shown by figures 25, 26, 27 and 28. Only one application, i.e Konservation was decreasing in size, instead of increasing. Similarly the second plot (module increment against release number) also showed linear trend, but in downward direction. Means the module increment rate was decreasing with the passage of time, as shown by figures 29, 30, 31 and 32. In order to expand their observation, they plotted three more measures against the release numbers, which are number of modules added in a release, number of modules deleted from a release and number of modified modules of a release. These three plots helped to observe the evolutionary trends in more depth. And many those issues were appeared which were hidden when only total number of modules in a release were considered. It was clearly observed that taking just the module difference between two releases into account is not enough. In order to get the complete picture of evolution, it must be noted that how that difference is distributed among additions and deletions. Finally they plotted total number of changes in a release against the release numbers. The total number of changes was a sum of the above mentioned three measures, means number of added modules, number of deleted modules and number of modified modules. They named this measure as incremental changes in a module. The number of incremental changes between modules i and i+1 can be calculated by taking summation of the number of modules added in

release i, number of modules deleted from release i and number of modified modules of release i. this measure is equitant to that used by Lehman to calculate incremental effort [Leh 78]. Lehman named this measure as number of modules handled in a release.

2.8.4 Conclusions:

The plots of incremental changes were near to straight horizontal lines for all four applications. It showed that number of incremental changes in successive releases remained constant or in-variant. It clearly proved the fourth Law of Lehman, according to which the incremental effort remains in-variant. Lehman concluded this law in his study of OS/360 when he observed that number of modules handled in successive releases remains in-variant [Leh 78]. From this, he concluded that effort spent on each release remains in-variant. In this study, because, the number of incremental changes remained in-variant, therefore it can be concluded that incremental effort remained same. It is clearly in accordance with the findings of Lehman. Moreover the plots of module difference between successive releases were also found to by nearly in-variant with a slight downward trend. It is in accordance with the fifth law of Lehman, according to which incremental growth remains in-variant or it slows down with the passage of time. The in-variant growth and change trends proved the observations of Lehman, observed by him in different case studies and formulated in the form of fourth and fifth laws [Leh 78]. The separate plots of number of modules added and number of modules deleted disclosed certain other evolutionary trends, which were hidden when just module difference was plotted. These trends helped to know about maturity and stability of an application. An application with larger number of deleted and added modules can be considered as immature and unstable. Because the greater values for these two measure

show that modules are not easy to modify. So developers prefer to replace old modules with new ones for correction of errors or enhancement of functionality rather than modifying the existing modules. Similarly a larger value for the number of modules modified, represent that code is more stable and modules are easy to be modified. The period in which there is larger value for number of modules added is a period in which main focus is enhancement of functionality. Similarly the period in which there is larger value of number for number of modules modified is a period in which main focus is correction of errors.

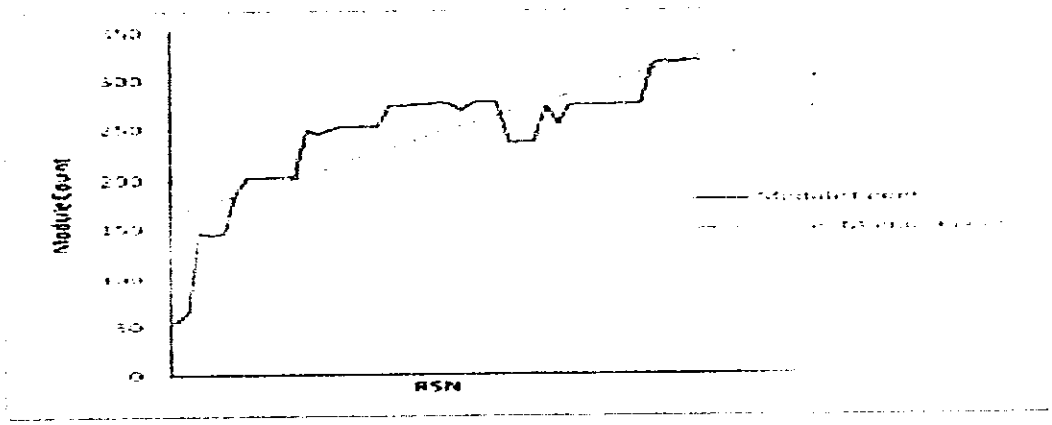


Figure 25: Total release size (in number of modules) plotted against release numbers for KTorrent [Ali and Maqbool 2009]

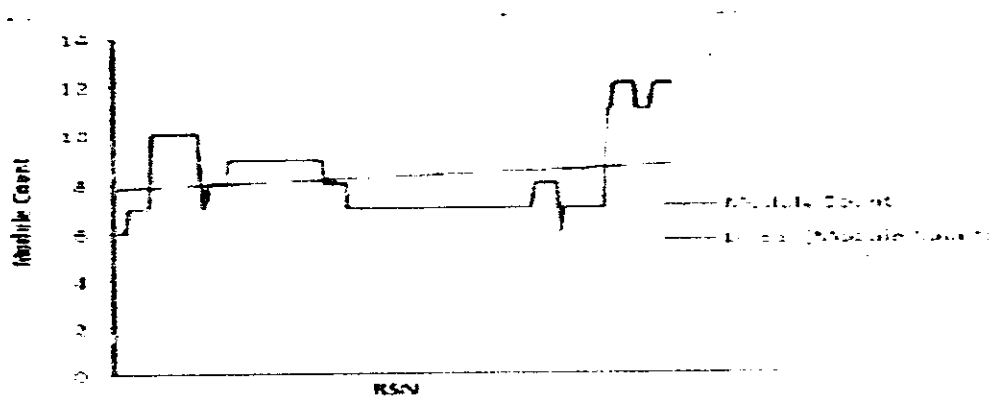


Figure 26: Total release size (in number of modules) plotted against release numbers for GNOME [Ali and Maqbool 2009]

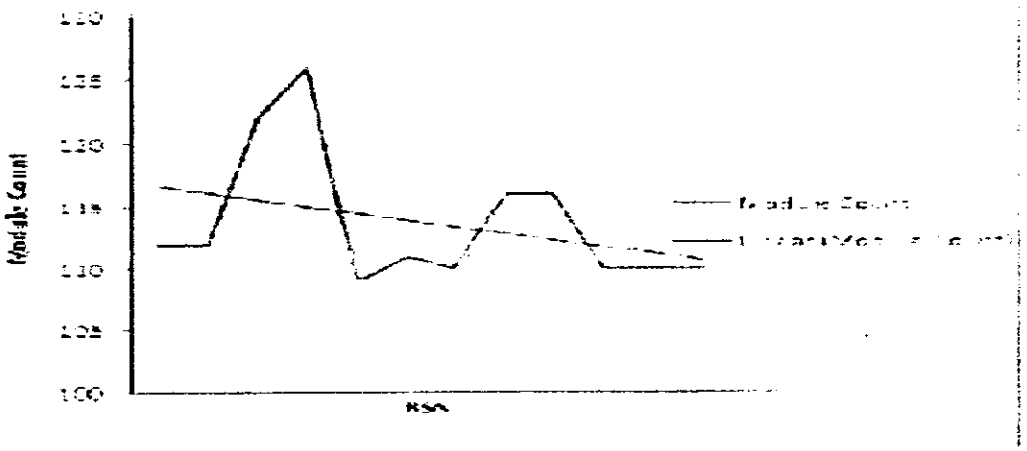


Figure 27: Total release size (in number of modules) plotted against release numbers for Konversation [Ali and Maqbool 2009]

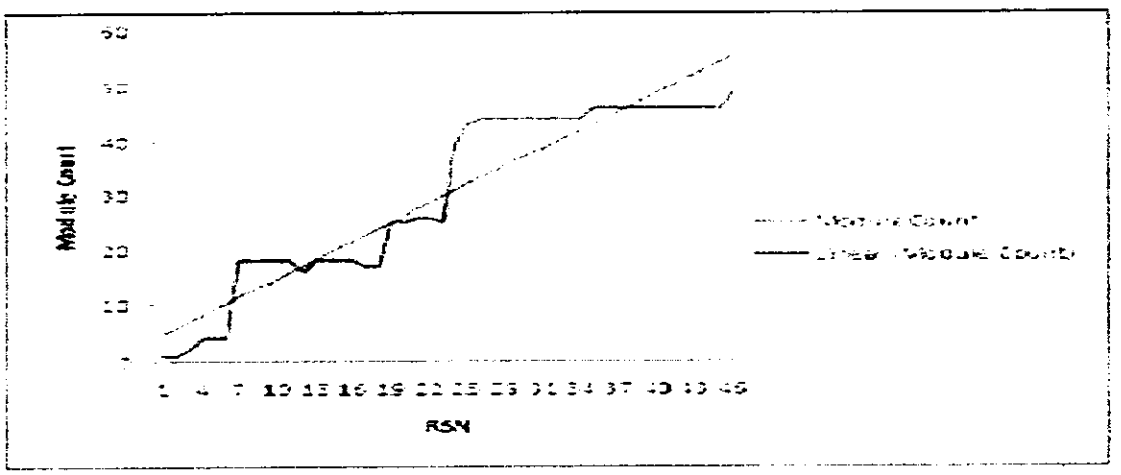


Figure 28: Total release size (in number of modules) plotted against release numbers for Evince [Ali and Maqbool 2009]

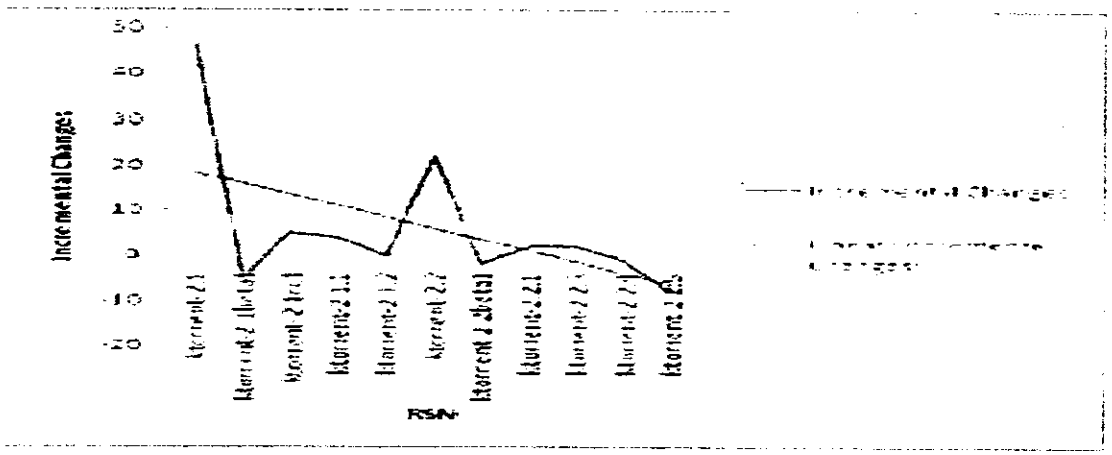


Figure 29: Module difference/increment plotted against release numbers for KTorrent [Ali and Maqbool 2009]

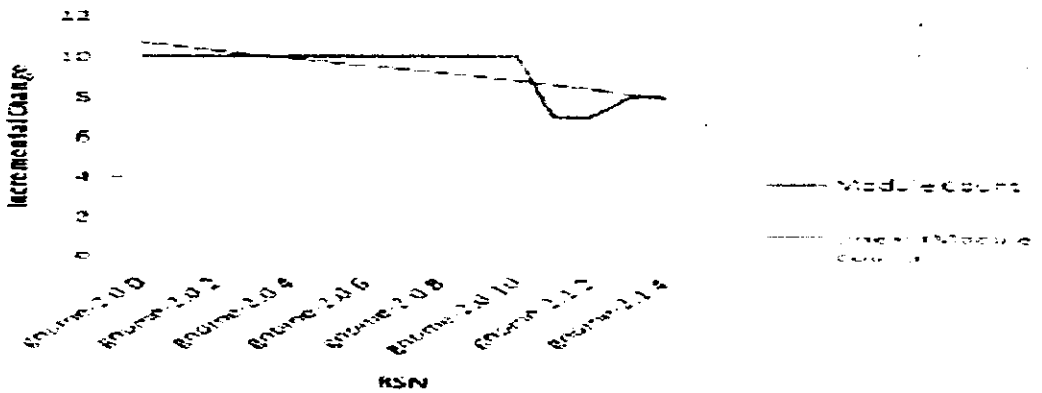


Figure30: Module difference/increment plotted against release numbers for GNOME [Ali and Maqbool 2009]

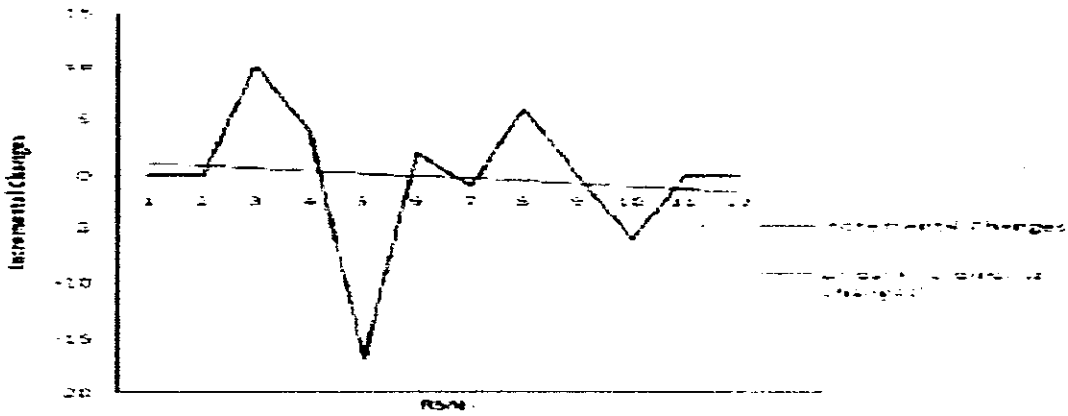


Figure 31: Module difference/increment plotted against release numbers for Konversation [Ali and Maqbool 2009]

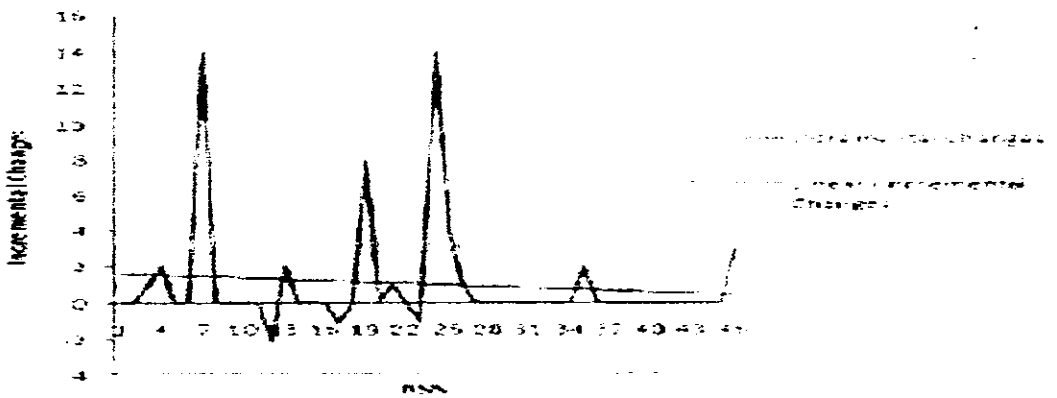


Figure 32: Module difference/increment plotted against release numbers for Evince [Ali and Maqbool 2009]

2.9 Growth, Complexity, and Quality of Linux [Israeli and Feitelson 2009]:

Israeli and Feitelson studied the evolutionary behavior of a commonly used, well known, long lived and large scale open source application, the Linux kernel. There was just one objective of study, to determine that either this application confirms to Lehman laws or

not. Although this application, Linux kernel, had already been focused in many of the previous studies, but Feitelson's work is unique in its nature. The three main distinguishing factors are:

1. They studied total 810 releases of Linux. And none of the past studies had observed too large number of versions.
2. They used many different types of metrics related to size, growth, complexity, quality and effort, whereas none of the past studies used so large variety of metrics.
3. They tested seven of the eight laws of Lehman using numerical measures, whereas just one of the past studies covered so many laws [Xei 2009] but they didn't use numerical measures for all of those.

Thus on the basis of all these points, it will not be wrong to say that this study is unique in its nature.

2.9.1 Software Applications:

As mentioned above, this study used the open source application Linux kernel for testing the Lehman laws of software evolution. Linux is the application used by most of the open source evolution studies. Its reasons include the wide spread use of Linux throughout the world, the interest of open source developer's community in it, free and easy availability of its different releases source codes. The first version of Linux was released in March 1994. At the time of study, August 2008, total 810 different types of major and minor versions had been released. This study covered all those versions.

2.9.2 Research Methodology:

They got source code of different releases of Linux from the Linux website and calculated many different types of measures for those releases in order to observe the trends of changes, growth, complexity, quality and effort. In this process they used a commercially available CASE tool as well as developed their own tool for their specific needs. In order to observe growth, they calculated two size measures LOC and number of modules. It is obvious that size is directly related to growth. Increasing size can be validly assumed as continuous growth. Because it indicates that new code and functions are being added to cover more and more features of the real life. While counting LOC, they considered only those statements which were executable. Means they ignored comments and blank lines. Similarly while counting number of modules, they considered each function as a module. Means in order to count number of modules, they counted number of C functions in the release. In order to observe changes that happened in different releases, they calculated size of "arch" and "drivers" sub directories of Linux kernel. They realized that growth and change are very close to each other, even both overlap. Therefore sometimes it is harder to distinguish among both. There are many changes in code that can be put in both of the categories, change and growth. However they distinguished change from growth on the basis of code directories. They assumed that enhancements in the two subdirectories "arch" and "drivers" are basically for accommodating the changes in real system rather than covering new features of real world. So the size increment of these two directories is basically an indicator of changes rather than growth. The "arch" subdirectory contains code related to the processor architecture and the "drivers" subdirectory contains all device drivers. In order to measure work rate, they used three different measures. One is the number of developer

participating in the code development of Linux. Second, the number of files added, deleted, grew and shrunk in a release. And third, the release rate, means the time interval between releases. Second of these measures was same to that used by Lehman to observe work rate in the study of OS/360 [Leh 78]. Lehman called this measure as number of modules handled in a release. In order to observe the law of "conservation of familiarity", they again observed the quantity of changes introduced by a release. But this time they just examined changes quantity rather than number of changes. They compared different types of successive releases in general to conclude that either the quantity of changes is limited to the point where familiarity will not suffer. This law, however, has not been tested by them on the basis of size measures, LOC and number of modules. It is also important to note that nearly all other studies have tested this law on the basis of size measures. In order to test the law of self regulation, they followed the same pattern as was adopted by Lehman. Means they plotted the size difference of successive releases against the release numbers. In order to observe complexity of the code, they used two commonly used measures, MCC (McCabe Complexity) and the Halstead complexity measures. The MCC can be calculated as:

MCC = number of conditional branches + 1

An extended version of MCC, named as EMCC, is also available, which counts number of comparisons in each condition, rather than counting the condition as a whole. This study also calculated EMCC for different versions of Linux. The Halstead complexity is a measure of number of operators and operands in the code.

2.9.3 Observations:

In order to observe the growth pattern, Israeli and Feitelson plotted release size against time. As mentioned above, the two different measures were used for size calculation, LOC and number of modules. Each of these measures was separately plotted against time. Both graphs generated very similar results. According to results of both graphs, Linux had been grown at a mix of super-linear and linear rates. Before year 2003, the growth pattern was super-linear, whereas after 2003 it was found to be linear. The graphs have been shown by figures 33 and 34. In order to observe the law of continuing change, the size of two subdirectories "arch" and "drivers" was plotted against time. It has already been mentioned that Feitelson realized that the size of these two subdirectories is directly concerned with the changes. The results were similar to those which were observed in case of whole Linux. Means both subdirectories were growing at a linear or super-linear rate. It was observed that the open source developers community participating in the code development of Linux was increasing rapidly. Means the developers taking interest in the Linux development were increasing in number. Linux was getting attention of more and more developers with the passage of time. The developers interest in Linux was basically assumed an indicator of the rate of work. Similarly in order to observe rate of work, the total number of files added, deleted, grown and shrunk were plotted against time. And it was found that the number of files added, deleted, grown and shrunk in each release remain almost same. The release rate was also calculated for the sake of observing work rate. It was found that the dominant trend of release rate was staying in-variant. The size increment of each release was also plotted against release numbers. The size increment was measured as number of modules added in a release. Or it will be more appropriate to say, the difference in number of modules

between every two successive releases. It was observed that average number of modules added in a release remained almost same. The actual line was found to be fluctuating along average line with small and repetitive ripples. The graph has been shown by figure 35.

