Measuring the Open Source Software using Version Control System to study the Software Evolution

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science and Applications

Submitted By
Nindya Kotwal
(Roll No. 601003017)

Under the supervision of:
Vineeta Bassi
Assistant Professor, SMCA

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS
THAPAR UNIVERSITY
PATIALA – 147004
June 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Measuring the Open Source Software using Version Control System to study the Software Evolution*", in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science and Applications* submitted in School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Vineeta Bassi* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Nindya Kotwal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Vineeta Bassi)
Assistant Professor
SMCA

Countersigned by

(Dr. S.S.Bhatia)
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ABSTRACT

Open Source Software is defined as the software products whose source code has been freely available to all its intended users. Over the years these kind of software is becoming popular. The open source software movement has received enormous attention in the field of software development industry. The open source software movement is another way of developing the software and poses a challenge to the commercial software market. Open source development movement has been getting focus on both the area of development and academics. This type of development is not a threat posed to commercial market of software development by a new competitor who works according to the same rules and process but the movement threatens because of its cheaper, faster and better work. The style of development is radically different as compared to closed source software development. Open Source Software systems are built by potentially large number of volunteers who seldom meet face to face but work together. The developers coordinate their activity by means of email, bulletin boards and the version control system tools. The open source software thus poses a real challenge and if open source market is really posing a threat to the commercial development so it is vital to understand and evaluate the open source software. This study thus has been conducted focusing on the evolution of the open source software. The idea in this thesis is to study the evolution of the open source software under the effects of the data mined from the version control system. Version Control System includes tools used in the open source software development which helps in coordinating the work by the developers distributed all over the world. Version Control System contains a repository of the source code of various versions and also tracks the previous version data. The method used is to observe the differences of the series of repositories and then comparing these observations and finally tracking how the system gets evolved over a selected period of time.

In this thesis report an open source programmer's editor is studied on its various parameters and attributes and then by using the empirical approach various evolutionary characteristics are measured and evaluated. The method used is quantitatively comparing the repositories, showing the relation among various measurements and providing evidence to support the Lehman's Laws of software evolution.

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Ms. Vineeta Bassi** Assistant Professor, School of Mathematics and Computer Applications, Thapar University, Patiala for her patient guidance and support throughout this report. I am truly very fortunate to have the opportunity to work with her.

I am also thankful to our **Head of the Department, Dr. S. S. Bhatia** as well as **PG Coordinator, Mr. Singara Singh, Assistant Professor,** School of Mathematics and Computer Applications, entire faculty and staff of School of Mathematics and Computer Applications and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would also like to thank my parents for their years of unyielding love and encourage. They have always wanted the best for me and I admire their determination and sacrifice.

Nindya (✗)

Nindya Kotwal
(601003017)

List of Contents Page No.

List of Figures Page No.

List of Tables Page No.

List of Abbreviations

BLOC	Blank Lines of Code
BSD	Berkley Software Distribution
C&CLOC	Lines of Code and Comments
CLOC	Comment Lines of Code
CVS	Concurrent Version System
CWORDS	Comment Words
GCC	GNU Compiler Collection
GPL	GNU General Public License
ISO	International Standardization Organization
LGPL	GNU Lesser General Public License
LOC	Lines of Code
MPL	Mozilla Public License
MSR	Mining Software Repositories
MTTF	Mean Time to Failure
MVG	McCabe Cyclomatic Complexity
NOC	Number of Classes
NOF	Number of Files
NOM	Number of Methods
NOP	Number of Packages
OSD	Open Source Development
OSI	Open Science Initiative
OSP	Open Source Projects
OSS	Open Source Software
OSSD	Open Source Software Development
OSSDP	Open Source Software Development Process
PHP	Hypertext Preprocessor
RCS	Revision Control System
SCCS	Source Code Control System
SLOC	Source Lines of Code

SLOC-L	Logical Source Lines of Code
SLOC-P	Physical Lines of Code
SVN	Subversion
TCO	Total Cost of Ownership
VC	Version Control
VCS	Version Control System
XML	Extensible Markup Language

Chapter 1

INTRODUCTION

Open Source Software (OSS) is becoming popular over the years and researchers also show interest in Open Source Software due to its quality and reliability. As the OSS is becoming more and more popular and relevant in both the areas of industry and academics so everybody is interested in understanding how these software projects has been produced. This thesis has focused on the evolution of the Open Source Software. The goal of the thesis is Open Source Software and its evolution under the effects of Version Control System. The work includes examining the observed differences in all the versions of an Open Source Software and then comparing these series of versions, and also tracking the changes in projects as they evolve, this thesis also proved the Laws of Evolution and provides a body of original empirical evidence to support it. In recent times, given the strong emergence of open source software, knowledge about its quality and how it evolves, particular knowledge that has been gathered in a clear and methodical fashion, is a very valuable thing.

This chapter will set the context for the remainder of the thesis. It will introduce the concept of open source software, version control and software evolution. Furthermore, the specific objectives will be set forth and the structure will be outlined.

1.1 Open Source Software

The term "Open Source" was introduced during the foundation of the Open Source Initiative in 1998. Open source software has becoming increasingly popular and important, is computer software that has its source code made available under an Open Source Definition (OSD) based license [1]. OSD based license implicates that the source code of software is released with binary, allowing users and developers to use and to modify the software and to distribute any improvements they make. Consequently, most of OSS is being developed in public accessible projects where everyone capable of contributing knowledge, ideas or code is welcome to join in and

such projects are called as Open Source Projects (OSP) [2] Figure 1.1 shows the relation among various common terms used in Open Source. Open Source Software is produced by loosely organized, communities consisting of contributors from all over the world who seldom if ever meet face to face, and who share a strong sense of commitment.

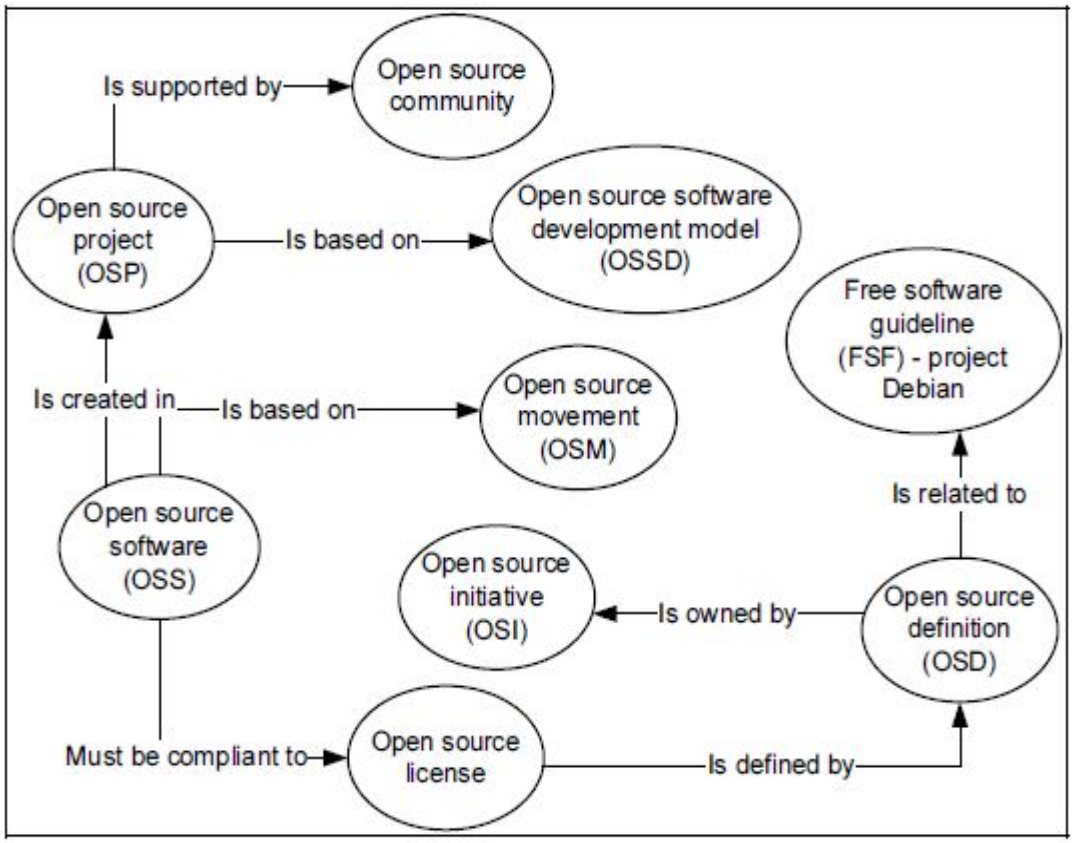


Figure 1.1 Relation among the open source terms [2]

OSS is software for which the source code is publicly available, though the specific licensing agreements vary as to what one is allowed to do with that code. OSS is software whose licenses give users these essential 'freedoms':

- to run the program for any purpose, to study the workings of the program,
- to modify the program to suit specific needs,
- to redistribute copies of the program at no charge or for a fee,
- to improve the program, and release the modified version of the program,
- There is a one which is basic; users must have access to its source code [3].

1.1.1 Some examples of Open Source Software

Table 1.1 shows some successful projects developed in open source society.

Table 1.1 Open Source Software Examples [4]

GNU/LINUX	Operating System
Apache web server	Web Server
MySql	Relational Database Management System
OpenOffice	Office Suite
Mozilla Firefox	Web Brower
Thundermail	Email Client
Ecllipse	Java Integrated Development Environments
Tcl,Python,Perl,PHP	Languages
Tex/Latex	Typesetting and Document
emacs, jEdit	Text Editors

1.1.2 Open Source Development Process

Open Source Development is not a silver bullet but just an alternative approach showing how the Internet can change the way software is constructed, deployed, and evolved. The basic principle for the OSS Development Process (OSSDP) is that by sharing source code, developers cooperate under a model of systematic peer-reviews, and take advantages of parallel debugging that leads to innovation and rapid advancement. The OSSDP model has the following features:

- Collaborative, parallel development involving source code sharing ,
- Collaborative approach to problem solving through feedback and peer review,
- Large pool of globally dispersed, highly talented, motivated professionals,
- Extremely rapid release times,
- Increased user involvement as users are viewed as co-developers [5].