2.9.4 Conclusions:

Linux has been found to be continually growing at a linear or super-linear rate, so it clearly validated the sixth law of Lehman, according to which a software application continually grows throughout its active life. Similarly the continuous growth of the two subdirectories "arch" and "drivers" validated the first law, according to which a software application continuously changes throughout its life. The overall complexity of code was increased. However there were also instances of reduction in complexity at functions level. Perhaps this reduction was due to certain efforts which were made for controlling complexity. Such efforts were evident and could be clearly observed. So it can be said that the second law has been validated in case of Linux, according to which software complexity increases continuously, unless work is done to control it. The work rate was found to be in-variant with two perspectives. However from one perspective, it was found to be increasing. It has been observed that the number of files handled (added, deleted, grown or shrunk) in each release remained almost same, it clearly indicated an in-variant work rate. Similarly the interval between every two successive releases of same type has also been found to be nearly same. Thus an in-variant release rate can also be considered as an indicator of the in-variant work rate. However the number of developers participating in Linux was found to be increasing rapidly, that is an indication of the increasing work rate. Because two of these perspectives indicated the in-variant

work rate, so it can be said that fourth law of Lehman also validated, according to which the average work rate, throughout the life of software, remains same. The two types of releases, development releases and stable production releases confirmed the law of familiarity conservation. Because it was observed that none of theses two types of releases introduced too many changes which affect familiarity. The general trend of theses two types of releases was that each new release introduced small amount of changes, which did not create any severe problem regarding familiarity of the software. The major releases, however, did not confirm this law. These releases were found to be introducing large amount of changes as compared to their predecessor releases. Even in some cases, a new major release was too much different from its predecessor that users preferred to use outdated release instead of learning later one. Thus it can be said that Linux didn't fully validate the law of "conservation of familiarity". It was also observed that the quality of Linux didn't decline with its evolution. It was basically a result of continuous efforts made to maintain quality. It can, however, be assumed that if those efforts were not present then new versions may lose their quality as compared to the old ones. Thus Linux validated the seventh law of evolution according to which quality of software application declines with its evolution unless efforts are made to maintain it. The module increment made in each releases was found to be in-variant on average. The graph line was found to be fluctuating on average line with positive and negative ripples. It is an indication of the trend that the number of modules added to each release tends to remain same. A release with large number of added modules was found to be following a release with small number of added modules to balance the average. It means that growth and anti-growth forces have balanced each other to maintain the level of

increments. It is in accordance with the law of self regulation, according to which desire for growth and constraints on growth balance each other, and as a result software grows at a constant rate.

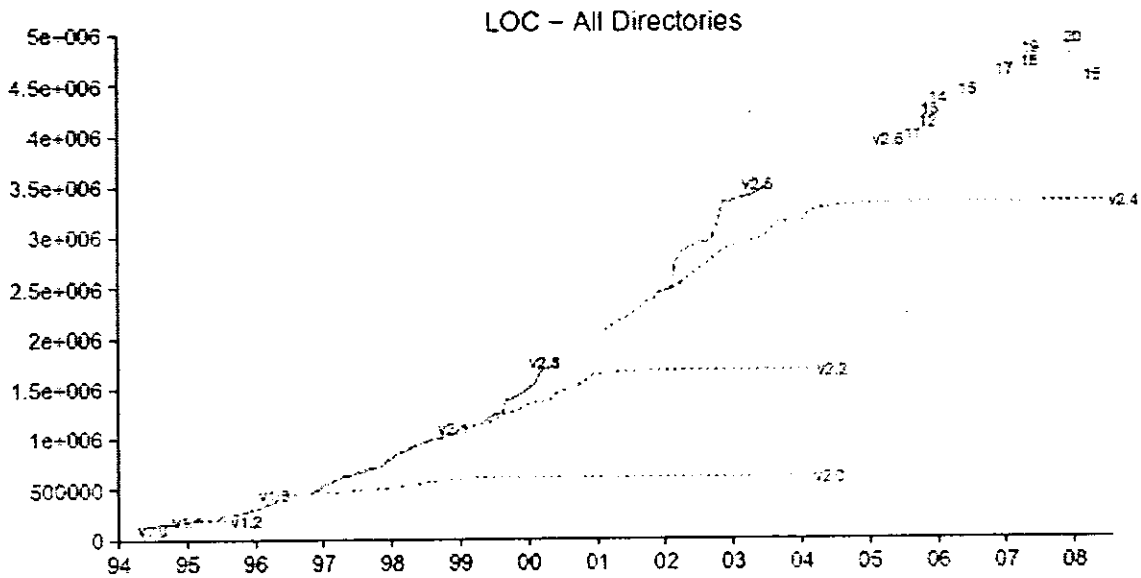


Figure 33: The total LOC of releases plotted against time [Israeli and Feitelson 2009]

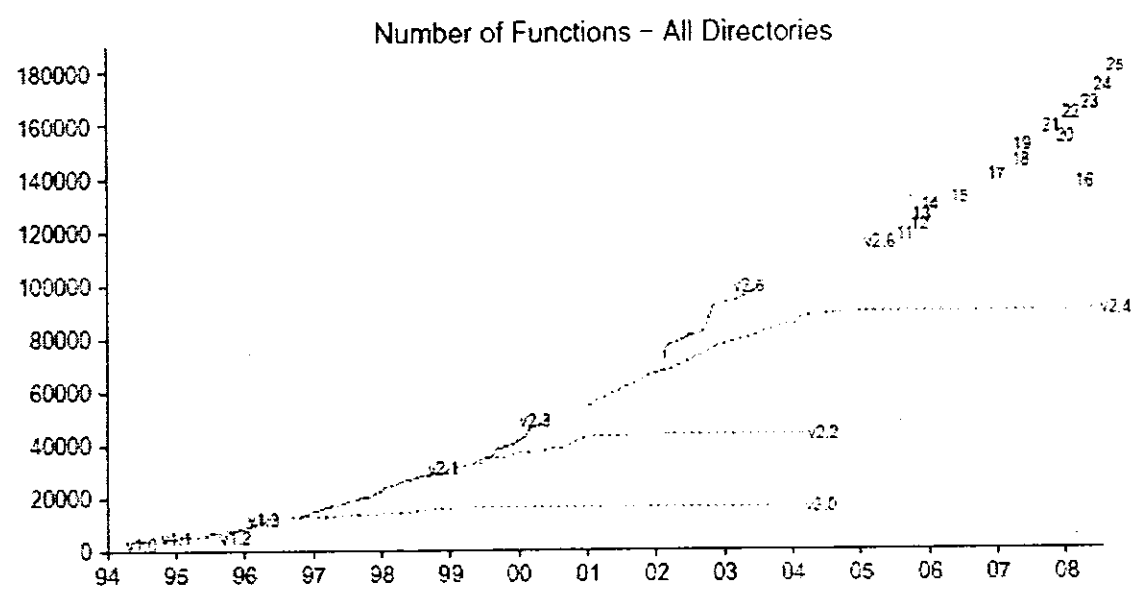


Figure 34: The total number of modules of releases plotted against time [Israeli and Feitelson 2009]

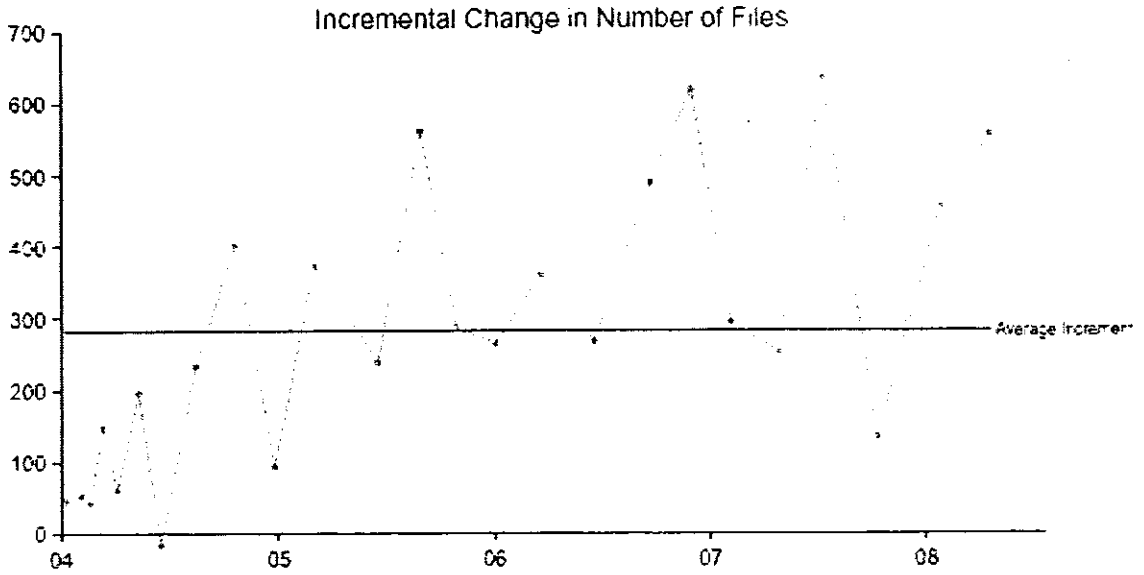


Figure 35: The increment in releases plotted against release numbers [Israeli and Feitelson 2009]

2.10 Growth, Change, and Complexity of Nagios [Bonkoski 2007]:

Bonkoski studied evolutionary behavior of a widely used relatively small size open source application with respect to growth, change and complexity. The one point aim was to compare evolutionary trends of open source applications with those of commercial applications. At that time, the evolutionary trends of commercial in-house applications had been determined and had been summarized in the form of eight laws by Lehman and his companions [Leh 78], [Leh and Ramil 2001] and [Turski 96]. These laws were proved to be valid for all type of commercial in-house applications, regardless of the type of organization and its environment. All case studies of commercial applications proved these laws. So these have been universally accepted and believed. However, validity of these laws for open source applications was still a question. Because certain studies had

disproved some of these laws in case of open source applications. And it was o hot topic for research community that either those laws, which were derived during the study of commercial applications, hold for open source applications or not. That's why Bonkoski performed this case study to add his contribution in this area of research.

2.10.1 Software Applications:

The software application studied by Bonkoski was a widely used application, named as Nagios. Nagios is used as a system administrator tool to ensure system's availability. It checks availability of the critical services of a business application and informs administrator about any availability related problems before those problems are faced by end user. This application is written in C language and has been proved very popular in users and developers community within a short period. According to the facts available on sourceforge.net, Nagios was downloaded more than 1.2 million times during the period of 4.5 years (Nov 2002 to Apr 2007). This application is relatively smaller in size as compared to those observed by other studies in this research area, like, [Godfrey and Tu 2000], [Robles et al 2005] and [Israeli and Feitelson 2009] etc. These studies observed very large size open source applications like operating system kernels etc. Another difference of this application form those of others in this area is that, this application didn't have a very long evolutionary history. Its first version was released in 2002. And at the time of study, 2007, it had just three major versions. Thus at the time of study, it had just five years evolutionary period to be observed. During these five years, its three major and many minor versions had been released. This study did not observe all of those versions, but it selected nine of them for observation.

2.10.2 Research Methodology:

This study observed Nagios from two perspectives, growth/change and complexity. For this purpose they used seven different measures. Three were size related measures used to observe change/growth and four were complexity related measures. As mentioned above, they studied total nine versions of the application released with a period of nearly five years. The source of versions was sourceforge.net which is a large repository of open source applications and has been used as a source of information in most of the prior studies in this area. In order to calculate different measures, they used an evaluation copy of Karakatau Metrics tool. This tool is written in Java language. It imports source code and calculates different types of software measures. The results generated by this tool were parsed using a scripting language and loaded into a database. Then database was queried to extract data. The results of queries were saved into Excel files to generate graphs. The size related measures used to observe change/growth trends were, SLOC, number of functions and number of downloads. Bonkoski believed that SLOC alone is not enough to represent all aspects of software growth. There are certain aspects which can not be represented by SLOC, like code refactoring or optimization. So they counted number of top level functions as well, to observe a full picture of growth. Moreover they also counted number of times, Nagios was downloaded by users community and considered the number of downloads as an indicator of growth and change. It is the unique point of this study because no other study has used number of downloads measure as an indicator of growth. The complexity related measures used in this study are, McCabe's cyclical complexity, Essential Complexity, NEST complexity and Halstead complexity. The first measure, McCabe's complexity is calculated on the basis of number of decisional paths used in the code. The Essential complexity is calculated same

as McCabe's complexity, but before that code is restructured to its most primitive parts. The NEST complexity is calculated on the basis of nested structures used in the methods. And finally the Halstead complexity is a measure of the number of operators and operands used in the code.

2.10.3 Observations:

In order to observe the change and growth trends, two of the size measures, SLOC and number of functions were plotted against the release numbers. The result was a clear increment in both of these measures along with releases. Means each new release was observed to be increasing in SLOC as well as in number of functions, as shown by figures 36 and 37. Thus it proved that the application, means, Nagios has continually grown and hence has continually changed. It was also observed that the growth rate was linear with respect to each of these two measures. The number of downloads as recorded from the website sourceforge.net was plotted against time (years) and it was found that Nagios had continuously gained attention of more and more users. User community was found to be attracting towards Nagios rapidly. The results of plot are a clear evidence of this claim as shown by figure38. It can clearly be observed that the graph line has moved upward rapidly, thus indicating that number of downloads have increased every year with a significant figure. As already mentioned, Bonkoski has considered increasing interest of user community in an application as an indicator of applications growth. So he concluded that because number of downloads of Nagios have increased rapidly, therefore this application has grown continually. In order to observe the complexity of application over time, the different complexity measures were plotted against releases. As already mentioned, four different complexity measures have been used in this study, McCabe

cyclical complexity, Essential complexity, NEST complexity and the Halstead complexity. Each of these measures was plotted separately and results were shown by figures 39, 40, 41 and 42 respectively. It can, once again, be observed that each graph line has moved upward nearly with each new release, thus indicating that complexity of the application has increased in all perspectives. There is just one instance of reduction in complexity in case of Essential complexity, when transition occurred from release 2.8 to 3.0 (figure 40). Except this one case, there is no instance of reduction in complexity.

2.10.4 Conclusions:

The application (Nagios) has been found to be continually increasing in size from all perspectives, which proved that Lehman's first and sixth laws apply to Nagios. According to these two laws, a software application continuously changes (first law) and continuously grows (sixth law). Thus we can say that these two laws of Lehman have been verified in this study. Moreover the graphs of different complexity measures also showed an upward trend, means complexity of application has increased in different perspectives. It is clearly in accordance with the findings of Lehman, which, he summarized in the form of second law, "increasing complexity". According to this law, the complexity of an application increases with time. As the complexity of Nagios has increased with time, thus it proved that this law has too been validated in this case.

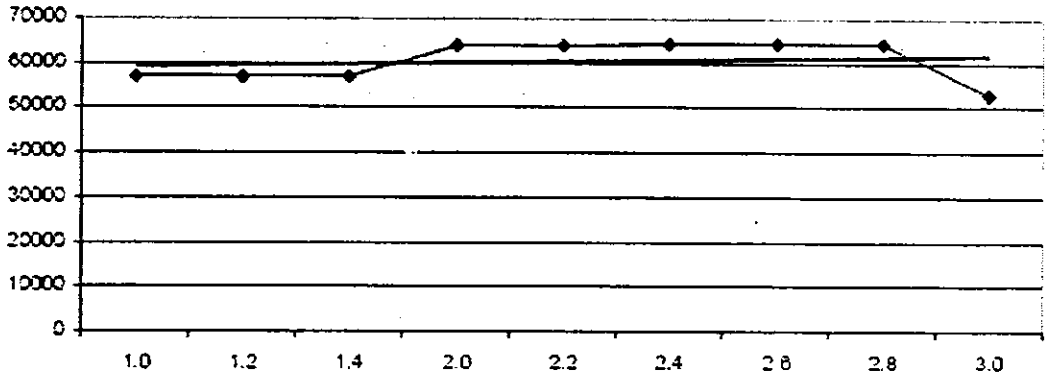


Figure 36: The SLOC plotted against releases [Bonkoski 2007]

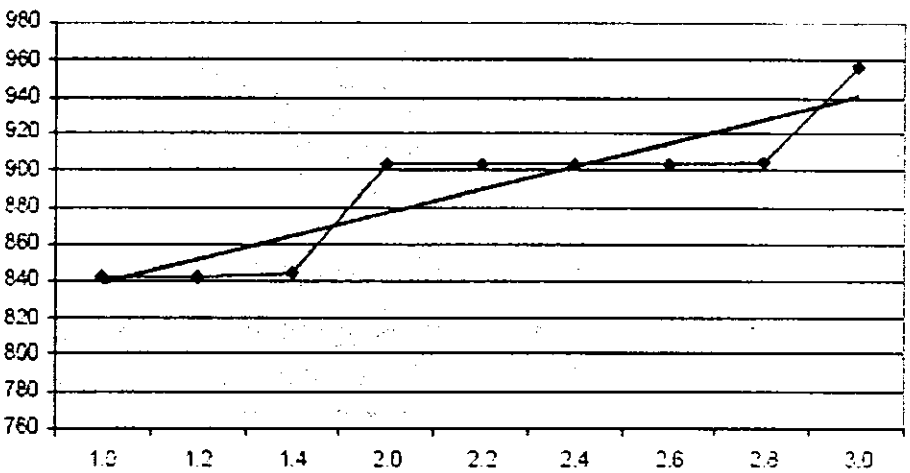


Figure 37: The number of functions plotted against releases [Bonkoski 2007]

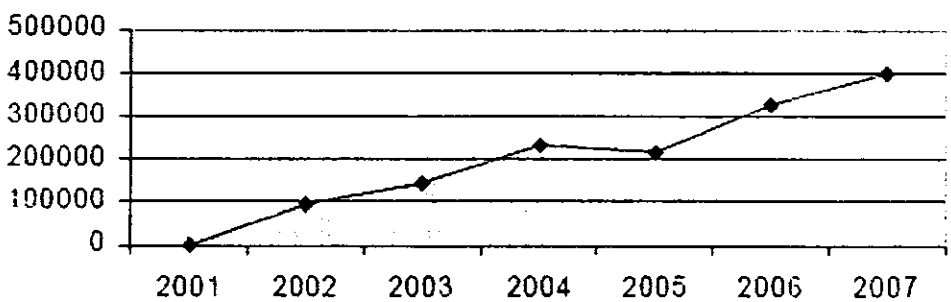


Figure 38: The number of downloads plotted against time [Bonkoski 2007]

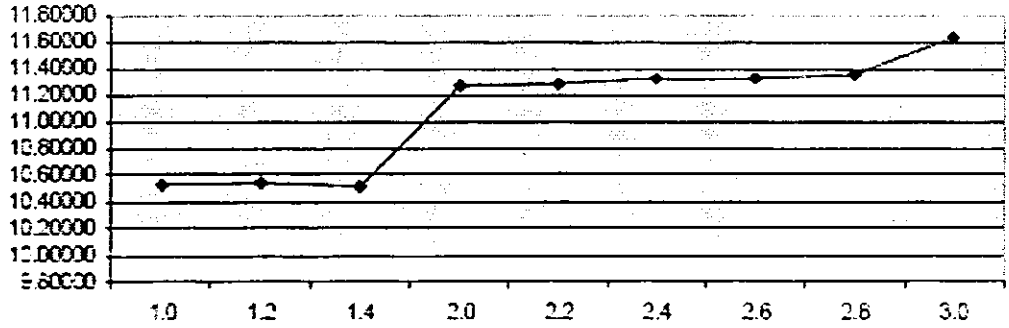


Figure 39: The McCabe's cyclical complexity plotted against releases [Bonkoski 2007]

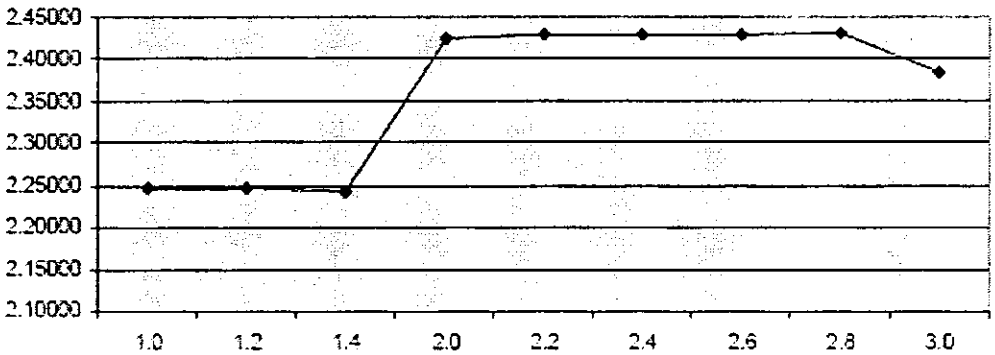


Figure 40: The Essential complexity plotted against releases [Bonkoski 2007]

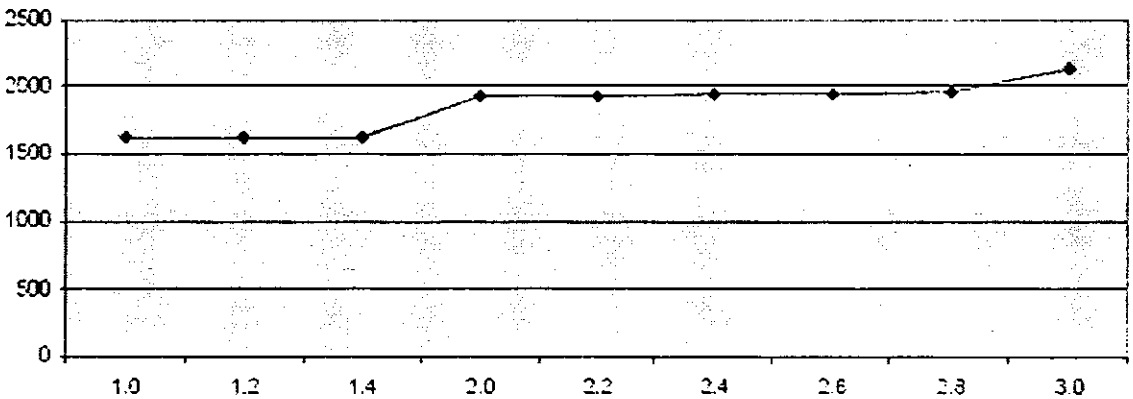


Figure 41: The NEST complexity plotted against releases [Bonkoski 2007]

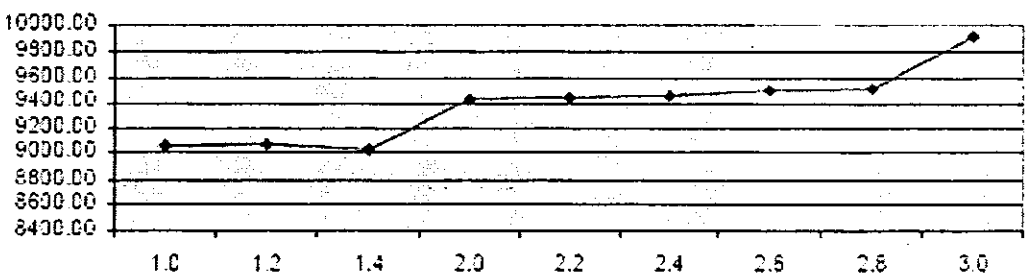


Figure 42: The Halstead complexity plotted against releases [Bonkoski 2007]

2.11 Growth, Complexity, and Quality of Firefox [Dong and Mohsen 2008]:

Dong and Mohsen found that the laws of evolution proposed by Lehman were basically for closed source applications, so they may not be valid for open source applications. Their assumption was on the basis of certain facts which include:

- When Lehman initially proposed these laws [Leh 78], then open source development was not popular, therefore the main focus of every type of study at that time was commercial in-house applications.
- Lehman proposed these laws on the basis of some case studies, which all are commercial applications, means none of which is an open source application.
- The statements used by Lehman clearly indicate that the type of projects, he was considering, were those controlled by strong managements. And those types of projects were not under his consideration in which management has less or no control on developers.