Woods, D. et al suggested the open source development model for the enterprise as shown in Figure 1.2. In this model each project begins with the initial

determination of an idea or need which can originate from any one person or community. The next step in the process is the initial development for a proof of concept to determine the feasibility of the project, leading directly into the initial public prototype.

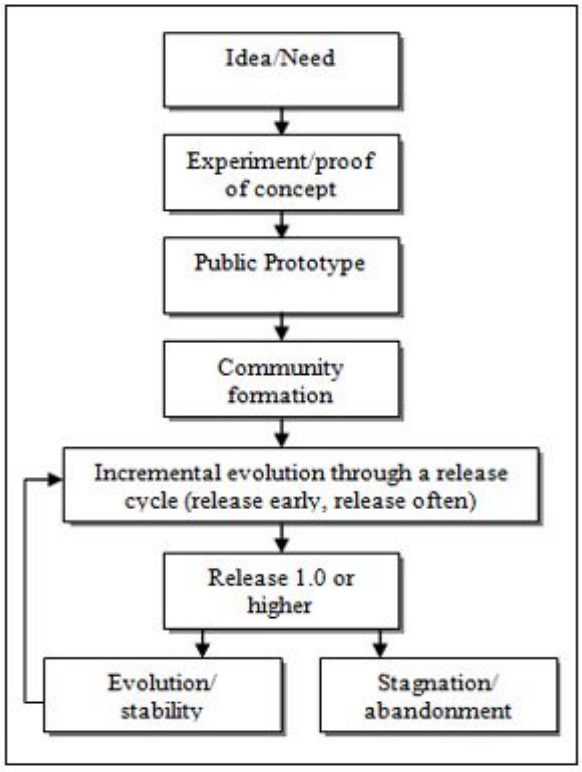


Figure 1.2 Open Source Development Process [6]

The public prototype's development is the core infrastructure surrounding the project and consists of the initial steps at programming for the new software program [6]. The intent of the public prototype is to assist in the creation of a community around the prototype with a clear understanding of the original project's proof of concept. Through time, the initial prototype is opened for review and contributions are made by others within the community. Each subsequent addition to the software program is released reflecting an incremental evolution in the product development. After this initial core development, the project will either become stagnant or will continue to evolve and mature. Stagnation or abandonment of open source projects may occur

through a perception of completion for the project, poor project leadership which removes incentives for future contributors, or simply through lack of interest [6].

1.1.3 Open Source Software Licenses

The Open Source community is big and diverse, and it is easy to say the same about the amount of Open Source licenses available. An OSS license defines the privileges and restrictions that a user must follow in order to use and modify software. If a developer wants to publish a program as OSS, he/she can distribute the program as an uncopyrighted product. The users of the program can read, copy, modify, and redistribute the program. However, it is also possible for someone to make the program copyrighted by modifying the original program. Consequently, the modified, copyright-protected program becomes a personal property, and is not OSS any more. To prevent this situation, most of OSS licenses implement "copyleft" concept: anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it [7]. Developers can make their own licenses as well. The process to publish an OSS license is as follows:

- **Choose a unique name for the license.**

  A user does not want to make the mistake of naming the license that closely resembles another or a name that is hard to remember. Making the name as close as possible to the purpose of the software is recommended.

- **Compile a legal analysis.**

  The license should satisfy the Open Source Definition. The analysis should come from a legal expert in the country and all documentation is considered confidential.

- **Review the license.**

  The legal analysis will be sent to an Open Source Initiation (OSI) discussion session where members review the license and notify the user of any changes needed to be made. Also if the license is composed from

a previous license, the user needs to explain what has been added and how it makes the license better. The license is voted, and accepted or rejected according to the result. If it is accepted, the license is published as an OSS license.

There are four well-known and classic licenses already published. The following gives a brief overview of the licenses:

- **Berkeley Software Distribution (BSD).**
  BSD was introduced in the 1970's where the license was originally written at the University of California Berkeley. BSD allows commercial use and the program under the BSD can be incorporated into other commercial products [7].
- **GNU Public License (GPL).**
  GPL was introduced in 1991 and allows users to use the program and to make modifications, but prevents code from being marketed under exclusive legal rights. It also forbids programs to link to the exclusive rights. GPL was written by Richard Stallman, and grants the four levels of freedom to the user [7].
- **Lesser General Public License (LGPL).**
  LGPL, formally known as GNU Library General Public License was published by the free software foundation. LGPL was written in 1991 and updated in 1999. This license is similar to GPL, but allows links to a library or non LGPL/GPL program [7].
- **Massachusetts Institute of Technology (MIT).**
  MIT commonly called X11 License was written at the Massachusetts Institute of Technology. MIT allows software to be used under exclusive rights. Programs under this license can be modified to fit the user's needs [7].
- **Mozilla Public License (MPL).**
  MPL was introduced by Mitchell Baker at the Netscape Communications Corporation and is used for Mozilla Firefox, Mozilla Application Suite, and Mozilla Thunderbird. MPL is a combination of the BSD and GPL

licenses. It allows the user to copy or change the source code, but has a restriction, which the source code changed under MPL must stay under MPL. In other word, it is prohibited under the MPL that the user modifies the MPL-licensed source code and put it under another license. However, it allows bringing source code from another license to the MPL code as long as the combined code is under MPL [7].

Following table show the comparisons between licenses adopted by OSS society

Table 1.2 Comparisons between Open Source Licenses [7]

	Notice of modification	Redistribution of the modified work	Original source code attached to the modification	Linking to source code	Liability notice
BSD	Yes	Yes	No	Yes	Yes
GPL	Yes	Only under GPL	Yes	No	Yes
LGPL	Yes	Only under LGPL	Yes	Yes	Yes
MIT	Yes	Yes	Yes	Yes	Yes
MPL	Yes	Only under MLP	Yes	Yes	Yes

1.1.4 Advantages of Open Source Software

Following are benefits of using open source software:

- **Development Speed.**

  Reuse of code implies speedier development: the quicker the product is released and becomes valuable to a user group [5].

- **User Involvement.**

  Users are treated as a valued asset in the development process. Viewing users as co developers leads to code improvement and effective debugging [5].

- **Access to existing code.**

  Developers have access to the "open source toolset", a huge amount of open source project code which can speed up development [5].

- **Cost.**

  Total Cost of Ownership (TCO) of OSS is widely debated, particularly within developing Countries. A basic tenet of open source is that all source code is free and available to any user to modify and improve. By contrast proprietary software requires paid-for licensing that is generally considered a barrier to access for developing countries. The cost of Windows XP and Office XP is about US$560 which equates to approximately 2.5 months of GDP per capita in South Africa, and 16 months in Vietnam. In some phases of ownership, there is evidence that OSS may have advantages in the area of TCO [5].

1.1.5 Disadvantages of Open Source Software

In spite of all the above features there are some disadvantages while using the open source software technology. Following are disadvantages of using open source software:

- **Support issues.**

  While open source projects have a wide variety of resources (developers themselves, Internet mailing lists, archives and support databases) that can be tapped for support, the problem is that there is no single source of information, no help desk that provides 'definitive' answers to problems [5].

- **No guarantee of updates.**

  Since we are not paying for the open source software nobody is bound to give you regular updates. We can get stuck with the same old version for years without ever getting an update. Of course commercial software is sometimes too expensive and people who don't have big budgets cannot afford them. Some buggy software is better than no software. Similarly there are many poor countries where low cost open source software can be used without incurring huge costs. But open source software shouldn't be promoted as a commercial alternative [5].

- **Higher Installation Cost.**

  Another great problem is that most of the open source applications are incompatible with the present day gadgets. For instance if we use some open source operating system we can forget about the cool plug and play hardware that we have been using for so many years. Technical support too is costlier compared to commercial software because people who provide support for free source [5].

1.2 Version Control

Version Control Systems (VCS) are used to store and reconstruct the past versions of program code. It will provide the information to better understand a program's development history. A version control system (VCS) tracks each change a developer makes to the system and, as a result, can recreate a consistent snapshot at any point in time. Examples of VCS include CVS (Concurrent Version System) and Subversion.

A version control system is a piece of software that helps the developers on a software team work together and also archives a complete history of their work.

There are three basic goals of a version control system:

- To make people work simultaneously, not serially,
- When people are working at the same time, and want their changes to not conflict with each other,
- Project requires archiving every version of everything that has ever existed ever [8].

1.2.1 A History of Version Control System

The history of version control tools can be divided into three generation as shown in Table 1.3. The forty year history of version control tools shows a steady movement toward more concurrency. The version control tools moves from no networking to the distributed networking now.

Table 1.3 Three Generations of Version Control [8]

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before Commit	CVS, SourceSafe, Subversion, Team Foundation Server
Third	Distributed	Changesets	Commit before Merge	Bazaar, Git, Mercurial

1.2.2 Version Control System

While this basic functionality of VCS is essential for version control and measuring the evolution of metrics over time there is much data in a VCS that is ignored when simply using it to extract snapshots of source code. The purpose of version control is to easily distinguish the version changes, versions retrieval and tracking and to indicate the relationship between the various versions. With the increasing complexity of software systems, as well as the user needs, software updates frequently, version control gradually becomes an important control process of software configuration management and it is the important part of the quality management in software engineering.

The version control system allows concurrent development and is widely adopted in the open-source community, especially for large projects like ECLIPSE, MOZILLA, and JEdit. Therefore, recent research used subversion to investigate the information about changed data, that is, who changed what, why, when, and how the changes grows in the program's.

Version, as one practical example is the configuration item that has been identified. With the progress in software development, versions are also

constantly evolving; these different versions of different configuration items constitute a complex version space. The new version of the system may have different functions, performance, may modify the system error. Some versions may have no difference in function, just design for different hardware or software configuration. If there is only some slight differences between versions, and then sometimes a version is called a variant of another version. Version control systems (VCS) are used to store and reconstruct past versions of program source code. As a by-product they also capture a great deal of contextual information about each change [9]. While this basic functionality of VCS is essential for version control and measuring the evolution of metrics over time, there is much data in a VCS that is ignored when simply using it to extract snapshots of source code [9].

1.2.3 Version Control History

Version control data is also referred to as the version control history and version control history majorly contains the source code but in addition to source code it is also been used to explore the relationships between various aspects of software development. For example, the requirements of the software, the implementation technology used, development process followed, and the organization of the developers have some effect on how the software evolved [9]. Figure 1.3 below is illustrating the idea of various interactions in version control system.

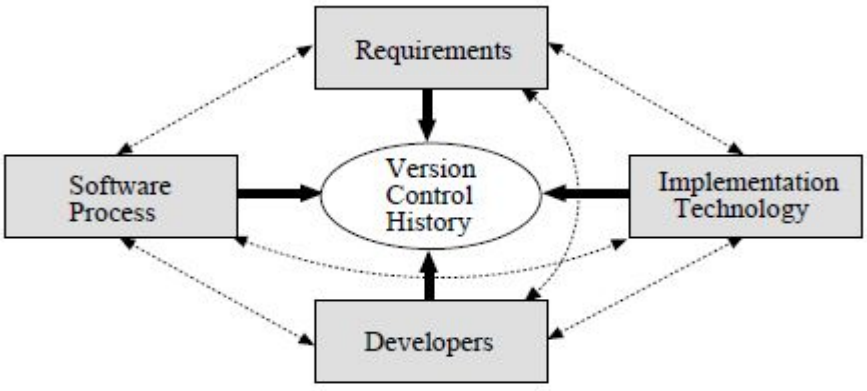


Figure 1.3 Version control history captures interactions [9].

1.2.4 Version Control Tree

The version control tree reflects the evolution history of the project development. Generally, version evolution has two ways: Serial evolution and Parallel evolution [10]. Each new version that formed by the Serial evolution may be evolved from the current latest version. So, according to the evolution process of different versions to forms a simple chain, known as the version chain. In this way, version evolution is forward based on one to one mapping relations, usually in order to compensate for deficiencies, improve performance and adapt to the environment. Parallel evolution uses one to many manners.

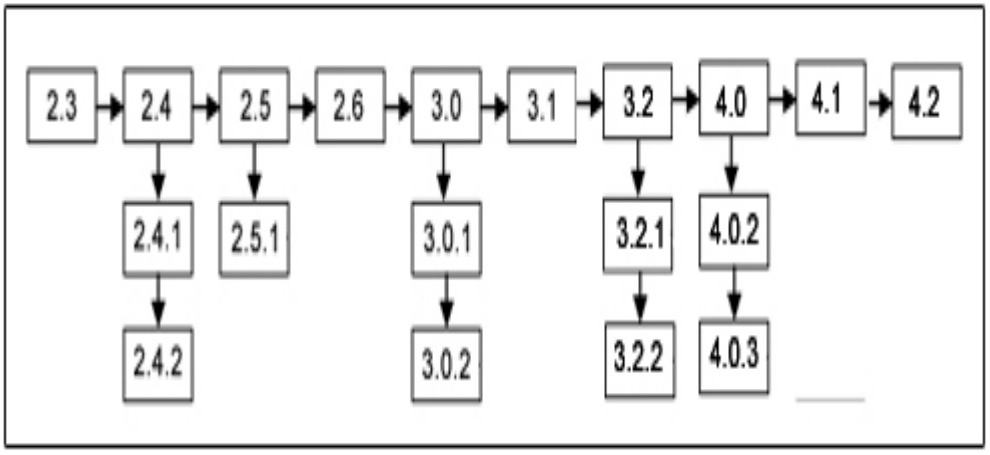


Figure 1.4 Version Tree Sample Graph

In practice, both versions evolution forms are usually merged with each other, form the more common version map with branches, known as version tree. The version tree reflects the evolution history of the project development.

Figure 1.4 shows the version tree graph. In Figure 1.4, a total of six version chains are showed, they are "2.3-2.4-2.5-2.6-3.0-3.1-3.2-4.0-4.1-4.2", "2.3-2.4-2.4.1-2.4.2", "2.3-2.4-2.5-2.5.1", "2.3-2.4-2.5-2.6-3.0-3.0.1-3.0.2", “2.3-2.4-2.5-2.6-3.0-3.1-3.2-3.2.1-3.2.2” and “2.3-2.4-2.5-2.6-3.0-3.1-3.2-4.0-4.0.2-4.0.3”. “2.3-2.4-2.5-2.6-3.0-3.1-3.2-4.0-4.1-4.2” is the serial evolution. "2.4-2.4.1" and "2.4-2.5" is the parallel evolution.

1.2.5 Advantages of Version Control System

The following list shows some of the main benefits, but since Version Control systems vary greatly on the implementation side, it cannot be guaranteed that all points are equally available or possible for all systems:

- **One Central Place for all the code.**

  Version Control system offers a central place to all files of a project and these files are accessible for all team members involved in the project.

- **History and People Tracking.**

  If a project is under version control then one can have the possibility to look back to any of the older versions of project and revert changes that have been made and identify each line of code by name of the author.

- **Management of Different Development Lines.**

  If any of the projects under version control needs to have different lines of development, e.g. different product versions or versions for different operating systems, then Version Control system organizes and helps in managing these different lines for the developers, and allows them to merge code parts from one line of development to another. Thus help in managing different areas of project in different development lines and finally merging to get full project.

- **Parallel Development.**

  Many of the available version control systems support the feature of parallel development. Thus it helps two or more than two developers to work on the same piece of code at the same time and finally changes have been merged automatically by the VCS with minimum contribution from the developers involved.

- **Low Network Usage and Small File Repositories.**

  Most Version Control Systems do not send the complete files over the network, but only deltas, which are then used to reconstruct the particular version at the client's side. This saves bandwidth and makes the work with such systems faster if one deals with many or large files of several megabytes.

Besides all these benefits, Version Control is also a necessary part to achieve ISO 9000 conformance, especially for (document) change tracking and auditing procedures. If a company likes to get certified there is no way of getting around VC. There are many solutions available for Version Control, both proprietary and Open Source.

1.3 Software Evolution

Software evolution is the defined as term used in software engineering which states it as the process of developing the software project initially, then repeatedly updating it for various reasons. Software evolution is defined as all the programming activity that is intended to generate a new software version from an earlier operational version [11].

Software evolution takes place only when the initial development was successful. The goal is to adapt the application to the ever-changing user requirements and operating environment [12]. Lehman et al [11] has done extensive research on evolution of large and long lived software. Lehman's laws of software evolution, based on the empirical study, indicate that continuous change and growth is required for keeping the software long-lived. The laws also suggest that over the period, due to changes and growth, software system becomes complex and it becomes more and more difficult to add new functionalities to it. Lehman's work is perhaps the most important origin of the systematic study of software evolution, which has been conducted over a number of decades and has given rise to a growing number of critical observations about how software develops. Lehman originally investigated the growth of the IBM OS/360 operating system and showed that its growth in size was essentially linear and predictable, but later became unstable and unpredictable.

From this investigation, and other later case studies, was built a set of statements about how evolvable real-world software systems develop over time and what activities should be carried out to ensure that their development does not become

chaotic. Proposing them to be external regulating and constraining forces requiring organizational dynamics to overcome, Lehman has termed them “laws” as "Laws of Software Evolution". Prof. Meir M. Lehman, who worked at Imperial College London from 1972 to 2002, and his colleagues have identified a set of behaviors in the evolution of software. These behaviors are known as Lehman's Laws of software evolution.

The following table shows the Lehman’s Law of Evolution.

Table 1.4 Lehman’s Laws of Software Evolution [13]

1.Continuing Change	A system must continually be adapted otherwise it becomes progressively less satisfactory in use.
2. Increasing Complexity	As a system is evolved its complexity increases unless work is done to maintain or reduce it.
3. Self-Regulation	Global system evolution processes are self-regulating.
4. Conservation of Organizational Stability	Unless feedback mechanisms are appropriately adjusted, average effective global activity rate in an evolving system tends to remain constant over product lifetime.
5. Conservation of Familiarity	In general, the incremental growth and long term growth of systems tend to decline.
6. Continuing Growth	The functional capability of systems must be continually increased to maintain user satisfaction over the system lifetime.
7. Declining Quality	Unless rigorously adapted to take into

	account changes in the operational environment, the quality of a system will appear to be declining.
8. Feedback System	Evolution processes are multi-level, multi-loop, multi-agent feedback systems

1.4 Thesis Motivation

In last several years, the open source software movement has received attention in the software industry. The OSS movement is an another way of developing the software product with faster and cheaper approach that pose a serious challenge to the commercial market and has also dominate the software market. OSS movement has been focused now-a-days from both research and industry point of view. The challenge pose by this movement is not due to a new competitor who is following the same rules and process to develop the software product but the movement has their own ways to develop the product which threatens because of its faster, cheaper and better work. Open source developments typically have a central authority that selects some subset of developers who develop code for the official releases and makes them widely available for distribution. The style of development is radically different as compared to closed source software. OSS systems are built by potentially large number of volunteers. Work is not assigned to the people they choose to work. There is no explicit system level design or even detailed design. There is no project plan, schedule or list of deliverable. These given differences has taken together suggest that developers who are geographically distributed, rarely or never meet face to face but coordinate their activity by means of email, bulletin boards and the version tools. The coordination is the main work and is generally considered as the most important mechanisms for geographically distributed development than for the co-located development. It has been proved and claimed also that defects are found, fixed and corrected very quickly because there are many eye balls that are looking for the problems which enhance the quality. The code is

written with more care and creativity because developers are not forced to do, they actually have a real passion to develop the code.

If the OSS movement has really posed a strong challenge to the commercial development so it's vital to study and evaluate these projects. The OSS movement has provoked research, debates about its implications which increase the research work and participation of researchers in this field. Due to increase in the study and discussion over the OSS movement the available information has been increased concerning the open source software. The information has been available over the websites, blogs, research papers, books, technical reports and articles.