Moreover at the time of this study, some cases had been recorded in which open source evolution was found to be deviating from the patterns of Lehman, the most prominent of which was the study of Godfrey and Tu [Godfrey and Tu 2000]. In Godfrey's study, Linux had been found to be evolving at a rate different from Lehman. So the main question of this study was that either it is just the single case of Linux in which Lehman laws were disproved or other open source applications too show a deviating behavior

from Lehman's patterns. This study, not only tested the laws of Lehman on an open source application, but it also suggested a modified version of these laws on the basis of their own observations as well as observations of their pioneer studies especially by Godfrey. In that modified version of laws, they excluded some of the existing laws (of Lehman) and suggested some more laws on the basis of their own observations.

2.11.1 Software Applications:

The software application observed by Dong is a well known open source web browser, Firefox. It is a very commonly used web browser which have a large community of users all over the world. It is mainly written in three languages, C++, ANSI C and the assembly language. Its different releases and their source code are freely available on its website. This study covered its 4.5 years evolutionary history from 2004 to mid of 2008. There are two main types of releases of Firefox, trunk and branch. Most of the development activities are performed on trunk releases. When a trunk release is found ready to be delivered as a version, then its code is frozen and is given to different teams for quality assurance and bug fixing. That frozen version of code is identified as a branch. After a branch is released, the trunk is unfrozen to absorb more developments. For each major release of Firefox there are some associated alpha releases, beta releases, release candidates and dot-dot releases. Dot-dot releases are basically including security patches and updates.

2.11.2 Research Methodology:

As mentioned above, the source code of different versions of Firefox is freely available on its website, so the authors of this study got source code of different versions from that

website. Then they calculated different types of measures for each of those versions, including LOC and effort (measured in man-month). In order to count LOC, they used a tool SLOC-Count. The evolutionary period covered by this study is spread over 4.5 years starting from 2004 and ending at mid of 2008. Dong and Mohsen realized that observing the application as a whole is not enough, but one should also observe the components/subsystems separately to get a true picture of evolution. Considering the application as a whole and ignoring the behavior of its individual components may cause to mislead us. As an example they pointed out the difference of LOC between releases 2.0 and 3.0 of Firefox. There was a significant decrement in LOC of release 3.0 as compared to its predecessor release, release 2.0. But it was because that a major component "mailnews" was removed from release 3.0 considering it an extra component that should not be a part of web browser. Otherwise the size of all other component in release 3.0 was increased as compared to release 2.0. And if "mailnews" were not removed from release 3.0 then it surely would be having larger LOC than release 2.0. So the authors concluded that it will not be fair to say that Firefox lost its size from release 2.0 to release 3.0, because it was due to a specific reason. The true is that it grew in size. The authors also considered the architectural changes that happened to Firefox as it evolved. Moreover they focused on the languages used in development, that how the language choice of developers changed over time. They observed that which language was preferred by developers more and which did lose its favorability for developers.

2.11.3 Observations:

In order to observe the change and growth trends of Firefox, the LOC measure of different releases was plotted against time (years). It was observed that Firefox had been

continuously increased in size between 2004 and mid-2008. It was also observed that the rate of growth was larger in initial period as compared to the later period. Means it was observed that growth rate had been decreased over time. The graph of LOC versus time (years) has been shown by figure 43. There was only one incident of decrement in size as it evolved from release 2.0 to release 3.0, but it was due to removal of an un-necessary component from the application (Firefox), which in-fact can not be considered as an evidence of size decrement. The growth rate of each individual component was also observed, because authors realized that it was necessary to view the true picture of evolution. It was found that the core components of browser remained almost stable. The components which showed the continuous increment in size were security, DOM, JavaScript interpreter and accessibility. Other components, which include user-interface, networking and parsing etc, also remained stable. Moreover the architecture of browser also remained stable to a significant extent. Although changes occurred in it, but those were minor. No major changes observed in the architecture. As for as the languages used in development are concerned, the C++ was the most used language, however it lost its ratio in the later releases. Its one reason was the removal of the "mailnews" component from browser which was mainly written in C++ language. The second large language was the ANSI C and third was the assembly language. The ratio of assembly language was found to be increasing in the later releases.

2.11.4 Conclusions:

As mentioned above that Firefox was found to be continuously increasing in size, this indicated that Lehman's first and sixth laws proved in case of Firefox. Because according to these two laws, software systems continually change (first law) as well as

continually grow (sixth law). Moreover the growth rate was found to be decreasing with the passage of time, which also proved fifth law of Lehman, according to which growth rate of an evolving application tends to decline. Although fifth law was proved in this study but the authors didn't include it in their modified version of laws, because it had been disproved in the study of Linux by Godfrey and Tu [Godfrey and Tu 2000]. It was also concluded that the decrement in growth rate would be due to the fact that the complexity of application would have increased, so it would become difficult for developers to add more functions. Thus the decrement in growth rate was considered a consequence of increment in complexity, so it was concluded that complexity of application had increased, which verified the second law of Lehman, according to which complexity of software applications increases with time unless work is done to control it. Increment in complexity means the poor structure of program. It means that the application (Firefox) has lost its structure with its evolution. In other words it degraded structurally with time. Thus it lost its quality with time, which proved the seventh law of Lehman, which says that software applications lose their quality, as they evolve, unless we do efforts to control it. The work rate (effort) was measured in man-month and it was found that work rate was tending towards stability, which is an indication of the trend stated by Lehman as "conservation of organizational stability". Lehman quoted this trend as fourth law and stated that the effort spend on each release remains in-variant. In this study, Dong and Mohsen too observed that effort had tended to remain in-variant. However they realized that they had not sufficient evidence to say with confidence that this law has been proved in their case study, because some more observations were needed to get confidence about validity of this law.

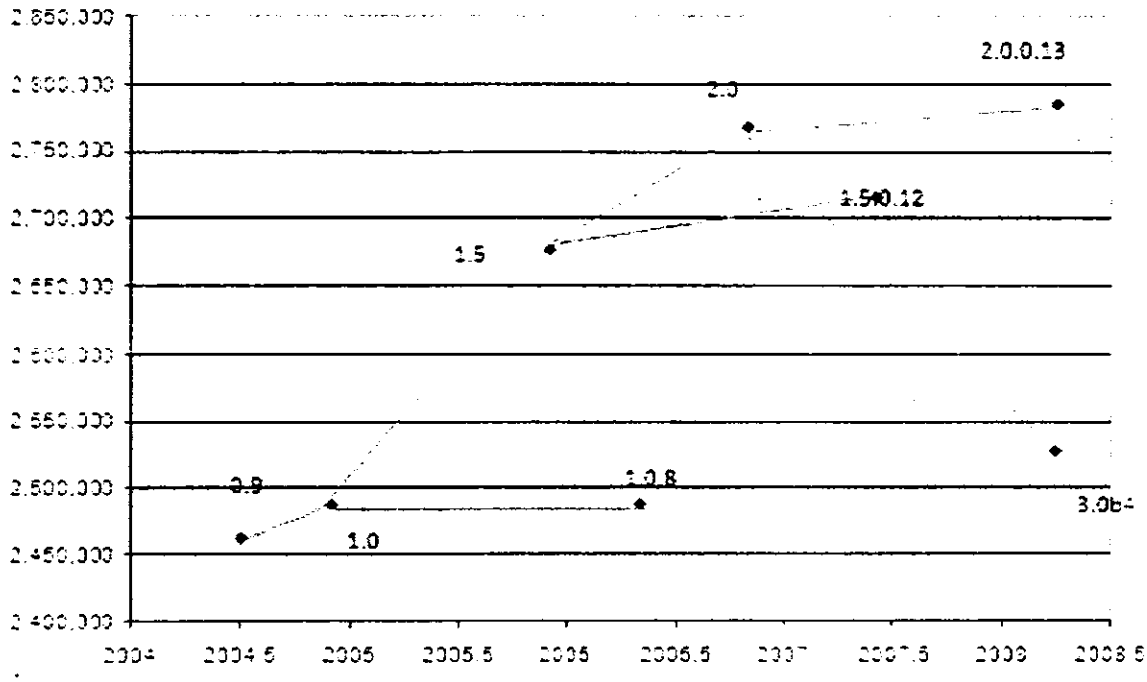


Figure.43: The LOC of releases plotted against time (years) [Dong and Mohsen 2008]

2.12 Growth and Change Trends of 8621 OSS [Koch 2005]:

Open source applications differ from commercially developed in-house applications in certain important aspects, which cause to change their processes from those of commercial applications. The basic difference is that open source applications are being developed by people who are not working under the control of a strong management which can push them to do something. They are not usually working for monetary gains, but their efforts are a result of their own interests. The open source development, being a relatively new area, is not as mature as the commercial development is. The software processes for commercial applications are more defined and tested as compared to open source applications. This is the reason why open source processes have got much attention of researchers in the past decade. It has been a favorite area of researchers to

define and test the open source development processes and to compare them with those of commercial application development. One of these software processes is the evolution process of open source software applications. Like other software processes, the evolutionary processes for commercial applications have been defined, validated and universally accepted. The evolutionary trends/laws have been defined by Lehman and others in different studies like, [Leh 78], [Leh 97a], [Leh 97b], [Leh and Ramil 2001] and [Turski 96]. These laws have been tested in many case studies and found to be verified in all cases. But the question for researchers, in the current decade, is that either these laws hold for open source applications or not. Open source evolution varies from that of commercial applications evolution in many aspects, like, in case of commercial applications, it is possible to mark clear distinction between the development and evolution process, which in case of open source development is usually not possible. Similarly open source applications are usually developed following the rule of "release early, release often", which is not suggested in case of commercial applications evolution. At the time of this study, some researchers had already tested the evolutionary trends of commercial applications (known as Lehman laws) on open source applications, most important of which studies, was that of Godfrey and Tu [Godfrey and Tu 2000]. In that study a large scale and widely used open source operating system "Linux" was studied to test the Lehman's laws of evolution. And it was observed that Linux was not following some of those laws. The present study is also a contribution in same type of work. It has tested Lehman's laws on a large collection of open source applications belonging to many different domains.

2.12.1 Software Applications:

Koch observed 8,621 software applications in this study. All of those applications were open source in nature. It is a very large number and no other study in this domain has observed too large set of applications. Moreover this set included all type of applications, large size, small size, successful, failed etc. Similarly, the applications included in this set did not belong to some specific domain. but belonged to many different domains. The total number of files, for the whole set of applications, were 2,474,175.

2.12.2 Research Methodology:

The source of applications, observed in this study, was sourceforge.net, a well known and most commonly used repository for open source applications. This repository was established to enable open source community (developers, researchers and users) to manage and control open source applications. It hosts thousands open source applications, from where users search and download applications, developers launch their own contributions as well as observe others contribution and researchers find the data about evolutionary and other characteristics of applications. This repository is especially helpful for the researchers who study the evolutionary trends of open source applications, because it enables them to get the timely (monthly, quarterly etc) snapshots of the code of applications hosted on it. For this purpose CVS or SVN clients are used, which download the timely snapshots of code from the repository. Koch, in this study, used the same methodology, means he downloaded the timely snapshots of code for all applications from sourceforge.net using a CVS client. He started from the start date of project/application and downloaded all monthly snapshots of code till the date of study. Then he calculated different measures for all those snapshots, including size (measured in

LOC), total number of files and number of programmers. Most of these measures were calculated using Perl scripts. And results were stored in a database for further analysis.

2.12.3 Observations:

It was observed that total 7,734,082 commits were made to the whole set of applications with 663,801,121 LOC added and 87,405,383 LOC deleted. The total number of programmers who contributed in the development was recorded 12,395. In order to observe the growth trends of these applications, two models were proposed, linear and quadratic. Both these models were proposed by taking application size (in LOC) as a function of time. Linear model was named as model A, whereas quadratic was named as model B. These models are given as:

Model A: $S(t) = a * t + b$

Model B: $S(t) = a * t2 + b * t + c$

According to the results of these models, all applications had grown over time. Then a statistical test was made to observe the long term tendency of growth rate and it was observed that growth rate had been declined for most of the applications. From the total sample, 61 % were found to be declining in their growth rate.

2.12.4 Conclusions:

As mentioned above, all applications included in the sample were found to be growing over time, which clearly proved the first and sixth laws of Lehman. According to first law, software applications must be continuously change, whereas according to sixth law, software applications must be continuously grow. Similarly a large number of applications (61 %) from the observed sample also proved fifth law of Lehman,

according to which growth rate declines over time due to increase in complexity. Koch also compared those projects which had declined in their growth rates with those which had shown an increasing growth rate. He found that the applications which had declined in growth rate were all large scale applications with a large number of programmers participating in them. From where they concluded that the large scale open source applications with a large community of developers don't decline in their growth rate and hence they disobey the fifth law of Lehman.

	[Godfrey2000]	[Robles et al 2005]	[Izurieta and Bieman 2006]	[Simmons et al 2006]
Software Application Studied	Linux Kernel	Linux Kernel, BSD family Kernels and 18 other applications	Linux, FreeBSD	Nethack
Nature of Application	An Operating System	Linux and BSD family are OS and 18 other belong to different types of domains	Both are Operating Systems	Game
Type of Application	Open Source	All are Open Source	Both are Open Source	Open Source
# of Versions / Evolution Period Covered	96 Releases	580 releases of Linux, monthly snapshots of all other applications over a period of four years or more	127 releases of Linux, 34 releases of FreeBSD	12 releases, Evolution period from mid 1980 t 2003
Measures Used	Total Release Size (Compressed Form), LOC, Uncommented LOC, Number of global Functions, Variables & Macros	Un-commented LOC for all applications, Release size in compressed form and number of files for just Linux	Total release size (in Kbytes), Total LOC, Number of directories, Number of different type of files, like, C files, C++ files, H (header files) etc Average and median LOC for C and H files	Total release size, Total LOC, Executable LOC, Commented LOC, Number of functions, Number of independent executio paths, Number of Operators and Operan
Tools Used	Unix Command "wc-l", An "awk" Script, Exuberant Ctags	SLOC-Count	UNIX "wc-l" command, Shell script	A commercial tool "understand f C"
Source of Release Information Used	Linux Kernel Archives Website	Linux Kernel Archives Website, Publically available repositories on Internet	Linux website for Linux releases, CVS repository of FreeBSD	Sourceforge.net
Characteristics Observed	Growth Rate	Growth Rate	Growth Rate	Growth Rate, Complexity
Findings/Observations	Linux grows at a super-linear rate	Linux grows at a super-linear rate, All other applications showed linear growth rate	Both Linux and FreeBSD have grown at a linear rate	Nethack has grown at a linear rat over a long time span (23 years nearly), The complexity of the Nethack with respect to the execution path has decreased, The complexity of Nethack with respect to number of operators an operands has increased
Conclusions	Lehman's Fourth and Fifth laws can't be validated by Linux	Lehman's Fourth law disproved	Lehman's fifth law holds true for open source applications	Lehman's fifth and second laws disproved

Table 1: Summarized description of the studies: [Godfrey2000], [Robles et al 2005], [Izurieta and Bieman 2006] and [Simmons et al 2006]

	[Capiluppi et al 2004]	[Xie et al 2009]	[Herraiz et al 2006]	[Ali and Maqbool 2009]
Software Application Studied	ARLA	Samba, Sendmail, BIND, OpenSSH, SQLite, VSFTPD, Quagga	Amaya, Evolution, Kaffe, Pre-Tools, Python, Wine, wxWidgets, XEmacs, XFree86, Linux, FreeBSD, OpenBSD, NetBSD	KTorrent, GNOME, Konversation, Evince
Nature of Application	Distributed File System	Samba (client-server interoperatability tool), Sendmail (mail transfer tool), BIND (DNS Server), OpenSSH (network security tool), SQLite (SQL engine), VSFTPD (File Transfer Protocol), Quagga (tool suit for router development)	First ten are major packages of Debian GNU/Linux, Last three are BSD family kernels (all are part of Operating System)	KTorrent (a data exchange tool), GNOME (a desktop for Linux), Konversation (an IRC client for KDE desktop environment), Evince (a document viewer)
Type of Application	Open Source	All Open Source	All Open Source	All four are small scale open source applications
# of Versions / Evolution Period Covered	62 Versions, released during a period of five years (Feb 1998 to Feb 2003)	Samba (89 versions), Sendmail (57 versions), BIND (168 versions), OpenSSH (78 versions), SQLite (172 versions), VSFTPD (60 versions), Quagga (29 versions)	Amaya (14), Evolution (11), Kaffe (46), Pre-Tools (122), Python (13), Wine (11), wxWidgets (29), XEmacs (15), XFree86 (12), Linux (533), FreeBSD (46), OpenBSD (12), NetBSD (43)	KTorrent (45 versions), GNOME (168 versions), Konversation (13 versions), Evince (46 versions)
Measures Used	LOC, SLOC, Total Size of source files (in KB),Total Number of source files,Total Number of source folders	LOC, Number of functions modified, Number of types modified, Number of global variables modified, Number of functions added, Number of types added, Number of global variables added, Number of functions deleted, Number of types deleted, Number of global variables deleted, Number of modules added	SLOC, Number of modules (source files)	Total number of modules, Modules difference, Number of modules added, Number of modules deleted, Number o modules modified, Total number of changes
Tools Used	UNIX utilities, "Xscc" awk script	CIL merger, ASTDiff, RSM	GlueTheos, SLOC-Count	Not Mentioned
Source of Release Information Used	Publically available CVS repositories	Public repositories	Linux versions were got from Linux website, Whereas the periodic snapshots of other twelve applications got from their CVS repositories	Not Mentioned
Characteristics Observed	Growth	Change, Complexity, Incremental Growth, Growth Rate, Quality	Growth	Growth, Change
Findings/Observations	All measures showed a continuous growth, All measures showed in-variant growth rate (except the number of source folders)	All applications have continuously changed and grown, Complexity of all applications have increased, Incremental growth showed ripples, Incremental growth neither decreased nor remained same	Six of the thirteen applications showed super-linear, four showed linear and three showed sub-linear growth trend, Both SLOC and number of modules showed same type of results	Both growth and change rates remainec constant in most cases
Conclusions	Lehman's fifth and sixth laws proved in case of ARLA	Lehman's first, second, third and sixth laws verified, Fourth, fifth, seventh and eighth laws were not confirmed or could not be confirmed	Lehman's fourth and fifth laws disobeyed	Lehman's fourth and fifth laws proved

Table 2: Summarized description of the studies: [Capiluppi et al 2004], [Xie et al 2009], [Herraiz et al 2006] and [Ali and Maqbool 2009]

	[Israeli and Feitelson 2009]	[Bonkoski 2007]	[Dong and Mohsen 2008]	[Koch 2005]
Software Application Studied	Linux	Nagios	Firefox	Total 8,621 applications studied
Nature of Application	Operating System	An administrator tool used to ensure systems availability	Web Browser	A large set of applications, which included all types of applications, small size, large size, successful, failed and belonging to different domains
Type of Application	Open Source	Open Source	Open Source	All Open Source
# of Versions / Evolution Period Covered	810 versions Released between March 1994 to August 2008	9 versions, 4.5 years evolutionary period	Covered evolution period from 2004 to mid 2008	From the start date of project till the time of study, a monthly snapshot o each application was observed
Measures Used	LOC, Number of modules, McCabe Complexity (MCC), Extended MCC (EMCC), Halstead Complexity, Release Increment in number of files, Number of modules handled (added, deleted, grown & shrunk), Release rate	SLOC, Number of functions, Number of downloads, McCabe's cyclic complexity, Essential complexity, NEST complexity, Halstead complexity	LOC, Effort (in man-month)	LOC, number of flies, Number of programmers
Tools Used	A commercial CASE tool as well as developed their own tool	Karakatau Metrics tool, Scripting language, Ms Excel	SLOC-Count	Perl script
Source of Release Information Used	Linux website	Sourceforge.net	Firefox website	Sourceforge.net
Characteristics Observed	Change, Complexity, Effort, Growth, Quality	Change, Growth, Complexity	Change, Growth, Effort, Complexity, Quality	Change, Growth
Findings/Observations	Linux has continually changed as well as continually grown, The overall code complexity has increased, The release increment tended to remain same on average, The work rate remained almost in-variant, The software familiarity issue was not entertained by many releases, The software quality was not declined with time	Nagios has continually grown with respect to SLOC and number of functions, The number of downloads have increased each year by a significant value, The all four complexity measures have increased with time	Firefox had continually grown (and hence changed), The growth rate decreased with time, Effort showed tendency towards in-variance, Complexity of application increased and quality decreased with time	All applications continuously changed and grown, The growth rat of large scale applications was not declined whereas in case of small scale applications, it was declined
Conclusions	Lehman's first, second, fourth, sixth and seventh laws were validated, Fifth law was not totally validated, Third law was validated but its validation requires more evidence	Lehman's first, second and sixth laws verified	Lehman's first, second, fifth, sixth and seventh laws proved in case of Firefox. Fourth law proved, but with in-sufficient evidence.	Lehman's first and sixth laws were verified in all type of applications whereas the fifth law was not verifi in case of large scale applications

Table 3: Summarized description of the studies: [Israeli and Feitelson 2009], [Bonkoski 2007], [Dong and Mohsen 2008] and [Koch 2005]

CHAPTER 3: RESEARCH METHODOLOGY

Introduction:

This chapter describes my research methodology in detail. The information given here can be used to repeat this work as well as to perform some other work of this type. It gives a detailed introduction of the techniques and tools which have been used for getting the older versions of my studied ERPs, the tools used for calculation of the different size related measures (like SLOC) for each of those versions, the tools used for preparation of results and finally the techniques used for analysis of different types of evolutionary trends.