Hence this work is motivated to establish research and contribute information borne of empirical observations that are focused upon a relatively narrow set of open source software properties. It is hoped that this work will add more information to the growing research literature that documents open source software.

1.5 Thesis Objective

The aim of this thesis is to present a method that will help in measuring the evolution of the open source software using the version control system data. The data is measured and then used to predict the flow of work done in terms of development efforts and maintenance efforts. The focus of this thesis is on the measurement of the product outcomes and the process used in the development. Tools and process have been used in the measurement of the source code across different versions of the software product.

To study the software evolution process, all the versions has been studied using the source code data and the code has been measured across various important parameters of the software system. The measurements are thus correlated to obtain the results which show how these parameters affect each other in the evolution of the Open Source Software Development.

The specific goals of this thesis are:

- To empirically study the evolution of an open source software system based on a set of metrics.
- To find correlations between the complexity metrics and the development effort measures on different versions of Open Source Software.
- To show that the experimental results follow Lehman's law of evolution.

1.6 Thesis Outline

This thesis is divided into six chapters, including this introduction.

Chapter 1 is the introduction.

Chapter 2 examines various research papers that are related to Open Source Software and its evolution. It discusses previous works that form the roots of subsequent research into open source software; the ideas and concepts across software measurement and the laws of software evolution and how these laws specifically apply to open source software. It also presents the principles, techniques and tools behind mining software data sources using the concept of version control. It also introduces the analysis methods and tools used to get the data from various versions of the software product.

Chapter 3 presents the problem definition and justification to the stated problem.

Chapter 4 presents further empirical studies; in this case, the analysis is applied to the product attributes of project in the same collection of repositories. Once again, the repositories are compared to each other; to reach conclusions about the evolution of the projects in relation to each other and then using these measurements to prove the laws of evolution. The selected set of attributes is also compared to find relationship among the project attributes.

Chapter 5 presents the results from the research undertaken.

Chapter 6 then draws the thesis to a close by formalizing the results presented and discussed in previous chapter. The discussion is followed by a section on what further work can be done within this field.

Chapter 2

LITERATURE REVIEW

Open Source Software is relatively a new field of research in the academic world. This literature review was conducted from the view of understanding how to build and measure a set of metrics used to define the evolution of Open Source Software with the intent of proving the Lehman's Laws of Evolution in the area of Open Source Software Development. In an effort to understand the Open Source Software evolution discipline the literature review examines the emergence of the OSS, version control data and software evolution discipline. The area of OSS was analyzed in terms of success and failure. The set of metrics used to define the evolution can be identified and then correlated also to show the relation between them.

This literature review is intended as an introductory guide to help understanding in the area of Open Source Software Development and from this to demonstrate the need for the creation of relevant metrics to measure the evolution of OSS and also in order to correlate the metrics used in measuring the software.

2.1 Open Source Software

About Open Source Software

The term "Open Source Software" has been gaining much popularity in both the fields of research and software industry as an alternative approach to software development. Open Source Software is defined as a "software whose source code is available to its users for modification, use, and redistribution" [14]. The basic attraction of open source software development area is open source code which is freely available to read, modify or redistribute by any of the interested user. For motivation of an individual in open source software the adopted license has crucial role [15].OSS can be seen differently from two end points, at one of its end it gives the developers all the possibility to contribute in the form of source code and at the other end it provides the users also the right to use or modify a program and its code

base. The central element in open source development model is the open and collaborative environment in which software products are created swiftly. The cooperation between both developers and end users in the open source community encourages the building of products with a higher level of quality throughout the product life-cycle [16]. The exact degree of freedom included in the distribution of code, relies on the license type on which the software is released. Certain restrictions are imposed on OSS licensing; an OSS license must not discriminate against any type of user group, field or endeavor [17]. In addition, an OSS license must be applied to all parties where the software is distributed, meaning that the Open Source distribution cannot be re-licensed by any user [17]. The open source software are current applicable in many fields. Internet programs, such as the mail transfer agent Sendmail, the Web server Apache, the operating system Linux, the database MySQL, and the office package OpenOffice, the popular languages PHP, the java integrated development environment Ecllipse are some of the most popular examples [18]. Comprehensive repositories for open source applications, which are already successfully competing with today's binary-only software (closed source software), are provided by the open source software development websites sourceforge [19] and freecode [20], the latter maintaining a large index of Unix and cross-platform software[18].

Open Source Software is produced by loosely organized, communities consisting of contributors from all over the world who seldom if ever meet face to face, and who share a strong sense of commitment [21]. Open source project communities begin when an individual or a group of individuals contribute an initial functional prototype of the software. People then gather around the prototype, with their own reasons and objectives, and work collaboratively to continue developing the software [22]. The number of Open source software (OSS) projects and the size of OSS systems has been growing at an exponential rate [23], becoming a significant part of the software economy. The persons starting the project usually become project leaders or administrators, and the project community grows as more and more persons are attracted into the project as developers [24]. Most contributors in OS projects are driven by a personal motivation, not directly linked to the size of the salary, but rather

factors of a more veiled nature. One explanation behind the personal motivation referred to in Maslow's hierarchy of needs as the category of self-actualization [25].

The following characteristics make OSS projects different enough from "conventional" software projects, making them interesting objects of study [26]:

- **Source code availability**. Source code of OSS projects is always available on the Internet, since most projects have a publicly-accessible version control repository. So all the interested user can access the code freely and can also modify the code.
- **User/developer symbiosis**. In most OSS projects the developers are also users of the software, and they also provide requirements. Maybe because of that, several free software projects do not have explicit requirement documents, and the development proceeds at a pace adequate for the developers to satisfy their own needs.
- **Non-contractual work**. A large amount of work in OSS projects is done in a non-contractual fashion. This does not imply that the developers are necessarily volunteers, but only that there is no central management with control over all of the developers' activities.
- **Work is self-assigned.** The absence of a central management with control over the contributors' activities promotes work self-assignment: volunteer developers tend to work on the parts of the project that most appeal to them, and employed developers will work on the parts that are of most interest to their employers.
- **Geographical Distribution.** In most OSS projects the developers are spread among several different locations in the world. In projects with high geographical dispersion, communication is mostly performed through electronic means.

Open Source Software Development

Open Source Software Development (OSSD) is the process by which open source software is developed within the confines of software engineering life-cycle methods [24]. The basic principle for the OSS Development Process (OSSDP) is that by sharing source code, developers cooperate under a model of systematic peer-reviews,

and take advantages of parallel debugging that leads to innovation and rapid advancement [5].

The term "Open Source Software Development" is a kind of distributed software development that has a large amount of contributors and because of using internet it is so successful and useful that developers can communicate over the distance [27]. It is a kind of distributed software development that use peer review technique in addition to make available source code for every volunteers who want to share their capabilities. Moreover due to putting so many people to develop, test and evaluate the code, OSSD gets more benefits rather than traditional closed source software. The OSS development team structure has a hierarchical or onion-like structure [24], which is shown in Figure 2.1. The focus of these studies has largely been on the contribution of code and they therefore have largely discussed development centralization.

At the center of the onion model are the core developers, who contribute most of the code and oversee the design and evolution of the project. In the next ring out are the co-developers who submit patches which are reviewed and checked in by core developers. Further out are the active users who do not contribute code but provide use-cases and bug-reports as well as testing new releases. Further out still, and with a virtually unknowable boundary, are the passive users of the software who do not speak on the project's lists or forums [24].

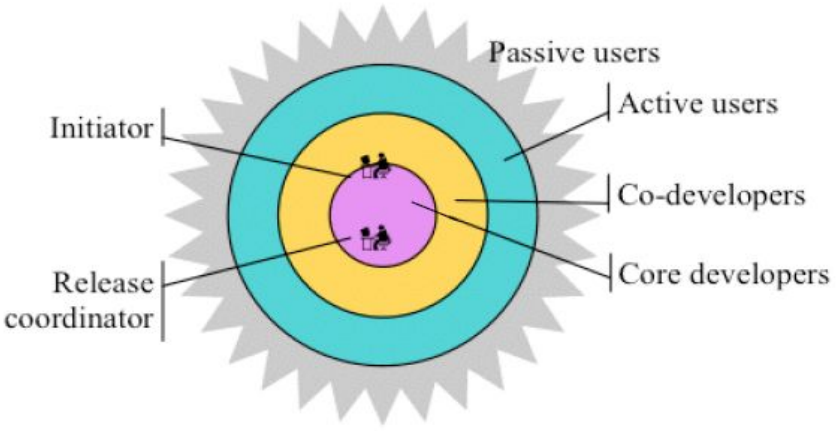


Figure 2.1 OSS Development Team Structure [24]

Advantages and Disadvantages of Open Source Software Development

The term Open source software development is significantly different from traditional software development in both organization and process [23].

Following table shows the advantages and disadvantages of open source software development over the traditional software development.

Table 2.1 Advantages and Disadvantages of OSSD [27]

ADVANTAGES		
For Users	**For Developers**	**For System**
Flexibility.	Allow to make own Solution.	Bug detecting.
Code availability.	Reuse many existing functionalities.	Reliability, Portability.
Ability to modify the code.	Allow to survey Problems freely.	Customizability.
Knowledge sharing.	Reduce damages in the beginning time of the system.	Rapid Evolution and extensibility.
Increasing motivation.	More motivation.	Reusability.
Participant in interested part	Support by many developers.	Sophisticated
Greater choice and control.	Work is self assigned.	Cost Effective.
DISADVANTAGES		
For Users	**For Developers**	**For System**
Unstructured development	Lack of Tools	Poor Design
Useless documentation	Collaboration with new developers	Version Proliferation
Irresponsible individuals	Review of large projects	Lack of Formal Process

2.2 Software Evolution

The term "Software Evolution" has been defined in the area of software engineering as the process of developing the software project initially, then repeatedly updating it for various reasons. The goal of software evolution research is to use the history of a software system to analyze its present state and to predict its future development, which made people, can better use and management software [28]. It is an activity related to programming that is used to generate the new version from the earlier version for various reasons.

With the approach of the new millennium, a primary focus in software engineering involves issues relating to upgrading, migrating, and evolving existing software systems [29]. Software Evolution is also defined as the dynamic behavior of programming systems as they are maintained and enhanced over their lifetimes. It is also defines as examining how the systems change overtime [30]. Lehman et al. have built the largest and best known body of research on the evolution of large, long-lived software systems [31]. Software evolution was born as field of research 40 years ago, and keeps being an intense matter of study. Meir M. Lehman, the pioneer in software evolution studies, coedited a book in 1985 where the laws of software evolution were stated [32]. Chris F. Kemerer and Sandra Slaughter (1999) focused on the processes, design, and structure of empirical research into software evolution [29]. For about thirty-five years now, Lehman has been concerned about, amongst other issues, software's long term health, in effect, beyond the next release, while most others – in practice and in research – have had "low beams" on as if the next release is the final release of the software product or system [33]. The review revealed a limited number of studies that have been largely focused in understanding and describing the dynamics of software evolution. Most published studies of software evolution have been performed on systems developed "in house" within a single company using traditional development and management techniques [34]. Michael W. Godfrey and Qiang Tu summarizes their preliminary investigations into the evolution of the best known open source system: the Linux operating system kernel and states that the growth is sub linear and it is useful to examine the growth patterns of the subsystems

to gain a better understanding of how and why the system seems to have been able to evolve so successfully. In 2002, Open Source Software (OSS) development is regarded as a successful model of encouraging "natural product evolution". Steven P. Reiss (2005) states software is changing and software evolution is going to change with it. Software evolution deals with the changes in a software system over time and Current trends point to a world where we only control a small fraction of our own software and the remainder evolves in unpredictable and uncontrolled ways [35]. It was believed that software evolution observations based on release history information opens up a whole area of research. Such structural information about a system is obtained relatively easily and valuable for the software engineers to identify potential shortcomings of their system [36]. Software systems need to evolve during the software life cycle by adding new features and to improve its quality. Recent studies on software evolution results in emphasizing that the statistical changes of the software system can be analyzed using its evolution metrics. Measuring the structural evolution of a software system has proven to be a straightforward effort that can easily be automated [37]. Lee, Yang and Auburn (2007) provide an overview of open source software evolution with software metrics, and present various metrics that can be used during the software evolution process to understand the evolution behavior [13]. Israel Herraiz (2009) suggests using SLOC, complexity metrics to characterize a software product and its evolution [38]. Breivold, Chauhan and Ali Babar (2010) examined the metrics used for assessing OSS evolution, and defined various type of metrics [39]. The metrics has been characterized in three major categories as Software Growth metrics, Complexity Metrics and Modularity Metrics. Numbers of packages, number of classes, source lines of code, number of methods are few of the growth metrics, McCabe's Cyclomatic Complexity is one of the complexity metric and number of classes, number of files for each release are the modularity metrics which are used to define the software growth pattern thus helps in measuring the evolution of the software. These metrics are also used to study the Lehman's Laws of Evolution. A software quality model for evolution includes the measurement of the properties of stability, analyzability, changeability and testability as sub-characteristics of a software product [40].

Open Source Software is a type of software system which is free to use and the source code of that software is freely available and accessible to the entire interested user for reading, and modifying. Most of the open source software starts with the developer need to solve some problem and make the system available to others also for free, some of the users gets interested to the system and become the co-developers by improving and enhancing the system as the code is freely accessible and thus together with initiator, users and co-developers creates a collaborative OSS community around the system. Most of the software has been not designed well in advance but the evolution due to needs of the users in the community results in a successful system thus the system evolves very fast. Thus the evolution of such type of the systems is carried out by the contribution of the co-developers of the well established and maintained community of the system. So OSS development work is regarded as the successful model of encouraging "natural product evolution". The evolution in this type of system is although not well planned but the access to source code results in the derivative works which allows enhancing the system and results in natural product evolution.

2.3 Software Metrics

The best motivation for metrics comes from a quotation by Lord Kelvin [41]:

"When you can measure what you are speaking about, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts, advanced to the stage of science."

There are two components touched upon in the above sentence: First is that metrics are measurements to measure your progress (in – process metrics). The second is the measurement of how well you have achieved the goals (end – result metrics) [41].

Software Metrics are intended to measure the software projects and their characteristics quantitatively, encountered during the planning and execution of software development. Sedigh et al, (2001) states "Metrics can also be used in

guiding decisions throughout the life cycle, determining whether software quality improvement initiatives are financially worthwhile" [42].

A lot of research has been conducted on software metrics and their applications. Most of the metrics proposed in literature so far are based on the source code of the application**.** It is known that the impact of any software metric is greatest when used in the early stages of the software development cycle when it is more feasible for changing and modifying the product under development [43].

Commonly Used Software Metrics

- **Size Metrics**

  A number of metrics attempt to quantify software size. Some of them are discussed here:

  - **Lines of Code.** LOC is the most widely used metric for program size. It seems to be easily and precisely definable. The most common definition of LOC seems to count any line that is not a blank or comment line, regardless of the number of statements per line.
  - **Function Points.** Function points are intended to be a measure of program size and thus, effort required for development.

- **Complexity Metrics**

  Numerous metrics have been proposed for measuring program complexity. Some of the better known complexity metrics [44] are discussed below.

  - **Cyclomatic Complexity**. McCabe (1976) proposed a complexity metric called Cyclomatic Complexity. McCabe proposed that Cyclomatic Complexity can be used as a measure of program complexity and hence, as a guide to program development and testing.
  - **Information Flow** - The information flow within a program structure may also be used as a metric for program complexity.

- **Quality Metrics**

  One can generate long lists of quality characteristics for software- correctness, efficiency, reliability, maintainability etc [45].

Some of the quality Metrics by Boehm, McCall and others are discussed below:

- **Defect Metrics**. The number of defects observed in a software product is a metric of software quality.
- **Reliability Metrics.** Reliability refers to the probability of software failure, or the rate at which the software failure will occur. Mean Time to Failure (MTTF) is reliability metric.
- **Maintainability Metrics**. According to experiments done by Rombach, software complexity metrics can be used to predict the maintainability of software.

2.4 Mining Software Data Sources

The work in this thesis makes use of data about software that is extracted from various sources. All possible type of data gathered from these resource are not necessary for this work, but the open software has become a rich provider of many sources quite apart from the availability of source code. The availability of open software source code helps in deriving many measures but in addition to this, open software projects typically provide public access to their mailing lists, discussion forums, bug databases, as well as their entire repository system, from each of which, like the source code, other measures may be derived.

The work involves to extract the information from various historical data sources to draw conclusions about specific attributes of software, best characterized by German Daniel who argued in favors of the role of the "software evolutionist" equating it to a paleontologist or private investigator who must follow trails left by contributors to a project, piece them together, and use them to prove or disprove their original hypotheses [46].

Mining Software Repositories (MSR) is a very active and interest-growing research field deals with retrieving and analyzing the retrieved data [2011]. The software repositories have been explored to collect the data and this mined data has been used by the researchers to draw relationship among the various attributes of the software.

Various Mining Aspects

Source Code

Among the various artifacts of the software projects the most important among them from the point of view of various software engineering researches is the source code of the software projects. This was used by researchers to derive various characteristics and attributes of software product. By counting lines of code from software source code anyone can find the idea behind length of a program. This measurement is again used to derive many more complex metrics. Some of the examples include counting lines of executable statements, line of comment, and ratio of comments to executable statements.

The source code analysis method also helps to derive the evaluation of complexity of program. Metrics derived from this process include McCabe's measure [44]. Source code is also used to find the structural aspects of the software design and also to find the interconnecting modules.

Version Control System

Version control (VC) systems manage multiple versions of the same artifact. Software projects typically use a VC system to manage source code, configuration files, documentation, and so on and also helps in coordinating a group of people that work concurrently to achieve a shared objective [47]. In this way, version control keeps a historically accurate and retrievable log of a file's revisions. More importantly, version control systems help several people (even in geographically disparate locations) work together on a development project over the Internet or private network by merging their changes into the same source repository. The reason version control is so universal is that it helps with virtually every aspect of running a project: inter-developer communications, release management, bug management, code stability and experimental development efforts, and attribution and authorization of changes by particular developers [48].

The core of version control is change management: identifying each discrete change made to the project's files, annotating each change with metadata like the change's date and author, and then replaying these facts to whoever asks, in whatever way they ask [48]. Some works have chosen to use relatively straightforward and easily obtained metrics from repositories, and were able to make interesting findings about the software project under the version control system. Other works have used these simpler metrics to compliment more complex ones that reveal more than is possible with such directly derived metrics. These include judging the type of commit (e.g. an addition of functionality or a bug fix or a comment change etc.) [49], judging modules with strong logical couplings because of frequent coincidental change [50], or reconstructing the structure of developer communities [51]. These days, everyone will expect at least the project's source code to be under version control, and probably will not take the project seriously if it doesn't use version control with at least minimal competence [48]. The quality of evolution research that can be performed is constrained by the versioning system used by the developers of a system, as each versioning system generates only certain types of information [52]. There are many version control systems available in the market. Following are some of the features of a version control system called Subversion.

- Subversion is a free/open source version control system (VCS). Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of "time machine." [53]. "Subversion is an open source version control system" that by default uses a command line user interface on the client side [54].
- Subversion is a full-featured version control system originally designed to be a better version control system. Subversion makes life so much simpler, allowing each team member to work separately and then merge source code changes into a single repository that keeps a record of each separate version [55].

2.5 Tool Support

Mining software artifacts generally involve software tool that is used to automate the process due to the voluminous data processing, and there are such tools available in both free and non-free forms. A small subset of tools with special relevance to the work in this thesis is described here.

TortoiseSVN [56]

TortoiseSVN is a Subversion (SVN) client, implemented as a windows shell extension. It's intuitive and easy to use, since it doesn't require the Subversion command line client to run. Simply the coolest Interface to (Sub) Version Control.