3.1 Open Source ERPs Selection:

An Enterprise Resource Planning (ERP) is an automated system used to manage the entire business. It provides information about the important parts of a business, which information can be used to manage that business. It provides management information about products, suppliers, customers, stock, orders, manufacturing, sales, finance, and human resource etc. It helps and enables management to monitor the performance of a business with respect to its objectives.

At my proposal stage, I decided to choose the systems for my study from ERP domain. So my first task was the selection of open source ERPs on which I will test the Lehman laws of software evolution. My study will be the first one (in published literature) in which validity of Lehman laws will be tested on open source ERP systems.

For my study, I decided to select those ERPs which are among the most popular ones in user's community and which have higher ratings. Moreover I decided to choose those which are all developed using the same programming language, so that the size measures, like LOC and SLOC, can generate consistent results. On the basis of these factors, I selected the following three, Openbravo, Adempiere, and ApacheOFBiz. These three can be placed in the list of top five open source ERP systems [Linuxlinks 2011] Moreover all three have been developed using the same programming language, i.e. Java.

3.2 Getting history (old versions) of my studied ERPs:

I got past versions of my studied ERPs from their repositories which are available online. Not only my studied ERPs, but most of the open source applications are being maintained in the form of online repositories. These repositories are a best source to get history of these applications. These repositories are maintained with the help of version control systems. Once we create repository of application with the help of version control system, then version control system keeps record of all types of changes made to that application. Whenever someone makes any change to the application, the version control system creates a new version of that application, assigns it a unique id as well as stores date and time of that version. In this way the different versions of application are generated as soon as changes are made to it. These versions are also called, revisions,

snapshots or changesets etc. Version control systems allow us to view a list of all revisions along with their date, time and ids. In order to get any older revision of application, we just have to tell id of that revision to version control system. In this way we can get snapshot/revision of application on any given date and time.

There are many repository hosting servers, like sourceforge, and gitorious etc. These servers contain repositories of a large number of open source applications. The most prominent of which, is the sourceforge, this server hosts repositories of thousands of open source applications.

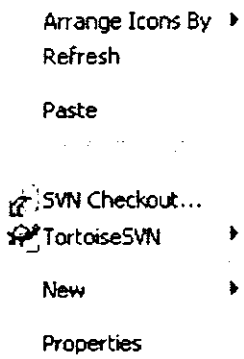
During my study, four different version control systems passed through my eyes, Concurrent Versioning System (CVS), Subversion (SVN), Mercurial (hg) and Git. But I have studied only two of them (SVN and hg) in detail, because repositories of my studied applications have been maintained using these two systems. Openbravo and Adempiere have been maintained using Mercurial (hg) and ApacheOFBiz has been maintained using Subversion (SVN).

3.2.1 Subversion (SVN) versus Mercurial (hg):

The difference between these two version control systems is that SVN is centralized version control system whereas hg is distributed version control system. That's why SVN didn't allow me to download whole repository of ApacheOFBiz to my local drive/PC. So in order to get many past snapshots, I have to download each snapshot separately. But hg, being a distributed version control system, allowed me to download whole repositories of Openbravo and Adempiere to my local PC. Once I got whole repositories of both ERPs to my local drive, I got required snapshots from those local copies of repositories.

3.2.2 Tools for SVN and hg:

There are many tools available for SVN, which are called SVN Clients. Some of those are, TortoiseSVN, RapidSVN, QSvn SmartSVN. Among those TortoiseSVN has more scores as compared to others. It is used for Windows operating system. Its commands once it is installed, begin to appear in the windows pop-up menus from where we can easily use them. When it is installed, the windows pop-up menu begins to look like:



SVN is also available as an API of UNIX. I used TortoiseSVN and SVN which is available as part of UNIX.

Mercurial (hg) is also available as part of UNIX. A GUI-based tool, called TortoiseHg, is also available, which, just like TortoiseSVN, used for windows and, after installation, begins to appear in the pop-up menus of windows. But I didn't use that GUI-based tool, I used the hg tool which is available as part of UNIX.

3.3 Tools Used:

The tools which I used during my study include, Cygwin, SLOCCount, TortioseSNV and MS Excel. From these, the TortoiseSVN, as described above, is SVN type version control system. And MS Excel was used to generate charts (graphs). The other two

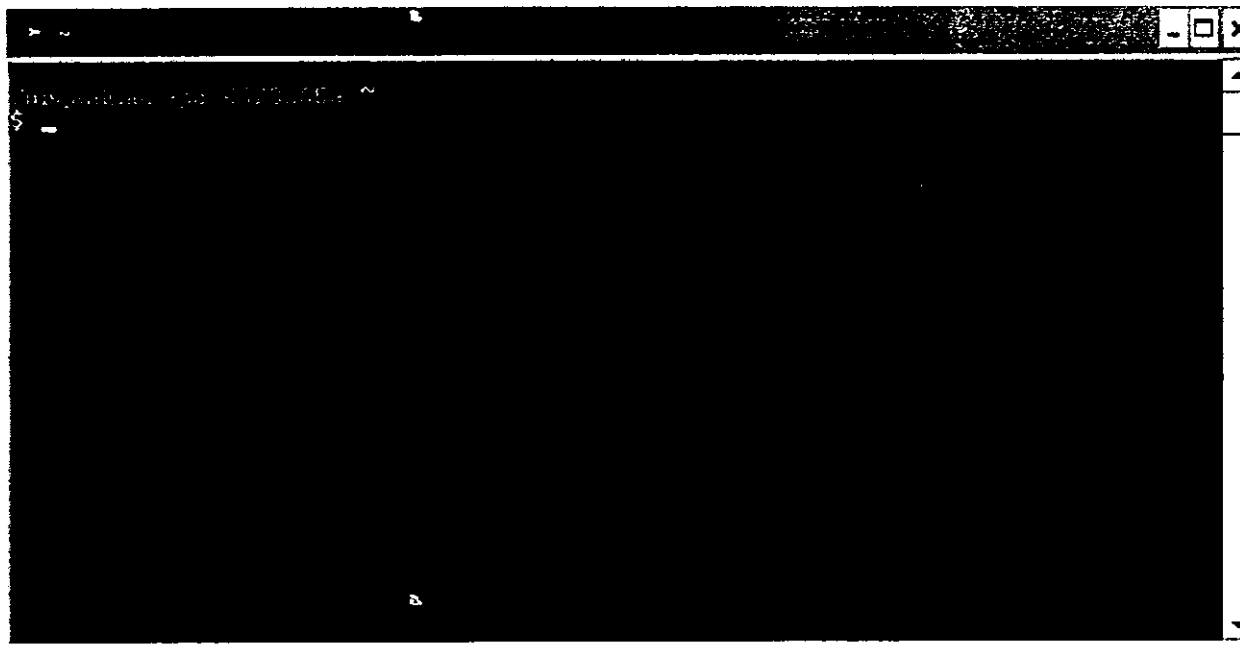
tools, Cygwin and SLOCCount are described here briefly with the perspective of my work.

3.3.1 Cygwin:

Cygwin is a tools which is used to enable/make-available UNIX APIs in Windows. If we want to use some of APIs of UNIX in Windows operating system or we want to run such application in Windows which require functionality from UNIX APIs, then we can install and use Cygwin. As mentioned above, I used SVN and hg tools which are available as part of UNIX, so I did it with the help of Cygwin. I was also needed some of UNIX APIs to run my second tool SLOCCount, which too, I made possible with the help of Cygwin.

I got Cygwin from its website www.cygwin.com where it is available free of cost. Cygwin is a very heavy tool which consumed much space (in GBs) on my secondary storage device as well as much time (in hours) for both downloading and installation. First I downloaded its setup to my local drive, which occupied 1.84 GB space. Then I installed it from that local setup and the size of installation folder was 6.45 GB. I did not change the default installation path and let setup to install Cygwin on default path which was "C:\Cygwin".

Cygwin is a prompt-based tool, which let us to type UNIX commands on command prompt and it runs those commands, because it contains UNIX APIs. In this way, it let us to "run UNIX" in the Windows operating system. The Cygwin interface looks like this:



During installation, Cygwin created a directory "home" on the installation path (means c:\Cygwin) and another directory with my user name "Furqan" (which I used to log on to windows) inside the "home" directory. Cygwin treats this directory as the default path.

3.3.2 SLOCCount:

I used this tool to calculate sizes of different releases of my studied ERPs. The measures which I selected for size calculation are "Source Lines of Code (SLOC)" and "Number of Source Code Files (Source Files)". This tool is developed by David A. Wheeler and, as its name indicates, is used to count lines of source code in a program/application. Its functionality is not limited to counting SLOC, but it can do much more. It also detects the language in which code is developed and categorizes count of SLOC by language. It can tell number of SLOC in each directory separately. It also tells the ratio of different languages used in the source code. I used it just for two types of calculations, one, for total lines of source code, and second, for total number of source code files.

I got this tool from http://www.dwheeler.com/sloccount/ where it is available free of cost. I downloaded it in zip form to my local directory. It was a very light file (just 188 KB) which consumed few seconds to download. The latest version which was available at the time of download was 2.26. Then I unzip that file, using the "winrar", to the home directory of Cygwin (c:\cygwin\home\furqan). Its user guide is available at http://www.dwheeler.com/sloccount/sloccount.html which is enough to learn all of its features.

3.4 Procedure & Commands:

My first task was to getting versions/revisions of the three ERPs, I had selected for my study. For this purpose, I started my work from Openbravo, and then moved to Adempiere and at the end moved to ApacheOFBiz.

3.4.1 Getting Versions of Openbravo:

The first task which I performed was downloading the Openbravo repository to my local drive. For this purpose, I used the clone command of Mercurial (hg). Because creating another copy of repository is called "cloning the repository", therefore this command is called the clone command. The command was as under:

hg clone http://openbravo.hg.sourceforge.net:8000/hgroot/openbravo/main/

/cygdrive/f/openbravo/head

The URL included in this command is the URL of Openbravo repository and the path "/cygdrive/f/openbravo/head" is the path of my local drive where repository is to be saved. This command downloaded the whole repository of Openbravo to my local drive

(at the path f:\openbravo\head). It took 2.5 hours nearly to download the whole repository. Now I was in the position to get any past snapshot/revision/changeset of Openbravo till the date when repository was first created. In order to view all the available revisions, I used following command:

hg log /cygdrive/f/openbravo/head/ >> /cygdrive/f/openbravo/revisions.doc

This command generated a list of all available revisions and saved this list to the file f:\openbravo\revisions.doc. I opened this file in the WordPad and the list of all revisions was there along with their dates, times and revision ids. Now I filtered revisions using the monthly gaps. I downloaded repository to my local drive on 23 Jun, 2011. So the top revision (also called the head revision) was made on 23 Jun, 2011. Before that I selected the revision made on 23 May, 2011, before that 23 Apr, 2011, before that 23 Mar, 2011 and so on. In this way I moved to the date of first revision. The earliest revision which I selected was made on 23 Nov, 2007. Thus I selected all monthly revisions between Nov, 2007 and Jun, 2011. This resulted in total 44 revisions. The file of filtered revisions is given in the Appendix. Here is a snapshot of very small part of that file:

```
changeset:   12953:b9eb624349da
tag:         tip
user:        RM packaging bot <staff.rm@openbravo.com>
date:        Thu Jun 23 23:39:44 2011 +0200
summary:     CI: update AD_MODULE to version 12952

changeset:   12357:a707bb7eade4
user:        Egoitz Castillo <egoitz.castillo@openbravo.com>
date:        Mon May 23 17:08:51 2011 +0200
summary:     Fixed Issue 17168. Error when clicking on reference link button

changeset:   11876:1fc4dc82c09f
user:        Stefan HÃ¼hner <stefan.huehner@openbravo.com>
date:        Wed Apr 20 13:09:38 2011 +0200
summary:     [lib-update] Fix deprecation warnings after log4j update

changeset:   11507:8b7e48a804a9
parent:      11506:9ed0ec8660cc
parent:      11344:103820d16f01
user:        Valery Lezhebokov <valery.lezhebokov@gmail.com>
date:        Wed Mar 23 21:54:04 2011 +0100
summary:     Merge with PI
```

Here each changeset/revision is given an id, like, revision of Jun 23, 2011 has id 12953 and May 23, 2011 has 12357 etc. In my first command, which I used to clone the online repository to my local hard drive, I didn't mention any revision. Due to which it cloned the latest revision of repository. In order to clone any of older revisions, I added the revision id in that command. For example, in order to clone the revision made on 23 May, 2011 (having id 12357), I used following command:

hg clone /cygdrive/f/openbravo/head/ /cygdrive/f/openbravo/12357 -r 12357

Here I didn't use the repository URL, but I used my local hard drive path (i.e. f:\openbravo\head), because now I had cloned repository to my local drive, from where I could generate past revisions, so I had no need to use online repository. The numeric value 12357, written after "-r", told that the revision having id 12357 was to be cloned. So this command cloned the revision "12357" to the mentioned path (i.e.

f:\openbravo\12357). And as shown in the revisions file snapshot, this revision (12357) was made on May 23, 2011. Thus after this command, I had got the one month older revision of repository. Now I used this command repeatedly and got older and older revisions of repository till the first revision (made on Nov 2007). Each revision of repository was containing the application's (Openbravo) snapshot on the given date as well as the information of past changes so that older revisions can be generated. But, for my analysis, I was needed past snapshots of the applications code just, not the information of past changes, so I deleted the folder named .hg from all these revisions, because the information of past changes was stored in that folder. And it was not part of the applications code. After deleting this folder from all repository revisions, the monthly snapshots/versions of Openbravo were ready from Nov 2007 to Jun 2011.

3.4.2 Getting Versions of Adempiere:

The procedure, I followed for Adempiere was same as of Openbravo. The first command which cloned its online repository to my local drive was as under:

hg clone http://adempiere.hg.sourceforge.net:8000/hgroot/adempiere/adempiere/ /cygdrive/f/adempiere/head

This command cloned the latest revision of online repository to the following path of my local drive "f:\adempiere\head". It took nearly five hours to download the repository. I used this command on 23 Jun, 2011. This means it downloaded the repository's snapshot of 23 Jun, 2011. Now by using the "hg log" command, I got the list of all available revisions of repository. Then I filtered that list and selected the revisions with one month gap. Means before Jun, 2011, I selected the revision made on May, 2011, then the revision made on Apr, 2011 and so on. I moved to the month, when the repository was

first time created. The earliest revision which I selected was made on Nov 2006. In this way I selected total 56 revisions. Now I noted the revision ids of all the filtered revisions and cloned all those revisions from the local copy of repository (which I had got by my first command). For cloning the revision made on May 2011 I used following command:

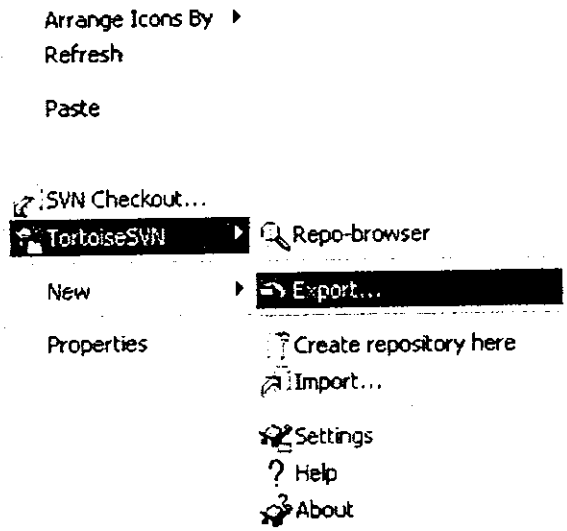
hg clone /cygdrive/f/Adempiere/head/ /cygdrive/f/Adempiere/6678 -r 6678

Here 6678 is the revision id which I selected from the month May 2011. This command cloned the "6678" revision of repository to the path "f:\adempiere\6678". This is just one example, and following the same pattern I cloned all 56 revisions of repository. And in the last step, I deleted the folder named .hg from all these revisions, because this folder just contains the changes history (which is used to generate older revisions) and it is not part of applications code. So after deleting this folder from all revisions, I had got the application's (Adempiere's) monthly snapshots/versions from Nov 2006 to Jun 2011.

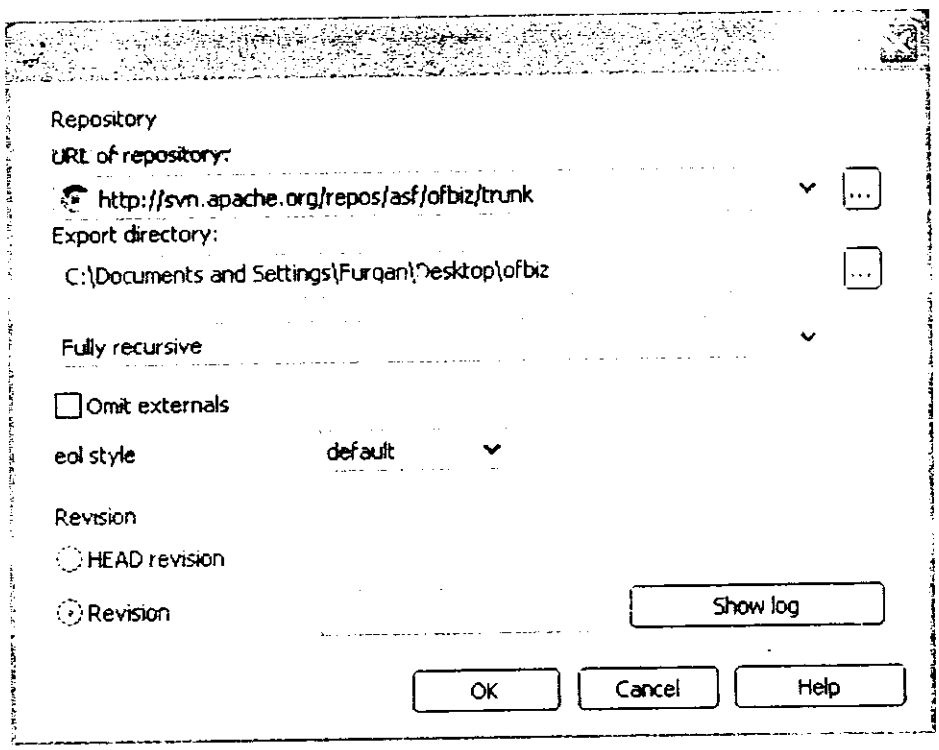
3.4.3 Getting Versions of ApacheOFBiz:

The procedure for preparing revisions of ApacheOFBiz was somehow different from those of Openbravo and Adempiere. Its because, as I have already described, the repository of ApacheOFBiz has been maintained using the SVN instead of hg. For SVN, I used the GUI based tool TortoiseSVN. Its commands, after installation, were available to me in the windows pop-up menu. SVN doesn't allow to get revisions of repository (called cloning), but it just allows to get revision of application (called export or checkout). So it was not possible now to download the latest revision of repository to my local drive and then get the older revisions from that local repository. But now I had to download each revision of application (ApacheOFBiz) from online repository, separately. In order to view a list of all available revisions of application, I just right-clicked on my

desktop and selected the option TortoiseSVN → Export from the pop-up menu, as shown:

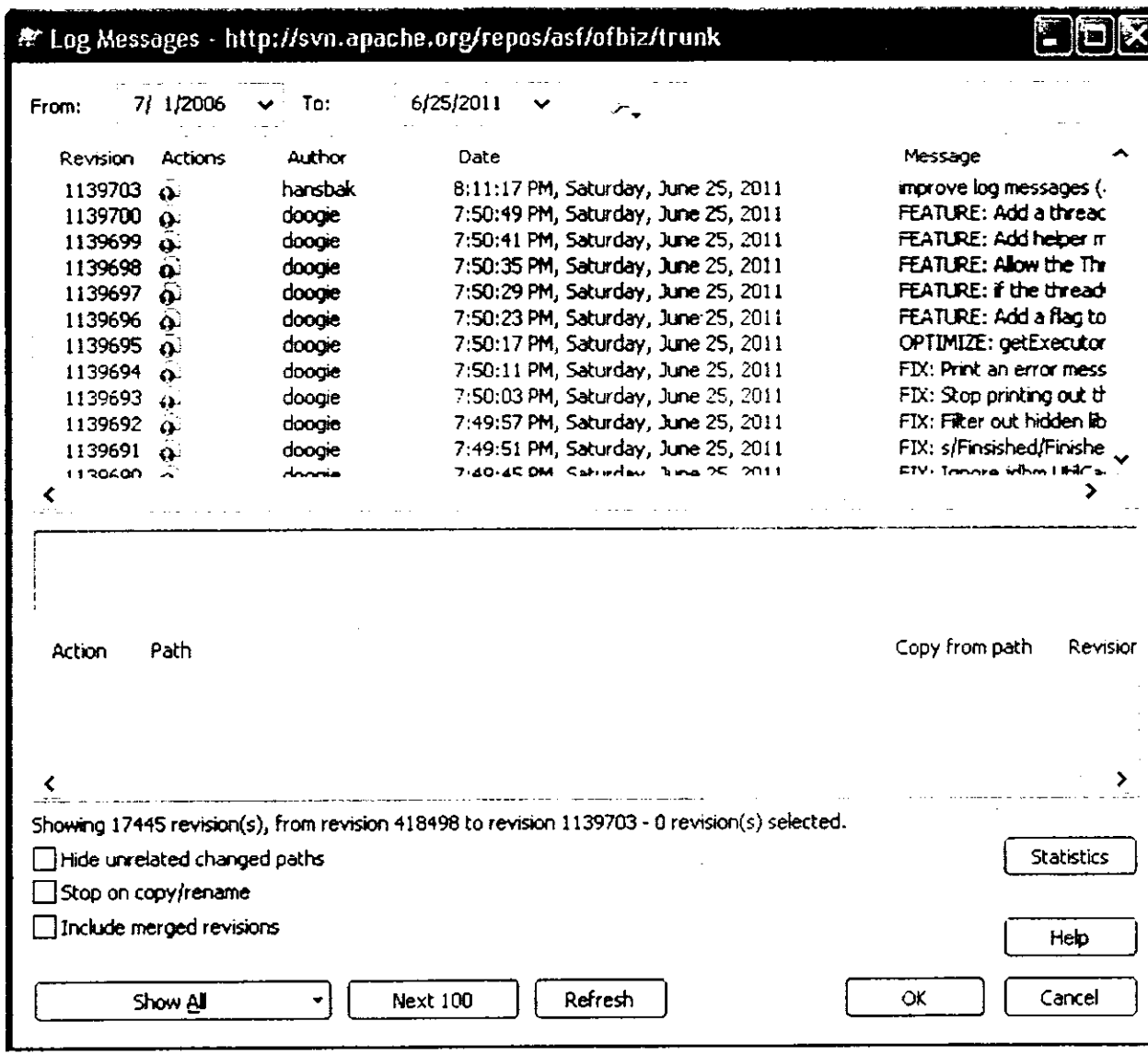


It opened the following dialog box:



Here I entered the URL of ApacheOFBiz repository and clicked the "show log" button.

The result appeared as the following dialog box:



It showed the top (latest) 100 revisions, but when I clicked the "show all" button, then it showed all available revisions. The latest revision, also known as the head revision, was made on 25 Jun, 2011 and its id is 1139703, as can be clearly seen in the snap. Here, once again, I filtered the revisions on the basis of monthly gaps. Before the revision 1139703, I selected the revision 1127662 which was made on May 2011 and before that 1096457, which was made on Apr 2011 and so on. I moved to month of first available revision, which was July 2006. In this way, I selected total 60 revisions for my study. I

noted ids of all of my selected revisions and downloaded (exported) them, one by one, from online repository to my local drive. For each revision, I created a folder, right clicked that folder, selected the option TortoiseSVN → Export, then entered the id of that revision in the proper text box and finally clicked the OK button.

3.4.4 Setting the SLOCCount Tool:

Now it was the stage of calculating size measures for all of the revisions of the three ERPs so that their evolutionary trends may be observed. I used, as already mentioned, the two size measures, SLOC and source code files. For this purpose, I used the tool SLOCCount. This tool could be used in UNIX operating system because it was required some of the APIs of UNIX. So I installed Cygwin for this purpose. Cygwin, as described above, provided me UNIX (its prompt along with its APIs) in Windows. I saved the SLOCCount folder (named SLOCCount-2.26) in the home directory of Cygwin (which was c:\cygwin\home\furqan). Before using SLOCCount, I had to make some settings, first of them was adding some text to its file named as "makefile". I opened this file in the WordPad and added the text ".exe" at the end of its 35th line. After addition of text, the line began to look like:

EXE_SUFFIX=.exe

After addition of this text, I opened the SLOCCount folder on Cygwin prompt, using the following command:

cd /cygdrive/c/cygwin/home/furqan/sloccount-2.26

Then I executed the two Cygwin commands, "install" and "make install". The purpose of these commands was to make possible to run SLOCCount from any path without moving to its directory. Now after these settings, the SLOCCount tool was ready to be used.

3.4.5 Calculating the SLOC:

Now I started to calculate SLOC for different revisions of my ERPs. I had total 160 revisions of the three ERPs (44 of Openbravo, 56 of Adempiere and 60 of ApacheOFBiz). So I executed the command of SLOC calculation for 160 times. Also I stored results of each command in a separate file. Thus after execution of these commands, I had got 160 results files. Here is given just one of those commands along with the snapshot of its result file. I used this command to calculate SLOC for 12357 revision of Openbravo:

sloccount /cygdrive/f/openbravo/12357/ >> /cygdrive/f/openbravo_results/L12357.doc