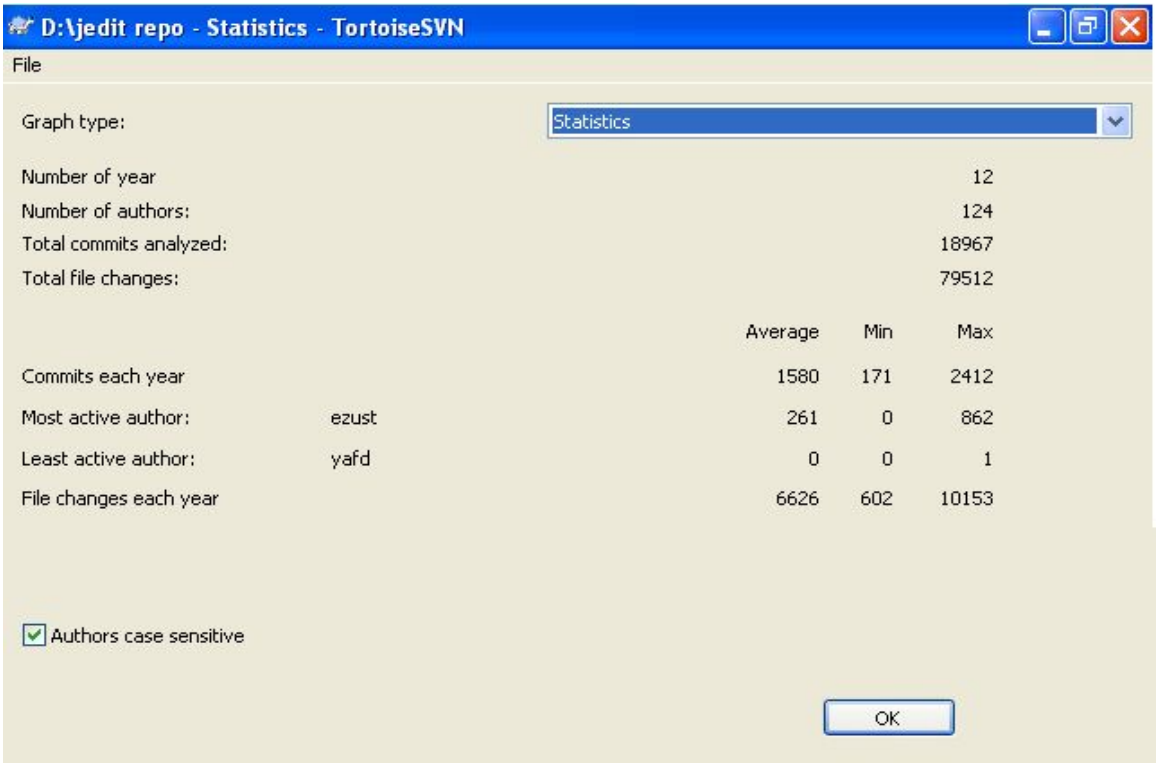


Figure 2.2 TortoiseSVN showing the statistics of JEdit

LocMetrics [57]

LocMetrics (http://www.locmetrics.com) counts total lines of code (LOC), blank lines of code (BLOC), comment lines of code (CLOC), lines with both code and comments (C&SLOC), logical source lines of code (SLOC-L), McCabe VG

complexity, and number of comment words (CWORDS). Physical executable source lines of code (SLOC-P) are calculated as the total lines of source code minus blank lines and comment lines.

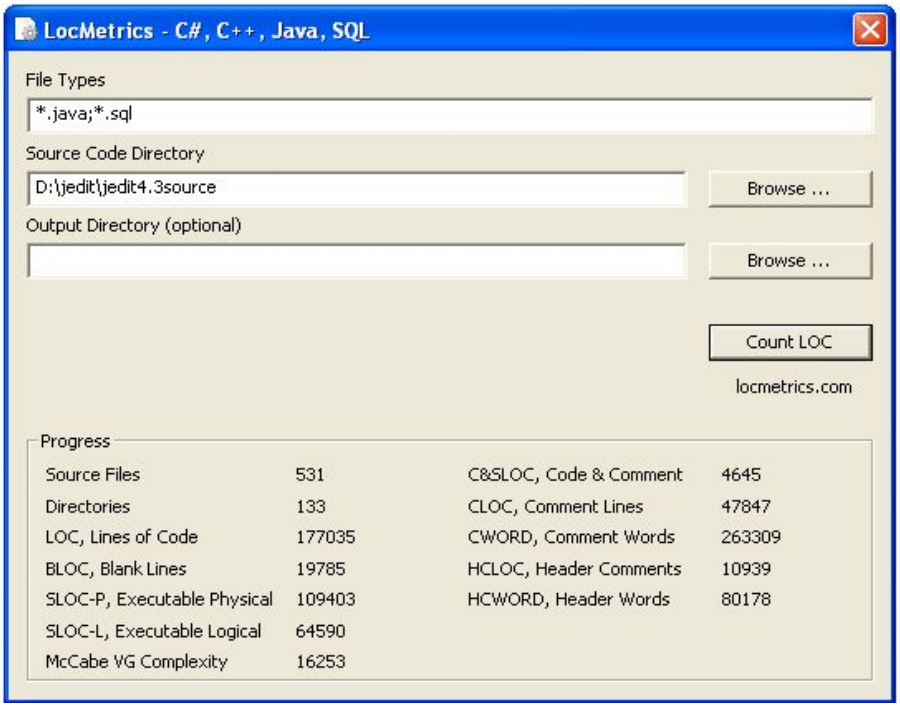


Figure 2.3 LocMetrics showing the measurement of JEdit4.3

Counts are calculated on a per file basis and accumulated for the entire project. LocMetrics also generates a comment word histogram.

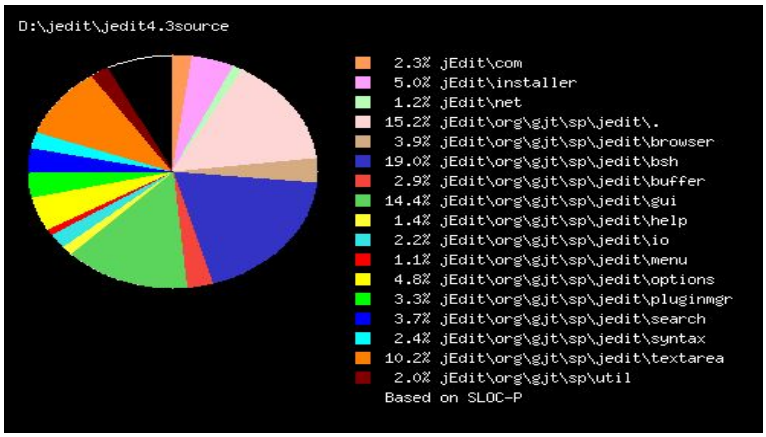
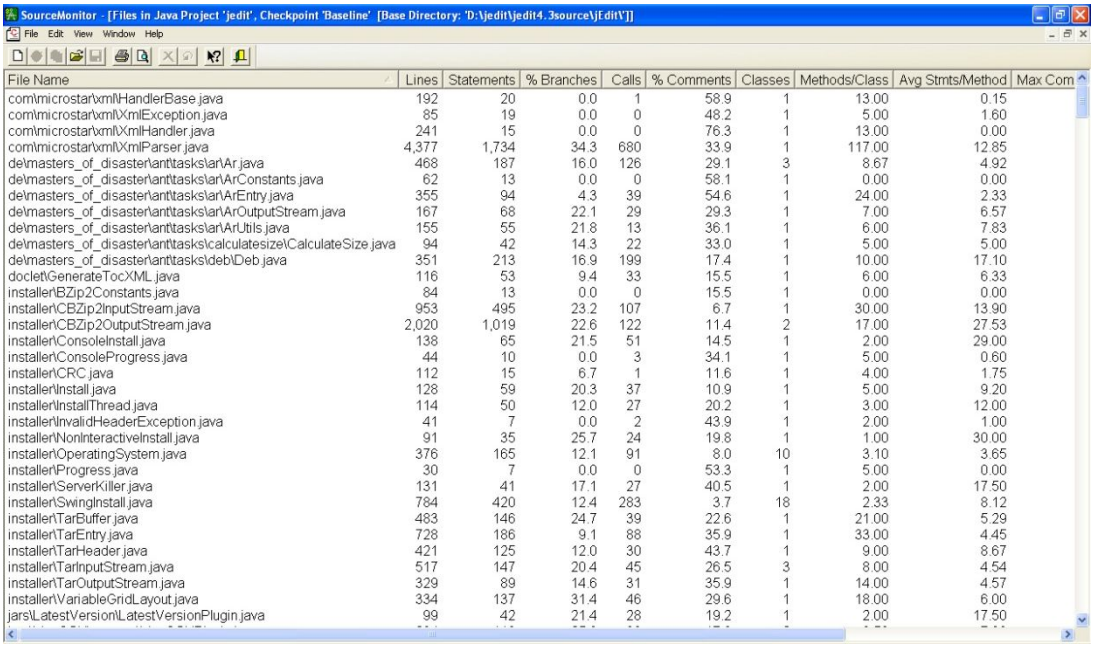


Figure 2.4 Histogram Using LocMetrics

SourceMonitor [58]

The freeware program SourceMonitor see inside the software source code to identify the relative complexity of your modules. SourceMonitor, written in C++, runs through your code at high speed. SourceMonitor provides metrics in a fast, single pass through source files. It also measures metrics for source code written in C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6) or HTML. It also helps in saving metrics in checkpoints for comparison during software development projects. The program displays and prints metrics in tables and charts and operates within a standard Windows GUI or inside your scripts using XML command files. It also exports metrics to XML or CSV files for further processing with other tools.