The first path in this command (/cygdrive/f/openbravo/12357/) is the folder where I had saved the revision and second path (/cygdrive/f/openbravo_results/L12357.doc) is the path and name of file in which command results were to be saved. The snapshot of this result file is as under:

```
Have a non-directory at the top, so creating directory top_dir
Adding /cygdrive/d/openbravo_revs/12357//CONTRIBUTORS to top_dir
Adding /cygdrive/d/openbravo_revs/12357//README to top_dir
Creating filelist for WebContent
Adding /cygdrive/d/openbravo_revs/12357//build.xml to top_dir
Creating filelist for config
Adding /cygdrive/d/openbravo_revs/12357//create.database.launch to top_dir
Adding /cygdrive/d/openbravo_revs/12357//eclipse.compile.complete.launch to top_dir
Adding /cygdrive/d/openbravo_revs/12357//eclipse.compile.launch to top_dir
Adding /cygdrive/d/openbravo_revs/12357//eclipse.install.source.launch to top_dir
Adding /cygdrive/d/openbravo_revs/12357//export.database.launch to top_dir
Creating filelist for legal
Creating filelist for lib
Adding /cygdrive/d/openbravo_revs/12357//log4j.lcf to top_dir
Creating filelist for modules
Creating filelist for referencedata
Creating filelist for src-core
Creating filelist for src-db
Creating filelist for src-gen
Creating filelist for src-test
Creating filelist for src-trl
Creating filelist for src-util
Creating filelist for src-wad
Adding /cygdrive/d/openbravo_revs/12357//update.database.launch to top_dir
Creating filelist for web
Have a non-directory at the top, so creating directory src_top_dir
Adding /cygdrive/d/openbravo_revs/12357//src/build.xml to src_top_dir
Adding /cygdrive/d/openbravo_revs/12357//src/buildAD.xml to src_top_dir
Adding /cygdrive/d/openbravo_revs/12357//src/index.jsp to src_top_dir
Adding /cygdrive/d/openbravo_revs/12357//src/log4j.lcf to src_top_dir
Adding /cygdrive/d/openbravo_revs/12357//src/log4j.properties to src_top_dir
Creating filelist for src_org
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.

SLOC    Directory     SLOC-by-Language (Sorted)
119102  src_org       java=119102
38992   modules       java=38983,sh=9
11109   src-core      java=11109
9177    src-wad       java=9177
7335    src-test      java=7335
3255    web           php=3063,perl=191,sh=1
968     src-util      java=925,sh=43
782     src-trl       java=782
215     src-db        java=215
136     src_top_dir   jsp=136
0       WebContent    (none)
0       config        (none)
0       legal         (none)
0       lib           (none)
0       referencedata (none)
0       src-gen       (none)
0       top_dir       (none)

Totals grouped by language (dominant language first):
java:        187628 (98.20%)
php:           3063 (1.60%)
perl:           191 (0.10%)
jsp:            136 (0.07%)
sh:              53 (0.03%)

Total Physical Source Lines of Code (SLOC)                = 191,071
Development Effort Estimate, Person-Years (Person-Months) = 49.69 (596.30)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.36 (28.35)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 21.03
Total Estimated Cost to Develop                           = $ 6,712,683
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

3.4.6 Calculating the Source Code Files:

After SLOC, I had to count number of source code files for each of the 160 revisions. For this purpose, I added the "filecount" parameter in the above mentioned sloccount command. Like, to calculate number of source code files in the Openbravo's 12357 revision, I used following command:

sloccount -- filecount /cygdrive/f/openbravo/12357/ >>

/cygdrive/f/openbravo_results/F12357.doc

This command calculated the number of source code files in the Openbravo's 12357 revision and saved the results to the file "f:\openbravo_results\F12357.doc". I used this command for each of the 160 revisions of my studied ERPs and stored results in separate files. Thus after completion of this process, I had got 160 more results files. In this way, I prepared 320 results files in total.

3.4.7 Loading Data & Preparing Charts:

The last step of my work was to load the results into MS Excel worksheets, so that I can generate graphs/charts required for my analysis. So I created a separate workbook (Excel file) for each of three ERPs, loaded data into those workbooks and generated charts. The data was consisting of three attributes, Revision Month, SLOC and Number of Source Files. I got this data from those results files which I had prepared in the previous step using the SLOCCount commands. These data sheets and the charts, I created from their data, are given in the next chapter.

CHAPTER 4: RESULTS

Introduction:

In this chapter, I have described the results of my case studies in detail. The different charts/graphs which I drew on the basis of different size related measures, have been shown here. Each of the charts has been explained in the perspective of the evolutionary trend indicated by that chart. At each step, the evolutionary trends are also compared with the laws of Lehman. At the end, the results are summarized and are compared with the results of past studies of this area.

4.1 Measures used in my Study:

I have tested the three laws of Lehman, first, fifth and the sixth. All these laws are related to the growth trends of the software applications, so I used the size related measures, SLOC and number of source code files. I calculated these measures for each monthly revision/snapshot of my studied applications and then plotted each of those measures against the revision month. I used the same type of plots, as were used by Lehman in his studies, when he formulated these three laws [Leh 78], [Leh 97b], [Leh 98a]. Those plots were:

1. Total size of release (measured in number of modules) plotted against the release numbers.
2. Incremental growth of each release plotted against the release numbers.

   Where incremental growth of release "x" was calculated as:

   Size of release "x" – size of release "x -1"

I used same plots, but I used two size measures, rather than only one. In this way, I repeated my observations twice, which resulted in strengthen of my conclusions. One of the measures used by me was same as of used by Lehman, i.e. "number of source code files", which maps exactly to the Lehman's measure "number of modules". However the second measure, SLOC, was not used by Lehman, but I repeated my work using this measure too because many well known studies in this area have used this measure, like [Godfrey and Tu 2000], [Robles et al 2005], [Simmons et al 2006]. The results of these measures for my studied applications can be seen in Tables 4.1 (for Openbravo), 4.2 (for Adempiere) and 4.3 (for ApacheOFBiz).

4.2 Graphs/Plots used in my Study:

I have used following four types of graphs for each of my studied applications:

1. Total size of revision (measured in source code files) plotted against the month of revision.
2. Total size of revision (measured in SLOC) plotted against the month of revision.
3. Incremental growth of revision (number of source code files added in that revision as compared to its predecessor revision) plotted against the month of revision.

4. Incremental growth of revision (SLOC added in that revision as compared to its predecessor revision) plotted against the month of revision.

The first two graphs are same except that different size measures are used. Similarly third and fourth graphs are also same in nature, but are based on different size measures. The first type of graph, i.e. total size of revision plotted against the revision month, helped me in testing of first and sixth laws as well as partial testing of the fifth law. This graph enabled me to observe that either my studied applications have continually changed and grown or not. Similarly, after addition of the reflection of linear growth in this graph, it enabled me to observe the trend of the growth rates of applications, means either their growth rate increased or decreased or remained same.

The second type of graph helped me in testing of the second part of the fifth law, according to which, software applications grow at a fixed rate (although this rate varies/decreases with the passage of time). The actual phenomenon in this law is that, unusual growth is not possible, and, if it occurs then the subsequent period must compensate this unusual growth. It is because an unusual growth diminishes/destroys familiarity of stakeholders with the software, so a negative feedback generates, which decreases the growth rate in the subsequent period. The reason of increased/unusual growth is the fact that growth is compulsory for software systems. Stakeholders of software need and desire growth, but when it increases from a specific amount, it diminishes familiarity and hence produces resistance to growth.

4.3 My Observations:

Let's look at the evolutionary trends of my studied ERPs, Openbravo, Adempiere and ApacheOFBiz.

4.3.1 Openbravo:

The size of application (plotted against time) showed clearly upward trend. The graph was found to be moving upward without any ambiguity, as can be seen in figures 44 and 45. The first figure (means 44) was drawn using the size measure SLOC whereas the second (means 4.2) was drawn using the measure "number of source code files". Both of the measures produced same type of results. The application has continuously grown and hence has continuously changed. This verified the first and sixth laws of Lehman. The "continuous growth" is in accordance with the sixth law and the "continuous change" is with the first law.

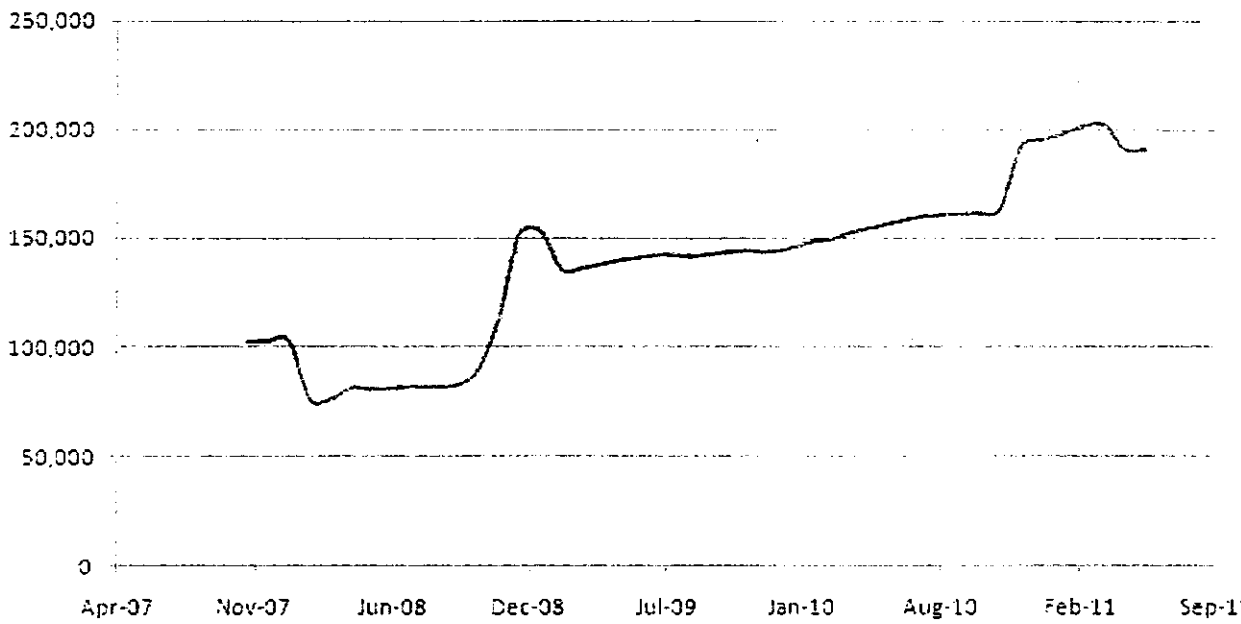


Figure 44: Openbravo: Revision size (in SLOC) against revision month (time).

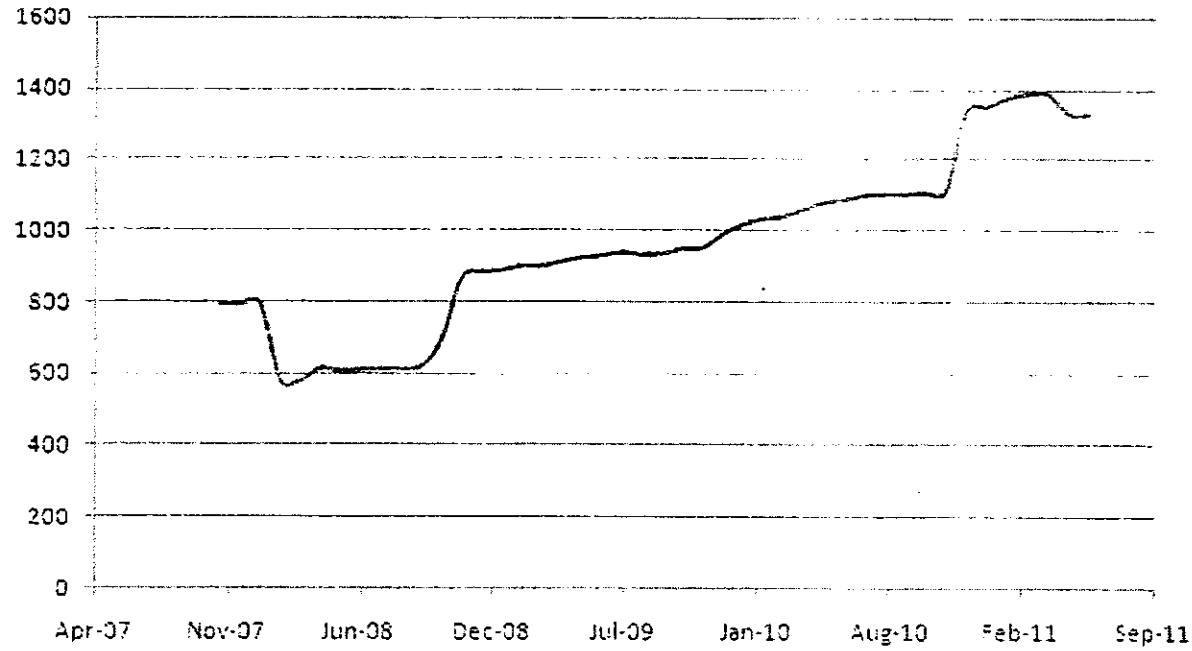


Figure 45: Openbravo: Revision size (in Source code files) against revision month (time).

In order to observe the trend of growth rate, I added the projection of linear growth in the actual growth graphs. Then by comparing the actual growth with the linear growth, I determined the trend of growth rate. These graphs can be seen in figures 46 and 47. In case of SLOC (figure 46), the actual growth has moved along with the linear growth by making positive and negative ripples with it. It has crossed the linear growth many times, which indicates that it (actual growth) is itself almost linear. Linear growth means that the application has grown at an in-variant rate. Or in other words, the growth rate has been remained same. However, in case of source files (figure 47), the actual growth has travelled below the linear growth for most of the time. It has crossed the linear growth just once (near Feb 2011). In the past period, from Nov 2007 to Feb 2011, the actual growth was moving below the linear growth. It is an indication of increasing growth rate, means that the growth rate was slower in the initial period, but it increased with the passage of time. Thus we can say that the growth rate of application (Openbravo) has

either remained in-variant (in case of SLOC) or increased (in case of source files). None of the measure has shown decrement in growth rate. This trend is not in accordance with the fifth law of Lehman. According to the fifth law, the growth rate decreases with the passage of time, but Openbravo has grown at an in-variant or even increasing rate.

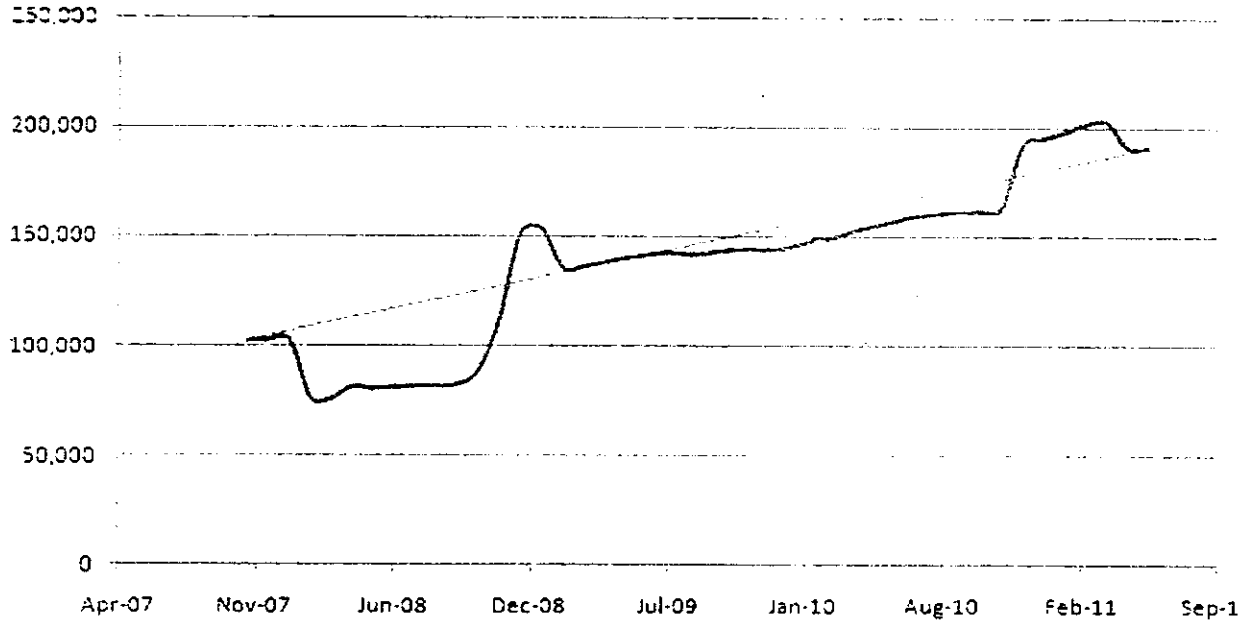


Figure 46: Openbravo: Revision size (in SLOC) against time (compared with linear growth).

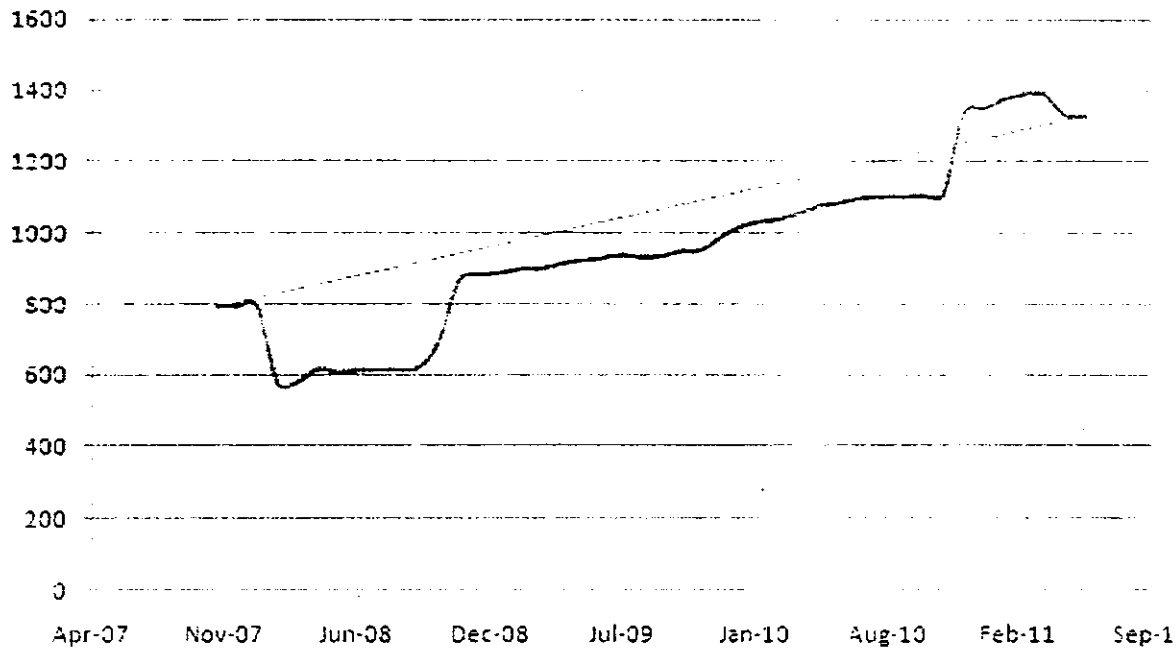


Figure 47: Openbravo: Revision size (in Source files) against time (compared with linear growth).

Now, I plotted the incremental growth against time, that is, the growth made in each revision against the revision month. The growth of a revision is equal to the difference between the sizes of that revision and its predecessor revision. For example:

Incremental Growth of Jun 2011 Revision = Size of Jun 2011 Revision - Size of May 2011 Revision

It must be noted that incremental growth can be negative, because it is possible that a revision may be smaller in size as compared to its predecessor revision. But, because the overall trend is increment in size therefore such examples (with negative incremental growth) are very few. The graph also included the average incremental growth. The purpose was to compare the actual incremental growth with the average incremental growth. These graphs are shown by figures 48 and 49. Figure 48 shows the graph drawn using the SLOC measure, whereas, the figure 49 shows the graph drawn using the source

files measure. Both the graphs have shown same type of trends. That is, the actual incremental growth has been found to be making cycles around the average incremental growth with positive and negative ripples. Larger positive ripples have been leaded by larger negative ripples to balance the overall growth. Theses ripples, according to the fifth law of Lehman, indicates the presence of feedback effect, according to which, a larger incremental growth (positive ripple) diminishes familiarity of stakeholders and as a result negative feedback (resistance to growth) generates, which causes a smaller incremental growth (negative ripple). Hence, Openbravo has proved the Lehman's "conservation of familiarity" concept.

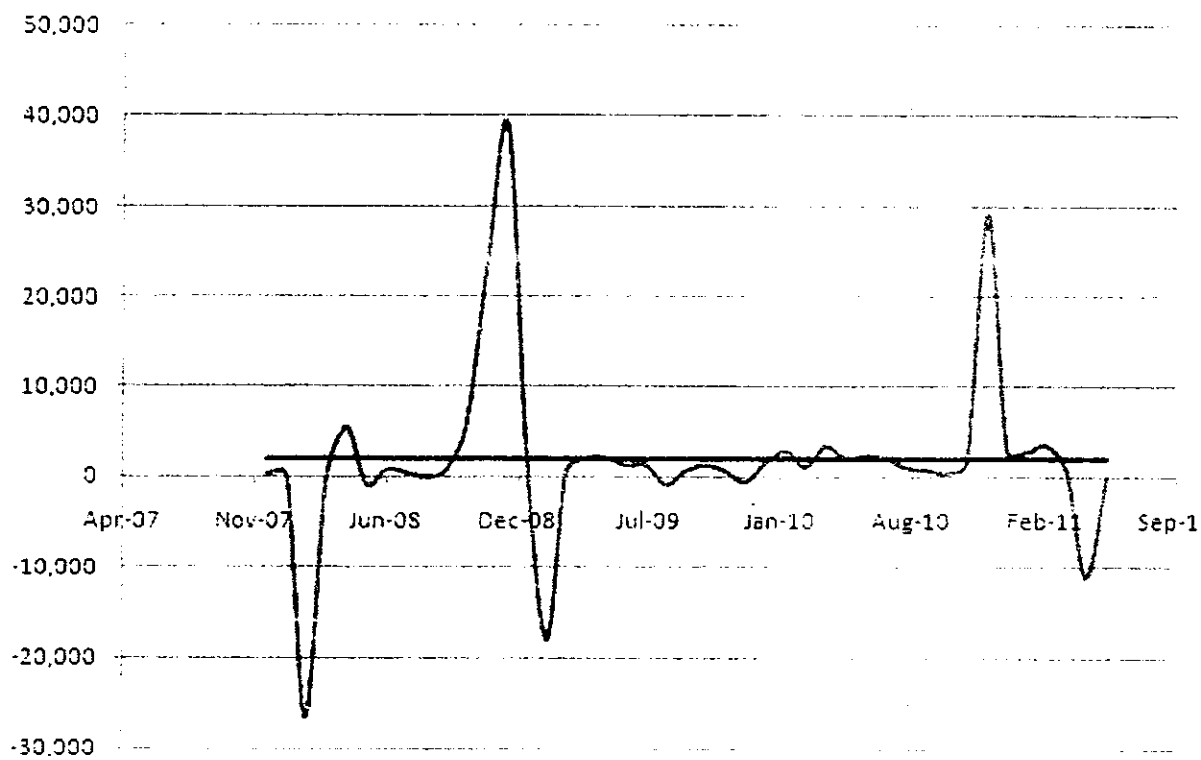


Figure 48: Openbravo: Revisions incremental growth (in SLOC) against revision month (time).

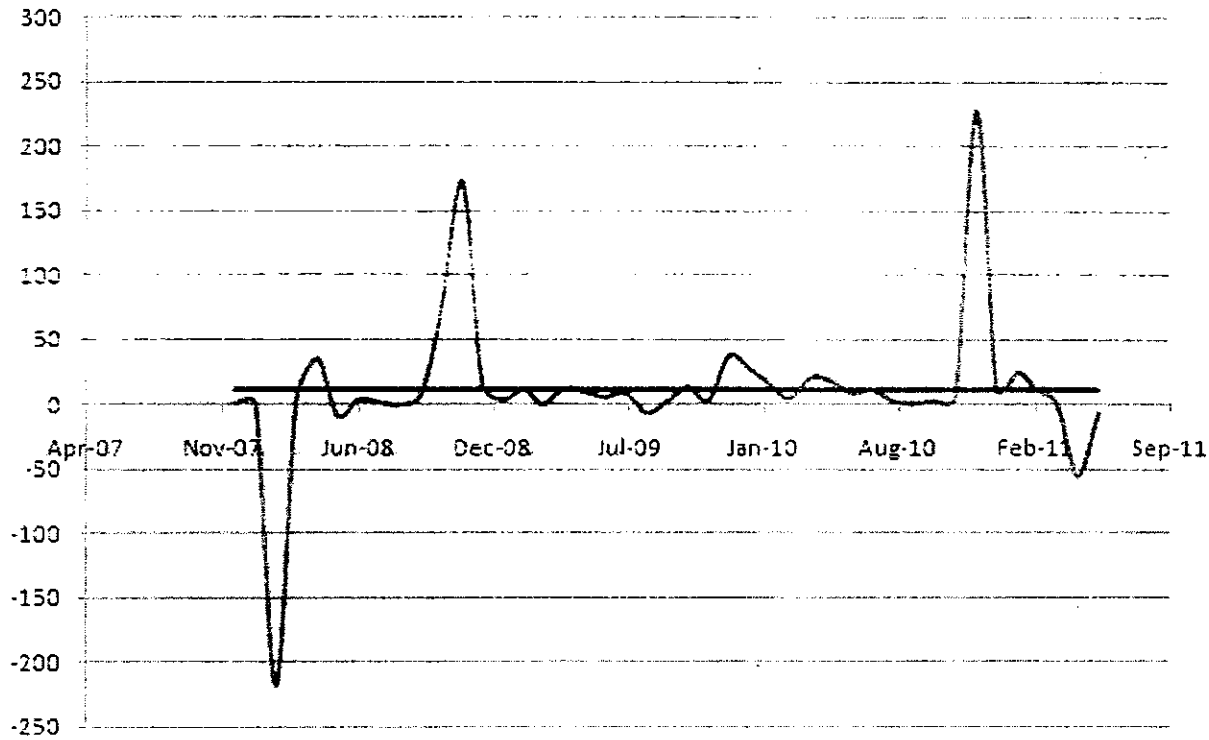


Figure 49: Openbravo: Revisions Incremental growth (in Source Files) against time.

4.3.2 Adempiere:

Adempiere too, like Openbravo, showed continuous growth and change trends. Here, once again, both of the size graphs (one for SLOC and second for source files) were found to be moving upward, thus showing continuous growth and hence continuous change. The graphs can be seen in figures 50 and 51. The first graph (figure50) has plotted revisions SLOC against its month and the second (figure 51) has plotted revisions source files against its month. In both cases, application (Adempiere) has been found to be continually increasing in size, thus verifying the first (continuous change) and sixth (continuous growth) laws of Lehman.

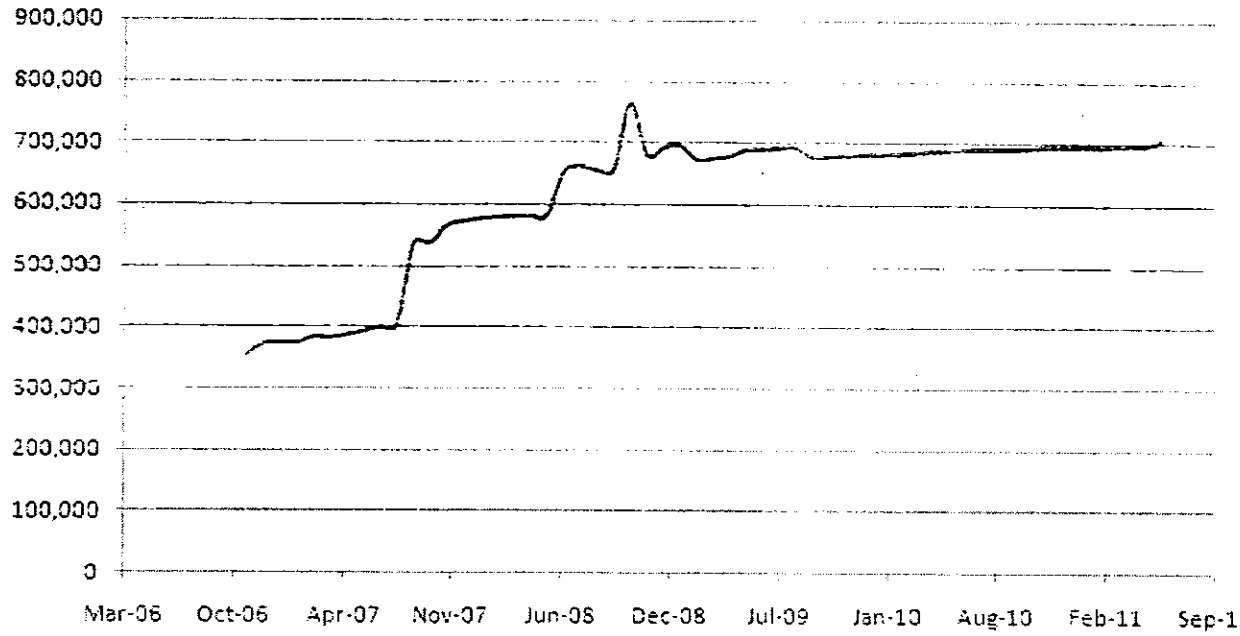


Figure 50: Adempiere: Revision size (in SLOC) against revision month (time).

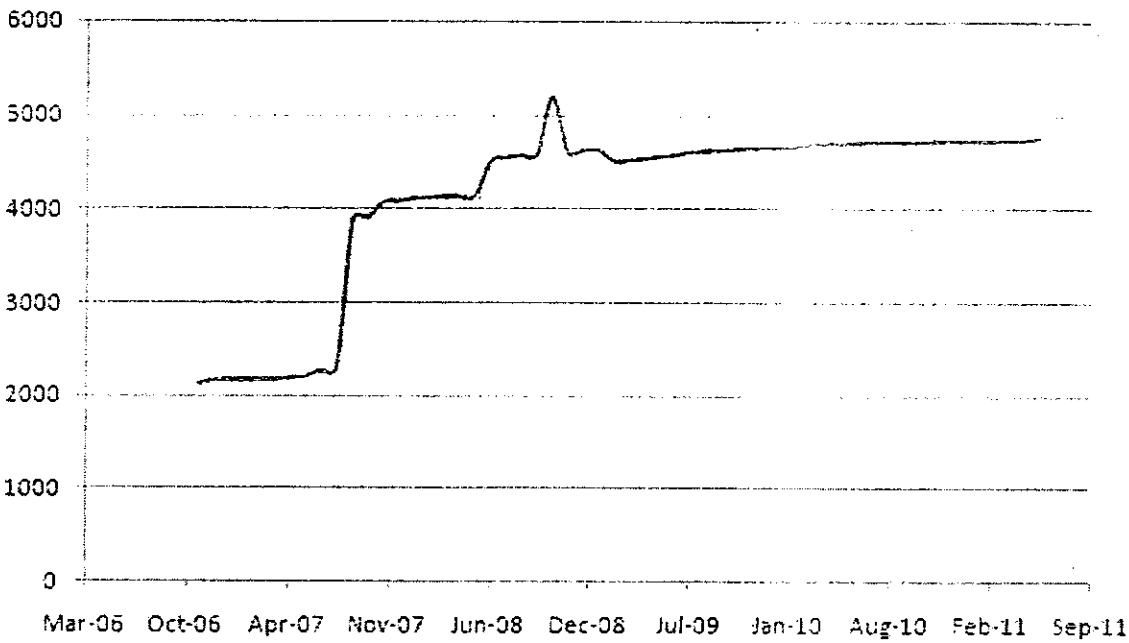


Figure 51: Adempiere: Revision size (in Source code files) against revision month (time).

In order to determine the trend of growth rate, the actual growth was compared with the linear growth. The graphs are shown by figures 52 and 53. Figure 52 shows the growth rate trend for SLOC measure whereas figure53 shows growth rate trend for source files

measures. The results of both measures have strong resemblance with each other and hence can be said same. In both cases (SLOC and source files), the actual growth has travelled above the linear growth, which shows that growth rate has been decreased with time. The existence of actual growth above than the linear growth means that growth was fast in the initial period but it slowed down with the passage of time. Thus the growth rate of Adempiere was declined (decreased) with the passage of time. It is in accordance with the fifth law of Lehman.

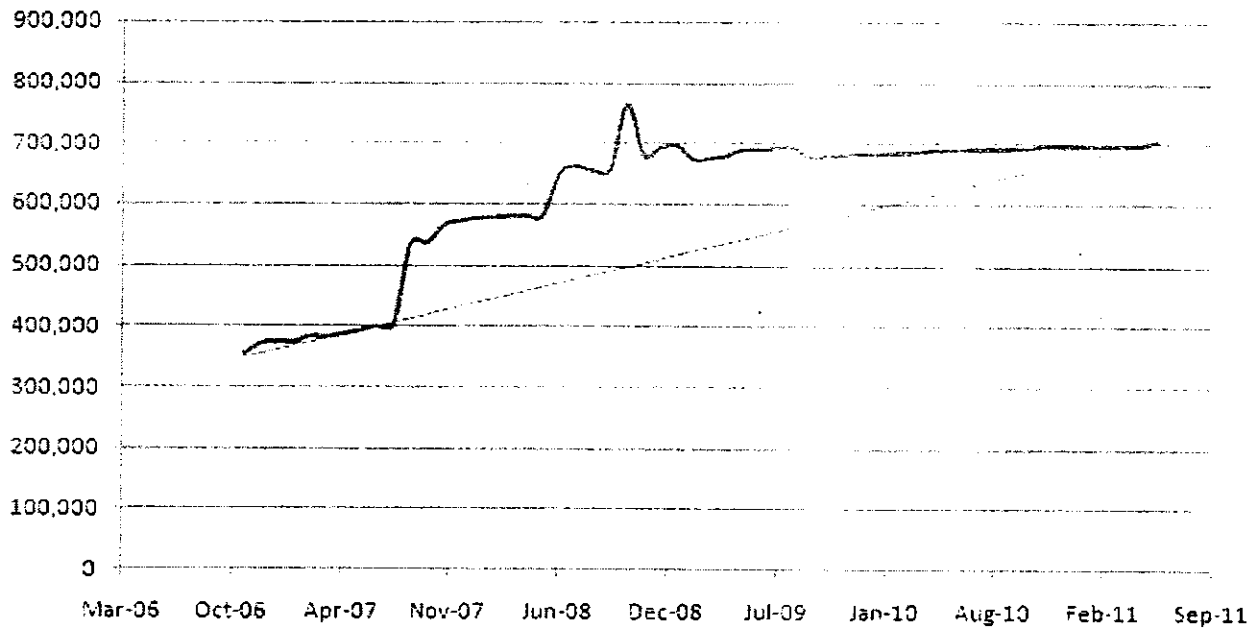


Figure52: Adempiere: Revision size (in SLOC) against time (compared with linear growth).

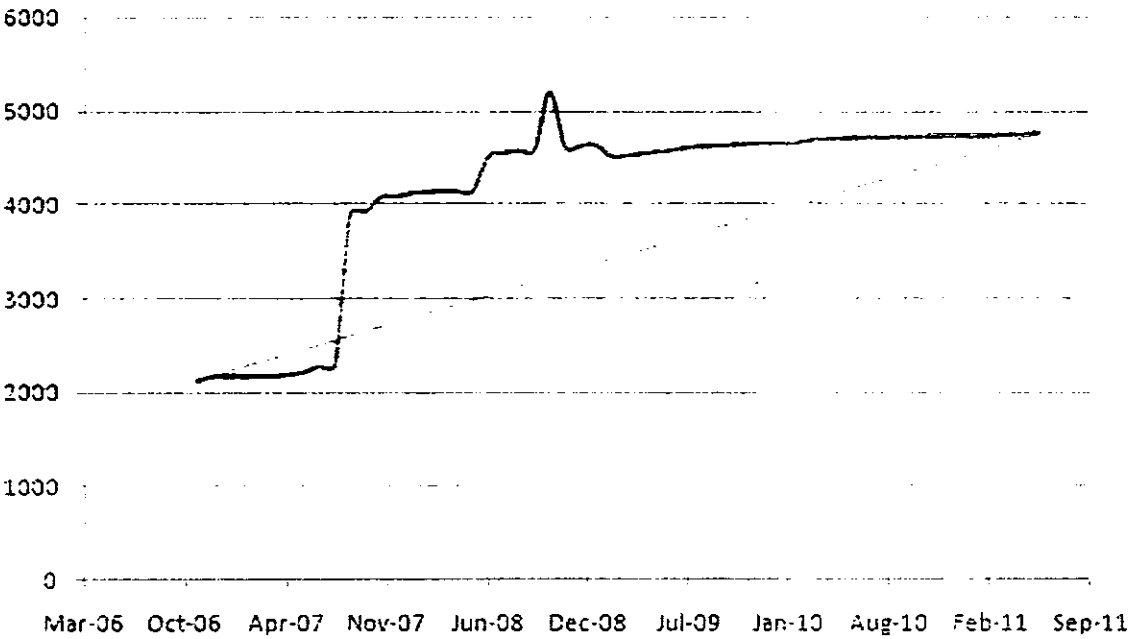


Figure 53: Adempiere: Revision size (in Source files) against time (compared with linear growth).

To determine the existence of "conservation of familiarity" concept, the incremental growth was plotted (against time) and compared with the average incremental growth. The graphs can be seen in figures 54 (for SLOC) and 55 (for source files). Both graphs have verified the existence of the "conservation of familiarity" concept. The incremental growth has been remained invariant during the last half period of evolution, from Jul 2009 to Jun 2011. During this period, the incremental growth has been remained nearly equal to average (as can be seen in both figures 54 and 55). However in the first half period, the incremental growth has been found to be varying and here cycles can be seen. Means, in this period, the actual incremental growth has made cycles around the average incremental growth, which is, as explained above, an indication of the existence of the "conservation of familiarity" concept. In this way, Adempiere has proved the Lehman's fifth law.

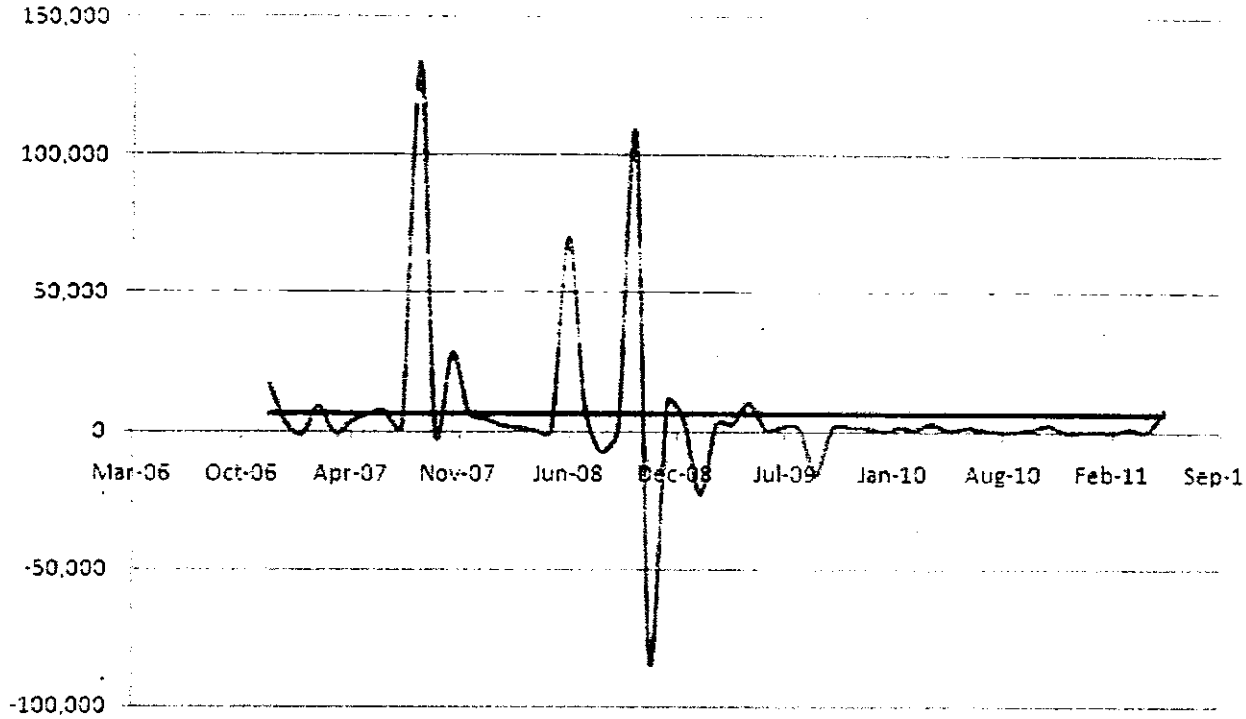


Figure54: Adempiere: Revisions incremental growth (in SLOC) against revision month (time).

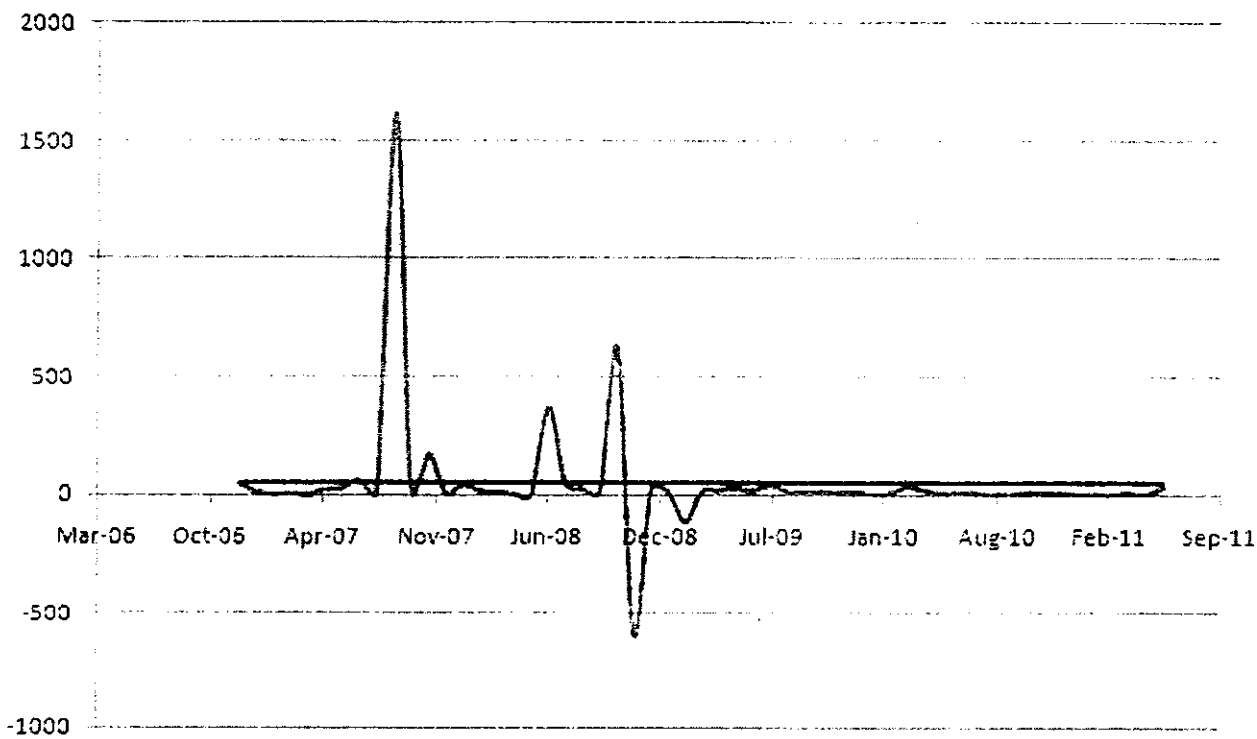


Figure 55: Adempiere: Revisions Incremental growth (in Source Files) against time.

4.3.3 ApacheOFBiz:

ApacheOFBiz, like Openbravo and Adempiere, has also shown continuous increase in size and hence has proved both of the Lehman laws, first and sixth. The graphs for the two measures SLOC and source files can be seen in figures 56 and 57 respectively. Both graphs have a continuous movement in the upward direction, which shows continuous growth and hence continuous change trends. In this way, this application (ApacheOFBiz) also aligns very well with the first and sixth laws of Lehman.

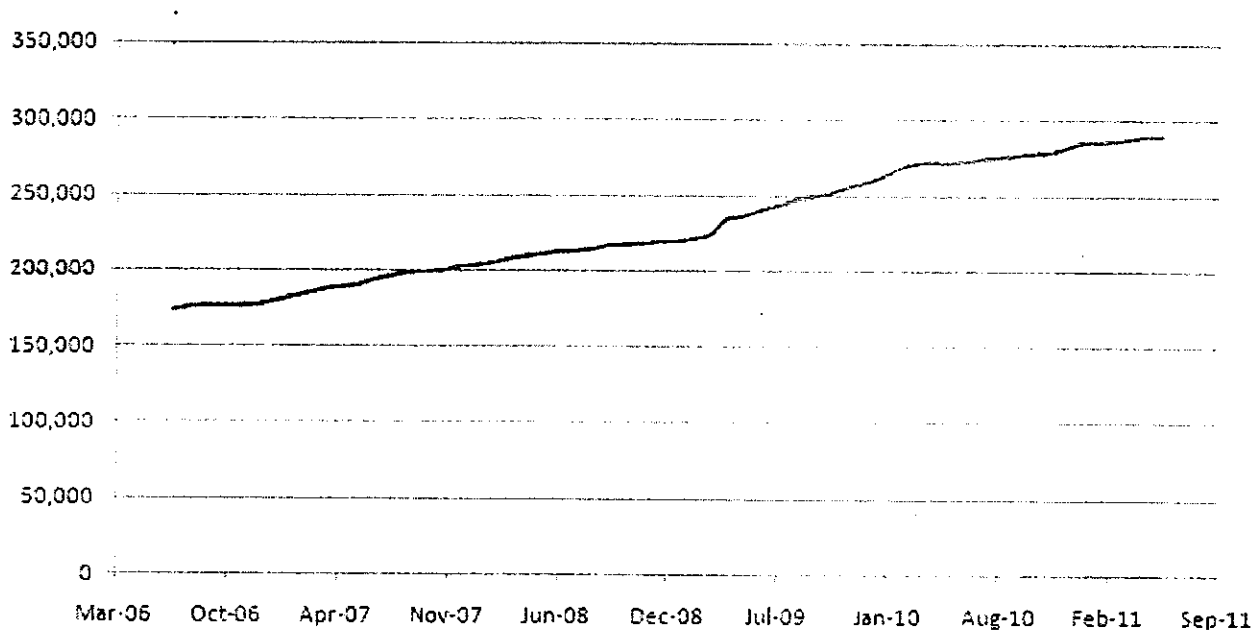


Figure 56: ApacheOFBiz: Revision size (in SLOC) against revision month (time).

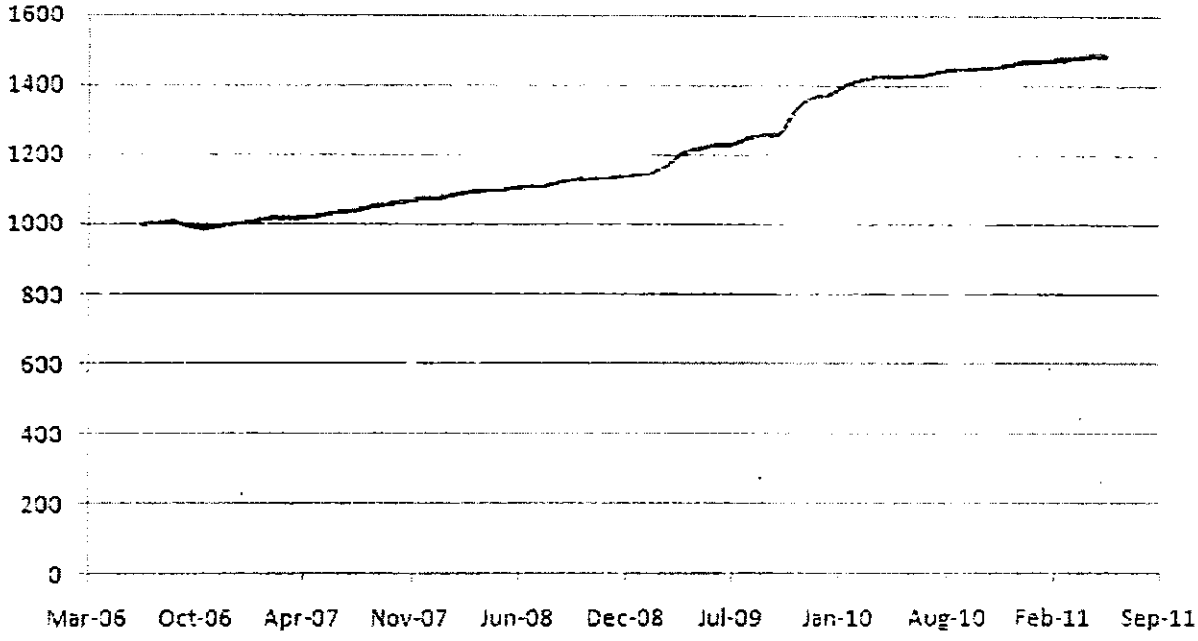


Figure 57: ApacheOFBiz: Revision size (in Source code files) against revision month (time).

The growth rate of ApacheOFBiz has been found nearly stable. If we compare its growth graph with the linear growth, as shown by figures 58 and 59, the actual growth can be observed (in case of SLOC measure) to be travelling around the linear growth with small cycles, which shows that the growth rate has remained nearly invariant. In case of source files measure (figure 59), the actual growth has travelled below the linear growth till Jan 2010, but it has travelled very close to the linear growth. However, it can be said that the growth rate has increased (although by a slight amount) in case of source files measure. Thus the growth rate of ApacheOFBiz has either increased or remained invariant. There is no case of decreasing growth rate. So the ApacheOFBiz has not proved the fifth law of Lehman, according to which growth rate declines (decreases) with time.

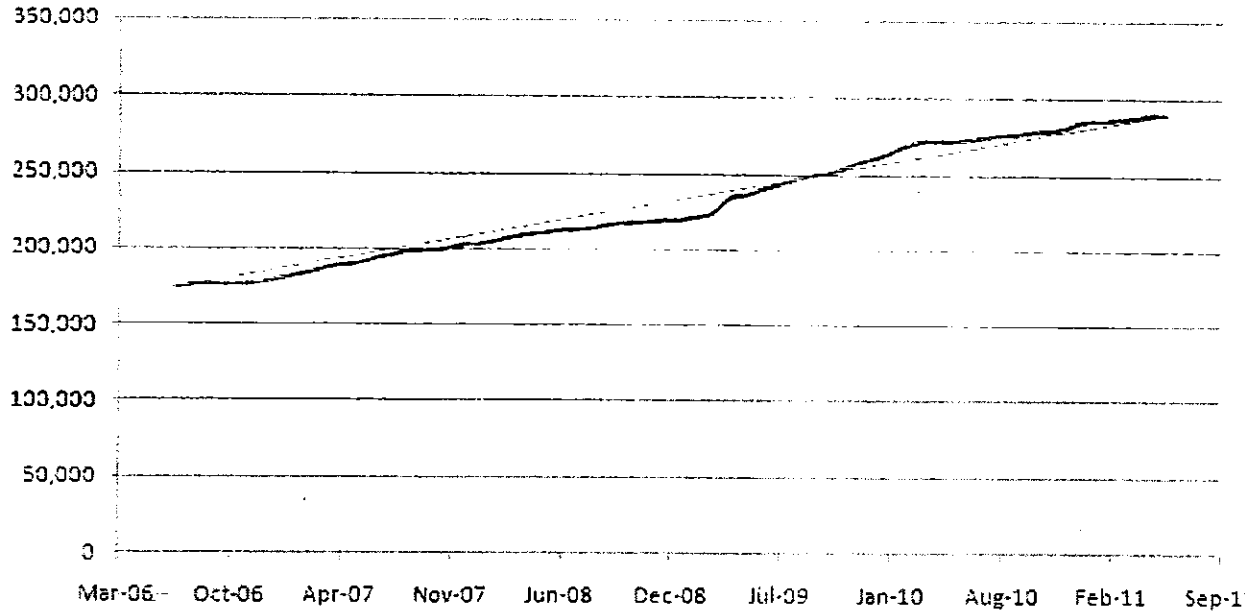


Figure 58: ApacheOFBiz: Revision size (in SLOC) against time (compared with linear growth).

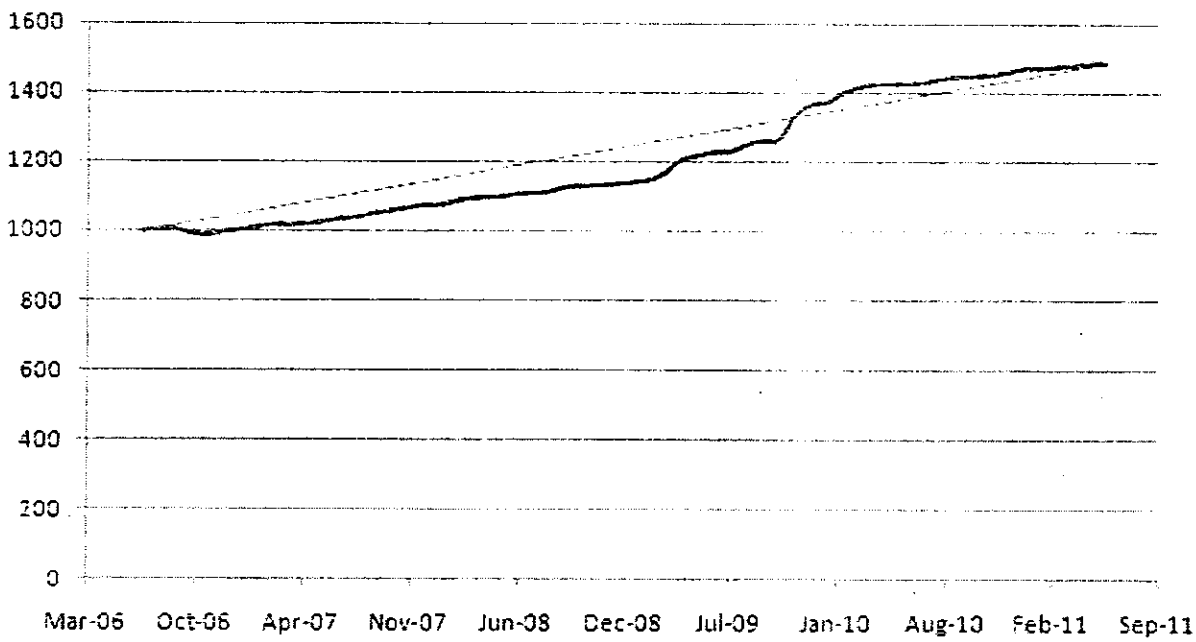


Figure 59: ApacheOFBiz: Revision size (in Source files) against time (compared with linear growth).

In the last, the incremental growth of ApacheOFBiz was plotted against time. It also, like the other two applications (Openbravo and Adempiere), proved the existence of the "conservation of familiarity" concept. The actual incremental growth formed regular cycles around the average incremental growth, just like that found by Lehman in his case studies of FW logica plc system, ICL VME kernel and the Lucent Tech real time system [Leh 98b]. The graphs are shown by figures 60 and 61. These cycles proved that larger incremental growth wasn't absorbable for the concerned (stakeholders) of ApacheOFBiz, so revisions with larger growth were followed by revisions with smaller growth to conserve/maintain the familiarity of stakeholders with the application (ApacheOFBiz).

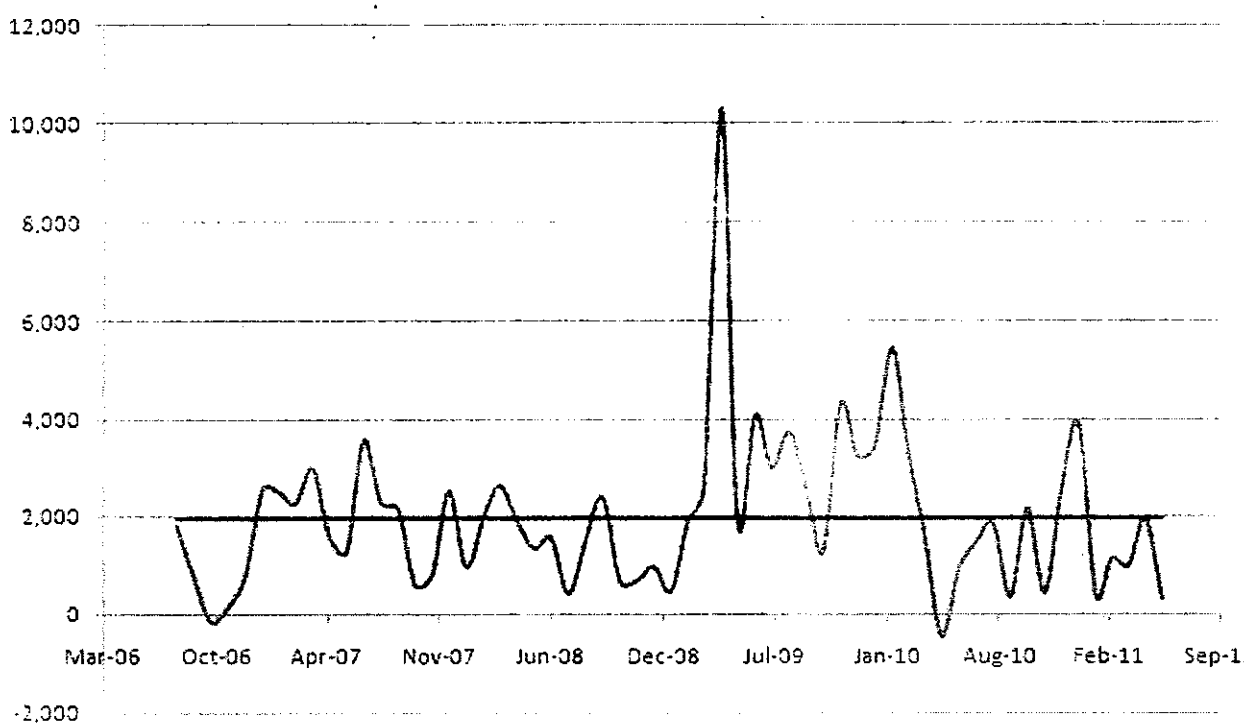


Figure 60: ApacheOFBiz: Revisions incremental growth (in SLOC) against revision month (time).

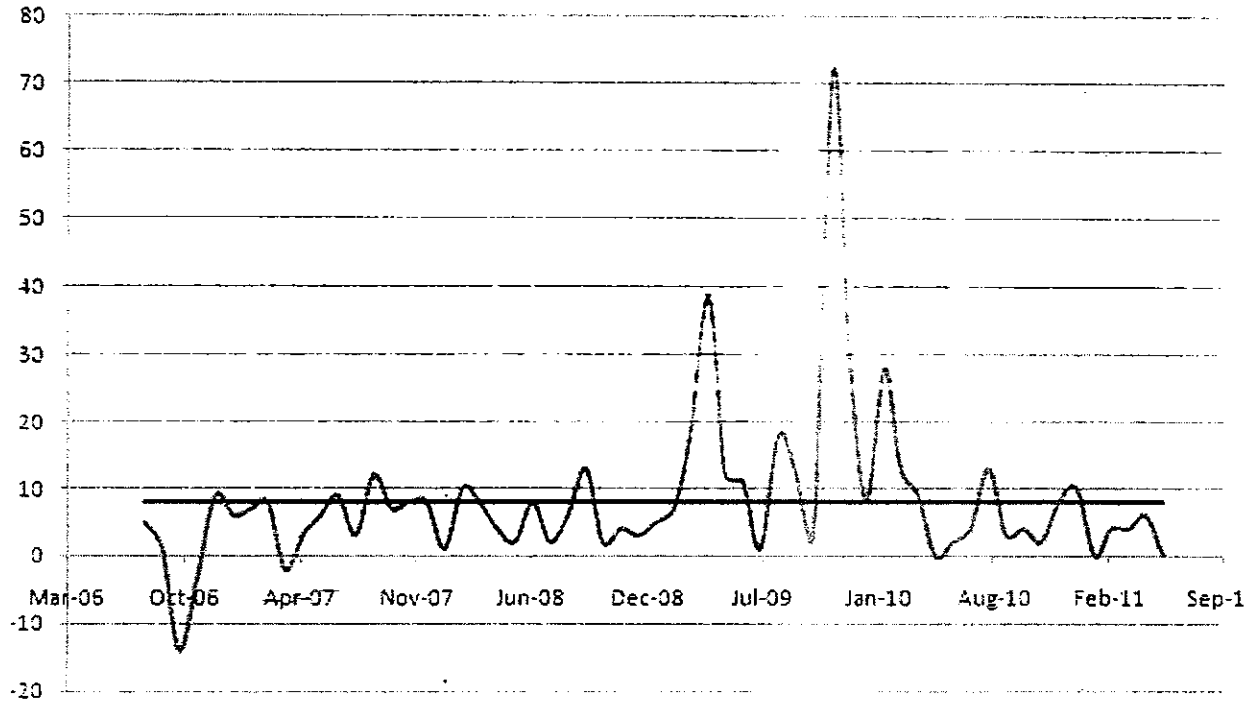


Figure 61: ApacheOFBiz: Revisions Incremental growth (in Source Files) against time.

4.3.4 Results Summary:

The above discussion can be concluded in following points:

1. All three ERPs (Openbravo, Adempiere and ApacheOFBiz) showed continuous growth and hence continuous change trends. In this way all three proved first and sixth laws of Lehman.
2. The incremental growth of all three ERPs showed that the revisions with larger growth are usually followed by revisions with smaller growth, so that the familiarity of stakeholders with the application may be maintained (conserved). This trend proved the existence of the "conservation of familiarity" concept, introduced by Lehman as his fifth law.

3. The growth rate of just one ERP (Adempiere) was found to be decreasing. The other two ERPs (Openbravo and ApacheOFBiz) showed an increasing or invariant growth rate. In this way, just one ERP proved the Lehman's law of "declining growth rate". The other two disproved this law.
4. So the first and sixth laws were totally proved. But the fifth law was partially proved. The concept of "conservation of familiarity" proved. But the concept of "declining growth rate" disproved.

Laws→	**1st Law**	**6th Law**	**5th Law**	
ERPs↓	Continuous Change	Continuous Growth	Limit on Incremental Growth	Declining Growth Rate
Openbravo	Proved	Proved	Proved	Disproved
Adempiere	Proved	Proved	Proved	Proved
ApacheOFBiz	Proved	Proved	Proved	Disproved

Table 4: Summarized results of my case studies

4.4 Comparison of my Results with Other Studies:

Nearly all of the studies, in which Lehman laws have been tested on open source applications, have verified the continuous change and growth trends in open source applications. In other words, all those studies have proved the first and the sixth laws of Lehman. So, with respect to the validity of the first and the sixth laws, my results align exactly with the past studies, like, [Godfrey and Tu 2000], [Capiluppi et al 2004], [Robles et al 2005], [Herraiz et al 2006], [Xie et al 2009].

The existence of the "conservation of familiarity" concept has been proved by some of the past studies, whereas some other studies have disproved it. This concept hasn't been tested at a large scale. I have found only two studies in which this trend was observed in open source applications, [Xie et al 2009] and [Fietelson 2009]. Xie studied seven open source applications and found ripples in their incremental growth graphs, which is an indication of the existence of this concept. Fietelson studied a well known open source operating system, Linux, and found that only some of its releases had entertained the issue of familiarity, but many of the releases had ignored this issue. In this way, Fietelson disproved the existence of this concept. My case studies have proved the existence of this concept. So in this respect, my results are in accordance with Xie, but are different from Fietelson.

The last trend, "declining growth rate", has also been proved as well as disproved in past studies. But most of the studies have disproved it. There are only few cases, in which this concept was proved on open source applications. In most of the studies, the open source applications were found with an invariant or an increasing growth rate. The studies in which this concept was disproved include, [Godfrey and Tu 2000], [Capiluppi et al 2004], [Robles et al 2005], [Izureita 2006], [Simmons et al 2006], [Herraiz et al 2006], [Xie et al 2009], [Ali and Maqbool 2009]. Godfrey disproved this concept on Linux, Capiluppi on ARLA (a distributed file system), Robles on Linux, BSD family kernels and other eighteen applications, Izurieta on Linux and FreeBSD, Simmons on Nethack (a game), Herraize on ten (of thirteen) different applications, Xie on seven different applications and Ali on four different applications. The few cases in which this concept was proved, include three of thirteen applications studied by Herraiz and some

small scale applications studied by Koch in [Koch 2005]. In this way, my results align with the results of past studies, as two of my case studies (Openbravo and ApacheOFBiz) have disproved and only one (Adempiere) has proved this concept.