File Name	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Com
com\microstar\xml\HandlerBase.java	192	20	0.0	1	58.9	1	13.00	0.15	
com\microstar\xml\XmlException.java	85	19	0.0	0	48.2	1	5.00	1.60	
com\microstar\xml\XmlHandler.java	241	15	0.0	0	76.3	1	13.00	0.00	
com\microstar\xml\XmlParser.java	4,377	1,734	34.3	680	33.9	1	117.00	12.85	
de\masters_of_disaster\ant\tasks\ar\Ar.java	468	187	16.0	126	29.1	3	8.67	4.92	
de\masters_of_disaster\ant\tasks\ar\ArConstants.java	62	13	0.0	0	58.1	1	0.00	0.00	
de\masters_of_disaster\ant\tasks\ar\ArEntry.java	355	94	4.3	39	54.6	1	24.00	2.33	
de\masters_of_disaster\ant\tasks\ar\ArOutputStream.java	167	68	22.1	29	29.3	1	7.00	6.57	
de\masters_of_disaster\ant\tasks\ar\ArUtils.java	155	55	21.8	13	36.1	1	6.00	7.83	
de\masters_of_disaster\ant\tasks\calculatesize\CalculateSize.java	94	42	14.3	22	33.0	1	5.00	5.00	
de\masters_of_disaster\ant\tasks\deb\Deb.java	351	213	16.9	199	17.4	1	10.00	17.10	
doclet\GenerateTocXML.java	116	53	9.4	33	15.5	1	6.00	6.33	
installer\BZip2Constants.java	84	13	0.0	0	15.5	1	0.00	0.00	
installer\CBZip2InputStream.java	953	495	23.2	107	6.7	1	30.00	13.90	
installer\CBZip2OutputStream.java	2,020	1,019	22.6	122	11.4	2	17.00	27.53	
installer\ConsoleInstall.java	138	65	21.5	51	14.5	1	2.00	29.00	
installer\ConsoleProgress.java	44	10	0.0	3	34.1	1	5.00	0.60	
installer\CRC.java	112	15	6.7	1	11.6	1	4.00	1.75	
installer\Install.java	128	59	20.3	37	10.9	1	5.00	9.20	
installer\InstallThread.java	114	50	12.0	27	20.2	1	3.00	12.00	
installer\InvalidHeaderException.java	41	7	0.0	2	43.9	1	2.00	1.00	
installer\NonInteractiveInstall.java	91	35	25.7	24	19.8	1	1.00	30.00	
installer\OperatingSystem.java	376	165	12.1	91	8.0	10	3.10	3.65	
installer\Progress.java	30	7	0.0	0	53.3	1	5.00	0.00	
installer\ServerKiller.java	131	41	17.1	27	40.5	1	2.00	17.50	
installer\SwingInstall.java	784	420	12.4	283	3.7	18	2.33	8.12	
installer\TarBuffer.java	483	146	24.7	39	22.6	1	21.00	5.29	
installer\TarEntry.java	728	186	9.1	88	35.9	1	33.00	4.45	
installer\TarHeader.java	421	125	12.0	30	43.7	1	9.00	8.67	
installer\TarInputStream.java	517	147	20.4	45	26.5	3	8.00	4.54	
installer\TarOutputStream.java	329	89	14.6	31	35.9	1	14.00	4.57	
installer\VariableGridLayout.java	334	137	31.4	46	29.6	1	18.00	6.00	
jars\LatestVersion\LatestVersionPlugin.java	99	42	21.4	28	19.2	1	2.00	17.50	

Figure 2.5 SourceMonitor showing metrics value for JEdit

OpenStat [59]

OpenStat is a powerful and free statistical package that was originally written to help in social science research. Nowadays OpenStat has been expanded to handle all kinds of data. OpenStat, rather than just present data in graph form, attempts to test and display data in one go, meaning it cuts out the need for a separate research package.

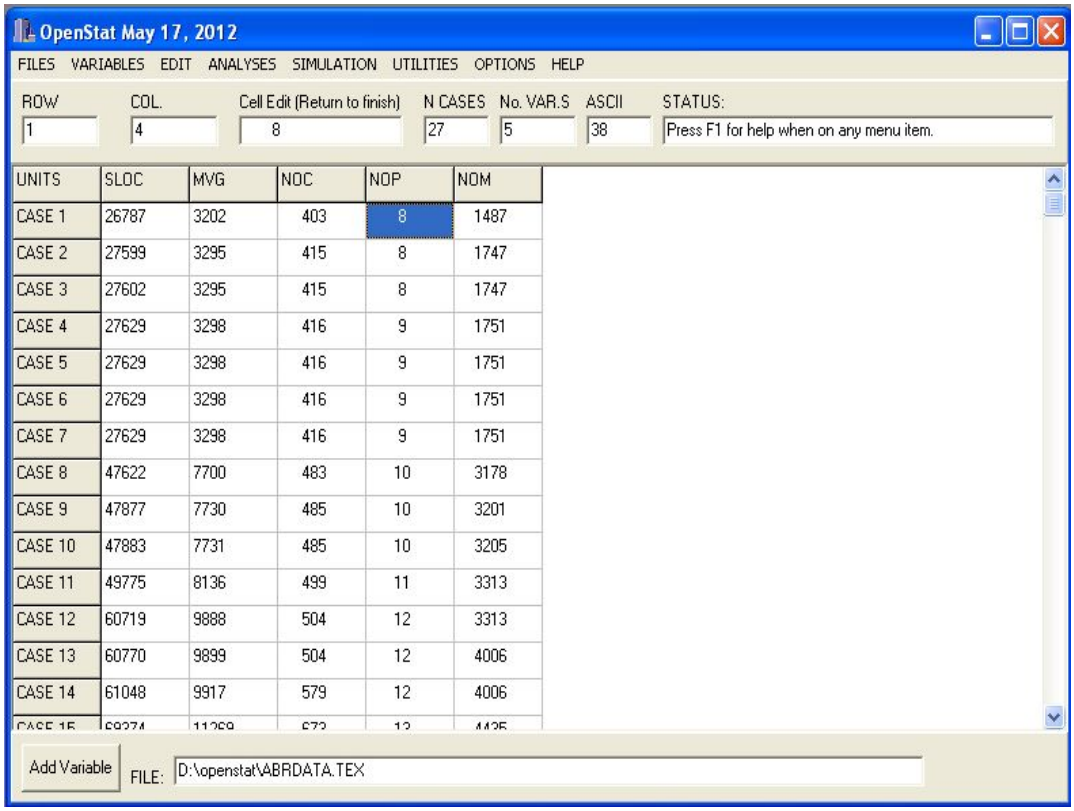


Figure 2.6 OpenStat

It looks remarkably simple and generally, it is easy to use but it's actually quite a complex package, able to handle all kinds of algorithms and parameters that you define. However, the developer could have put more effort into making both the interface and graphs look better. OpenStat is simple to use but powerful statistical analysis and graphical display package that can handle most of what you throw at it. It is useful as it can analyze and displays results in one go but its interface is very limited.

Chapter 3

PROBLEM STATEMENT

3.1 Problem Definition

The development and evolution of Open Source Software components is based on the ability of programmers to freely use the source code to enable adaptations and improvements of the original software. Open Source Software is software being developed under a license compatible with OSI licenses.

Open Source Software Development process leads to following advantages of OSS:

1. OSS is publicly available, and the parallel distribution enables faster development.
2. OSS components can be as reliable, efficient and robust as their conventional cousins, if not more so.
3. OSS holds the potential to avoid the threat of vendor instability or "lock-in" on support of maintenance and further evolution.
4. The parallel development efforts towards OSS allow for faster updates and bug fixes.
5. No additional licensing is required towards additional installations of the same software.

Gregorio Robles-Martínez, Jesús M. González-Barahona [60] states that “Many works has still to be done to enhance the methodology itself, in order to obtain more insight on the process of creating open source software. In this respect, although various research papers give an idea of a methodological approach, it is in fact very limited. Future research should include many other parameters that have not been taken actively into account yet. For instance, the use of complexity measures (McCabe) is one of the considerations that are missing and should be introduced. From other point of view, more specific correlations should be studied so that

characterization of a project from some points of view would permit the inference of other information more difficult to obtain.

As this thesis partly considers the changes in the values of software attributes over time the subject of software evolution has been reviewed. Important background works by Lehman and others deliver some fundamental governing principles of software evolution, but it is highlighted that, in the opinion of some authors, software evolution has been the victim of a general paucity of strongly empirical studies. The literature review has identified that software evolution research has diversified and that studies more recently have branched into examining the subject from the perspective of the development paradigm. These perspectives include the open software paradigm, and when the discussion moved to works on open software evolution, it was demonstrated how this particular paradigm possesses evolutionary characteristics specific to itself. It has therefore been shown very strongly that open software evolution is a credible and promising avenue of investigation, and furthermore that the reports of past authors urge that studies should include a suitable realistic projects from which to draw their observations. The Evolution process can be well analyzed using the Metrics and visualizing the metrics approach to get the relevant information. So this thesis will going to propose a metric driven approach to predict the evolution of the software which results in predicting the future and analyzing the present behavior of the projects. The Software evolution thus helps in maintaining the quality of the software.

So the literature review results in defining the problem. The well-defined problems are:

- To identify the metrics to measure development efforts and complexity of the software project.

- To evaluate, measure and compare the various development and complexity metrics of a real time open source project.

- Using the software repository data, studying the software evolution patterns of open source software.

- To prove the basic Lehman's laws of software evolution by using the measurement of the open source software attributes.

3.2 Justification

The review has also identified that software evolution research has diversified. The software evolution research has been majorly done on the basis of development paradigm but here in this thesis the evolution is studied on various paradigms like development, complexity, and maintainability. Software metrics have been thoroughly studied and explored for several features of the Open Source System. These features majorly include growth, complexity and maintainability. The thesis proposed quantitative evaluation of complexity and growth metrics for open source system and subsystem for a real time open source project called jEdit. Various tools are explored, reviewed and compared to check their validity of results. A correlation analysis has been conducted to establish the relationships between these metrics in the area of open source software projects. The growth patterns are then plotted on graph and evolution of the system is studied thoroughly. Finally the measured data from repositories of the project helps in evaluating the Lehman's Laws of Software evolution which has been proved in closed source development projects but still a topic of research in the open source software development. Various works has been done in software evolution but this thesis uses the metrics visualizing approach to find the growth patterns of the open source software project and hence predicting the software evolution.