Revision Month	SLOC	Source Code Files	Incremental Growth (SLOC)	Incremental Growth (Source Files)
Nov-07	102,699	794		
Dec-07	102,982	795	283	1
Jan-08	103,087	796	105	1
Feb-08	76,449	578	-26,638	-218
Mar-08	76,113	581	-336	3
Apr-08	81,676	617	5,563	36
May-08	80,838	609	-838	-8
Jun-08	81,620	613	782	4
Jul-08	82,070	615	450	2
Aug-08	82,037	615	-33	0
Sep-08	82,875	623	838	8
Oct-08	88,823	697	5,948	74
Nov-08	112,672	870	23,849	173
Dec-08	151,680	885	39,008	15
Jan-09	153,710	889	2,030	4
Feb-09	135,631	901	-18,079	12
Mar-09	136,075	901	444	0
Apr-09	138,163	914	2,088	13
May-09	140,272	924	2,109	10
Jun-09	141,526	930	1,254	6
Jul-09	142,852	939	1,326	9
Aug-09	141,978	933	-874	-6
Sep-09	142,645	936	667	3
Oct-09	143,929	950	1,284	14
Nov-09	144,489	953	560	3
Dec-09	143,906	991	-583	38
Jan-10	145,574	1019	1,668	28
Feb-10	148,462	1034	2,888	15
Mar-10	149,511	1039	1,049	5
Apr-10	152,954	1060	3,443	21
May-10	155,023	1078	2,069	18
Jun-10	157,358	1087	2,335	9
Jul-10	159,477	1099	2,119	12
Aug-10	160,488	1102	1,011	3
Sep-10	161,228	1103	740	1
Oct-10	161,642	1106	414	3
Nov-10	163,205	1111	1,563	5
Dec-10	192,347	1339	29,142	228
Jan-11	195,291	1351	2,944	12
Feb-11	198,079	1376	2,788	25
Mar-11	201,623	1388	3,544	12
Apr-11	202,264	1388	641	0
May-11	191,071	1333	-11,193	-55
Jun-11	191,057	1327	-14	-6

Table 5: Size measures for Openbravo revisions

Revision Month	SLOC	Source Code Files	Incremental Growth (SLOC)	Incremental Growth (Source Files)
Nov-06	356,429	2120		
Dec-06	373,773	2161	17,344	41
Jan-07	376,758	2168	2,985	7
Feb-07	375,984	2167	-774	-1
Mar-07	385,271	2170	9,287	3
Apr-07	384,628	2166	-643	-4
May-07	388,361	2183	3,733	17
Jun-07	394,573	2207	6,212	24
Jul-07	401,955	2269	7,382	62
Aug-07	403,470	2276	1,515	7
Sep-07	537,536	3895	134,066	1619
Oct-07	537,850	3909	314	14
Nov-07	566,501	4078	28,651	169
Dec-07	573,517	4082	7,016	4
Jan-08	578,176	4118	4,659	36
Feb-08	580,391	4130	2,215	12
Mar-08	581,853	4137	1,462	7
Apr-08	582,275	4137	422	0
May-08	582,591	4137	316	0
Jun-08	652,915	4505	70,324	368
Jul-08	663,223	4549	10,308	44
Aug-08	656,066	4570	-7,157	21
Sep-08	656,651	4579	585	9
Oct-08	764,941	5201	108,290	622
Nov-08	681,929	4604	-83,012	-597
Dec-08	692,645	4621	10,716	17
Jan-09	696,592	4630	3,947	9
Feb-09	674,050	4511	-22,542	-119
Mar-09	676,518	4518	2,468	7
Apr-09	679,419	4534	2,901	16
May-09	689,704	4558	10,285	24
Jun-09	690,394	4572	690	14
Jul-09	692,165	4606	1,771	34
Aug-09	693,394	4618	1,229	12
Sep-09	678,094	4625	-15,300	7
Oct-09	679,457	4638	1,363	13
Nov-09	681,126	4644	1,669	6
Dec-09	682,700	4650	1,574	6
Jan-10	682,794	4650	94	0
Feb-10	684,543	4654	1,749	4
Mar-10	685,441	4682	898	28
Apr-10	688,528	4698	3,087	16
May-10	689,102	4700	574	2
Jun-10	690,699	4706	1,597	6
Jul-10	691,635	4709	936	3
Aug-10	691,691	4709	56	0
Sep-10	691,877	4710	186	1
Oct-10	692,874	4717	997	7
Nov-10	695,386	4724	2,512	7
Dec-10	695,487	4729	101	5
Jan-11	695,517	4729	30	0
Feb-11	695,883	4729	366	0
Mar-11	695,885	4729	2	0
Apr-11	697,165	4735	1,280	6
May-11	697,508	4735	343	0
Jun-11	705,163	4762	7655	27

Table 6: Size measures for Adempiere revisions

Revision Month	SLOC	Source Code Files	Incremental Growth (SLOC)	Incremental Growth (Source Files)
Jul-06	174,495	1000		
Aug-06	176,297	1005	1,802	5
Sep-06	177,041	1006	744	1
Oct-06	176,867	992	-174	-14
Nov-06	177,011	989	144	-3
Dec-06	177,769	998	758	9
Jan-07	180,318	1004	2,549	6
Feb-07	182,786	1011	2,468	7
Mar-07	185,045	1019	2,259	8
Apr-07	188,022	1017	2,977	-2
May-07	189,596	1020	1,574	3
Jun-07	190,880	1026	1,284	6
Jul-07	194,433	1035	3,553	9
Aug-07	196,714	1038	2,281	3
Sep-07	198,851	1050	2,137	12
Oct-07	199,448	1057	597	7
Nov-07	200,229	1065	781	8
Dec-07	202,739	1073	2,510	8
Jan-08	203,678	1074	939	1
Feb-08	205,642	1084	1,964	10
Mar-08	208,266	1092	2,624	8
Apr-08	210,142	1096	1,876	4
May-08	211,468	1098	1,326	2
Jun-08	213,002	1106	1,534	8
Jul-08	213,388	1108	386	2
Aug-08	214,862	1114	1,474	6
Sep-08	217,243	1127	2,381	13
Oct-08	217,887	1129	644	2
Nov-08	218,560	1133	673	4
Dec-08	219,511	1136	951	3
Jan-09	219,966	1141	455	5
Feb-09	221,867	1148	1,901	7
Mar-09	224,400	1166	2,533	18
Apr-09	234,696	1205	10,296	39
May-09	236,571	1217	1,875	12
Jun-09	240,654	1228	4,083	11
Jul-09	243,629	1229	2,975	1
Aug-09	247,354	1247	3,725	18
Sep-09	249,863	1259	2,509	12
Oct-09	251,127	1262	1,264	3
Nov-09	255,444	1334	4,317	72
Dec-09	258,666	1365	3,222	31
Jan-10	262,082	1373	3,416	8
Feb-10	267,552	1401	5,470	28
Mar-10	270,946	1414	3,394	13
Apr-10	272,494	1423	1,548	9
May-10	272,024	1423	-470	0
Jun-10	272,950	1425	926	2
Jul-10	274,390	1429	1,440	4
Aug-10	276,212	1442	1,822	13
Sep-10	276,547	1445	335	3
Oct-10	278,696	1449	2,149	4
Nov-10	279,109	1451	413	2
Dec-10	281,636	1459	2,527	8
Jan-11	285,533	1469	3,897	10
Feb-11	285,893	1469	360	0
Mar-11	287,019	1473	1,126	4
Apr-11	287,988	1477	969	4
May-11	289,912	1483	1,924	6
Jun-11	290,200	1483	288	0

Table 7: Size measures for ApacheOFBiz revisions

CHAPTER 5: CONCLUSION AND FUTURE WORK

The aim of this work was to check validity of three of the Lehman laws on three different open source ERP systems, which has been achieved. It was decided to check validity of Lehman's growth related laws (first, fifth and the sixth) on three open source ERPs (Openbravo, Adempiere and ApacheOFBiz), which has been done. In this way this work is complete in its mandate. According to the results of this work, the Lehman law of "declining growth rate" has been found to be disproving on two of three ERPs. However, on the basis of these results, it can't be concluded that this law really don't hold for open source applications because these cases of invalidity can be the exceptional ones. Although there exists certain other examples in the past studies, in which this law was disproved on open source applications, like Linux kernel, BSD family kernels and Nethack etc, but even all these cases don't provide enough evidence to challenge this law at all. Lehman postulated his laws on the basis of thirty years experience, and a large number of case studies conducted under the project named as Features, Evolution And Software Technology (FEAST). These laws are widely accepted and have been validated by many other researchers. The few examples of invalidity are not enough that the validity of these laws can be argued. For this purpose more open source applications (taking from different domains) are needed to be studied and observed.

One of the possible reasons of invalidity of Lehman laws on some open source applications can be that the open source applications, unlike closed source applications, don't have a clear distinction between development and evolution phases. In case of closed source applications, the requirements for whole application are collected and then on the basis of those requirements, a full featured application is developed. Once it is fully developed then it is delivered to end-users. Before first delivery, all changes and growth are considered to be part of development rather than evolution. After delivery of the first version, when end-users report errors or demand for enhancements, then evolution starts. But in case of open source applications, the application is sometimes delivered or started to use even when it is in development phase. In this case, the development and evolution phases overlap each other, means the application is both developed and evolved simultaneously. Thus, it becomes difficult to differentiate between developmental and evolutionary changes.

If some application is started to use even when it is not fully developed then in the initial period of evolution, two types of growth will occur, developmental growth and evolutionary growth, which can cause rapid growth in that application. At that time, it will be showing deviation from fifth law of Lehman. But it is possible that after some time period, such type of application also become align with Lehman, when its development becomes complete and only evolutionary growth remains left.

REFERENCES:

[Ali and Maqbool 2009] Ali S, Maqbool O, Monitoring Software Evolution using Multiple Types of Changes, IEEE International Conference on Emerging Technologies (ICET) 19-20 Oct, 2009, pp 410–415

[Bonkoski 2007] Bonkoski B, Open Source Software Evolution: Case Study Nagios, http://blog.greedygeeks.com/bonkoski_nagios_evolution.pdf

[Capiluppi et al 2004] Capiluppi A, Morisio M, Ramil JF, Structural Evolution of an Open Source System: A Case Study, Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC'04), 24-26 Jun, 2004, pp 172-182

[Dong and Mohsen 2008] Dong Y and Mohsen S, Does Firefox Obey Lehman's Laws of Software Evolution? Masters Candidate, Department of Computer Science, University of Waterloo, Waterloo, ON, Canada

[Israeli and Feitelson 2009] Israeli A, Feitelson DG, The Linux Kernel as a Case Study in Software Evolution, Journal of Systems and Softwarem, Volume 83, Issue 3, March 2010, pp 485-501

[Godfrey and Tu 2000] Godfrey MW and Tu Q, Evolution in Open Source Software: A Case Study, Proceedings of the International Conference on Software Maintenance, pp. 131 - 142, 2000

[Herraiz et al 2006] Herraiz I, Robles G, Gonz´alez-Barahona JM, Capiluppi A, Ramil JF, Comparison between SLOCs and Number of Files as Size Metrics for Software Evolution Analysis, Proceedings of the Conference on Software Maintenance and Reengineering (CSMR'06), 22-24 Mar, 2006, pp 213-220

[Izurieta and Bieman 2006] Izurieta C and Bieman J, The Evolution of FreeBSD and Linux, Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE), 2006

[Koch 2005] Koch S, Evolution of Open Source Software Systems – A Large-Scale Investigation, The First International Conference on Open Source Systems, Genova, Italy, July 11 - 15, 2005

[Leh 78] id, Laws of Program Evolution—Rules and Tools for Programming Management, Proc. Infotech State of the Art Conf., Why Software Projects Fail?, Apr. 1978, pp. 11/1-11/25.

[Leh 97 a] Lehman MM, Laws of Software Evolution Revisited, Proceedings of EWSPT'96, Nancy, LNCS 1149, Springer Verlag, 1997, pp. 108–124

[Leh et al 97 b] Lehman MM, Ramil JF, Wernick PD, Turski WM and Perry DE, Metrics and Laws of Software Evolution - The Nineties View, in Proceedings of the Fourth Intl Software Metrics Symposium, Nov 5-7, Albuquerque, NM, 1997, pp 20-32

[Leh et al 98 a] Lehman MM, Perry DE, and Ramil JF, On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution, in Proceedings of the Fifth International Metrics Symposium, Metrics '98, Bethesda, Maryland, Nov. 20-21, 1998

[Leh et al 98 b] Lehman MM, Perry DE, and Ramil JF, Implications of Evolution Metrics on Software Maintenance, in Proc. Of the 1998 Intl. Conf. on Software Maintenance (ICSM'98), Bethesda, Maryland, Nov 1998

[Leh and Ramil 2001] Lehman MM and Ramil JF, Rules and Tools for Software Evolution Planning and Management, Annals of Software Engineering, 11(1):15–44, 2001

[Linuxlinks 2011] Linuxlinks, 2011. http://www.linuxlinks.com/article/20091129070817552/ERP.html

[Robles et al 2005] Robles G, Amor JJ, Gonzalez-Barahona JM and Herraiz I, Evolution and Growth in Large Libre Software Projects, Proceedings of the International Workshop on Principles of Software Evolution, pp. 165 – 174, 2005.

[Simmons et al 2006] Simmons MM, Vercellone-Smith P, Laplante PA, Understanding Open Source Software through Software Archaeology: The Case of Nethack, Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop SEW-30 (SEW'06), Apr 2006, pp 47-58.

[Sommerville 2005] Sommerville I, Software Engineering, Pearson Education, 2005

[Turski 96] Turski MW , Reference Model for Smooth Growth of Software Systems, IEEE Transactions on Software Engineering, Volume 22, Issue 8, August 1996

[Xie et al 2009] Xie G, Chen J and Neamtiu I, Towards a Better Understanding of Software Evolution: An Empirical Study on Open Source Software, Proc. IEEE ICSM (International Conference on Software Maintenance), Sep 20-26, 2009, Edmonton, Canada