Chapter 4

Measuring the Evolution of Open Source Software

4.1 Proposed Software Evolution Process

The evolution of Open Source Software components is based on the ability of programmers to freely use the source code to enable adaptations and improvements of the original software.

4.1.1 Tools used in the process

The proposed methodology is based on the use of a versioning system called SVN (Subversion). It is widely accepted version control system because of its better performance than other version control system like CVS. There are a lot of projects hosted at Sourceforge which uses this versioning system like jEdit, Mozilla, and Eclipse etc. Various tools have been available in the open source software forums which help in achieving the given objectives but out of them a set of tools have been chosen. The core tools used to study the data in the SVN repository are the following:

- **Tortoise SVN.**

  It is used to calculate the statistics regarding the Software Project under the supervision of the Subversion version control system. It helps in maintaining and managing the repositories and also helps in finding the various project characteristics using show log command.

- **LocMetrics.**

  LocMetrics counts total lines of code (LOC), blank lines of code (BLOC), comment lines of code (CLOC), lines with both code and comments (C&CLOC), logical source lines of code (SLOC-L), McCabe VG complexity (MVG), and number of comment words (CWORDS).

- **OpenStat.**

  OpenStat is a simple to use but powerful statistical analysis and graphical display package that can handle most of what you throw at it. OpenStat is a powerful and free statistical package that was originally written to help in social science research. Here it is used to study the correlation among the various software attributes.

- **SourceMonitor.**

  SourceMonitor computes metrics in a fast, single pass manner through source files. It also measures metrics for source code written in C++, C, C #, VB.NET, Java, Delphi, Visual Basic (VB6) or HTML.

4.1.2 Metrics used in the process

This section helps in presenting the definition of various growth and complexity metrics used in the study. The metrics used to study the data in the SVN repository are the following:

- **SLOC :** Physical Lines of Source Code

  Source Lines of Code also known as physical lines of code is the calculated using the three metrics that are lines of code, blank lines and lines of comment.

  $$SLOC = LOC – CLOC – BLOC$$

  LOC here means lines of code, CLOC means comment lines of code, and BLOC means blank lines of code. SLOC is considered as a good metric as good as the number of files for software evolution analysis. So SLOC can be used which defines the source lines of code.

- **NOC :** Number of Classes

  NOC for a project here defines the number of classes used to implement the project requirements. Number of classes is a good metric to object oriented software systems.

- **NOM :** Number of Methods

  NOM for a project here defines the number of method defined in the all the packages.

- **MVG :** McCabe's Cyclomatic Complexity

  MVG is used to calculate the complexity value of the software project which is used to predict the maintenance level and testing requirement of the project.
- **CLOC :** Comment Lines of Code

  Lines of Comment are defined as the total number of comment line used by the developers in the software project source code. It helps in predicting the readability and understandability of the code and helps other developers in the OSS society to learn about the code easily.
- **NOP :** Number of Packages

  NOP for a project here defines the number of packages used to implement the project requirements.
- **NOF :** Number of Files

  NOF for a project here defines the number of source files used to implement the project requirements.

4.1.3 Software Evolution Measuring Process in Open Source Software

The proposed process to measure and study the evolution of the open source software consists of some steps and flowchart of the process steps is given in Figure 1.4. The software evolution is a process which helps to visualize the development efforts and all the other attributes of the software under study. Software evolution can also be defined as the study of the statistical changes of software system using set of evolution metrics. Following are the steps to the proposed process.

1. **Selection of the evolution areas.**

   Evolution of open source software has been analyzed using the different areas of the software development. Anyone can study the evolution in terms of the growth of the software under study, quality variation, and maintenance etc. The evolution study helps also in predicting the maintenance required to achieve a standard quality level so it helps in the

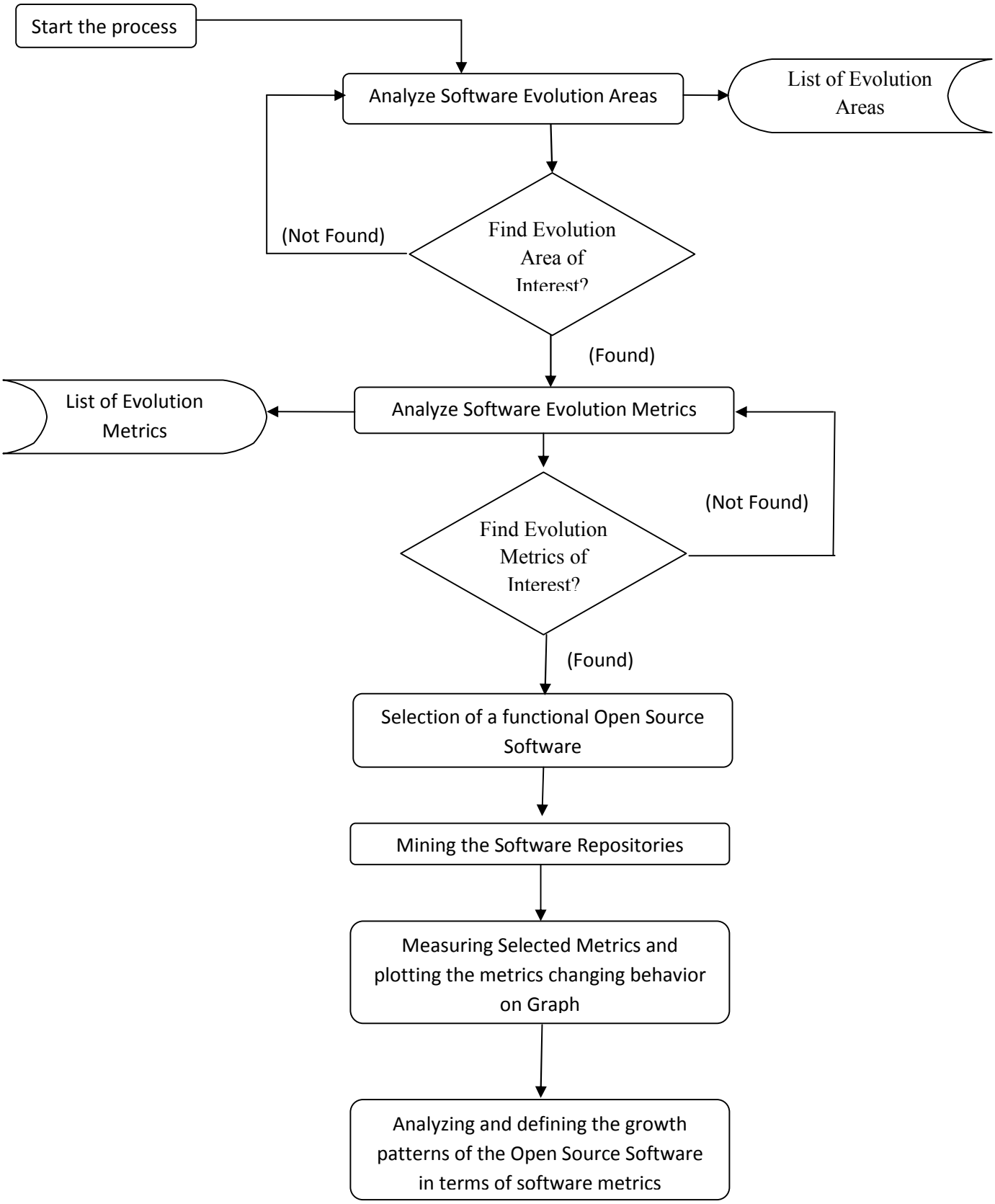


Figure 4.1 Metrics Driven Open Source Software Evolution Process

software future prediction. So the researchers have to decide and analyze the field where the evolution of the software has to be study. Here in this thesis the software is studied for its growth and also the maintenance using the software attributes measurement. List of the evolution areas can be maintained.

2. **Selection of the set of evolution metrics.**

   The area of study once decided and analyzed helps in deciding the attributes to be studied. These attributes once studied helps in finding the metrics which measures these attributes quantitatively which is better way to understand and visualize the software attributes changes. For study of growth SLOC is the best metrics, for study of maintenance complexity metric can be used, for study of quality the coupling/cohesion can be used. Here in this thesis since the areas are of growth and maintenance so the attributes results in selection of growth and maintenance metrics. List of evolution metrics can be maintained.

3. **Selection of the appropriate open source software.**

   Now when the areas and the metrics list have been finalized, the study should move on deciding the appropriate software which has fulfilled all its requirements and also have an appropriate number of versions so the evolution process can be studied properly. In this thesis after studying a lot of open source software the study finalized the jEdit. The jEdit is a java based text editor used by maximum number of users and have more than 12 years of development and still everyday it is going to be modified.

4. **Mining the software repositories of that software using set of tools.**

   Set of tools have been then analyzed and compared to finally get the best set used to extract the software repositories to get the data used to predict the evolution. Version control data has been extracted to get it converted to evolution data. The set of tools required consists of version control

system tool, software metrics calculation tools, statistical analysis tool. Version control tool is used to find the source code and changes in the source codes, number of packages, number of commits, statistics of the software, number of developers etc.

5. **Analyzing and visualizing the evolution data.**
   The data has to be converted to some valuable information then compared and correlation has been calculated and properly analyzed. The Graph can be drawn so the visualization gets easy. Graphically viewing the software data is best way to predict the variations in the evolution process and thus also helps in solving and predicting various analytical questions. The data can be compared using the product moment correlation method and then stated either the metrics are correlated or these metrics are not correlated which help us in stating that growth helps in increasing or decreasing the maintenance of the OSSD.

4.2 Case Study: jEdit

Now that the methodology for Open Source Software evolution has been defined, it needs to be tested on real software systems. jEdit[61] were chosen as the target software category, because this relatively new area of software where Open Source alternatives are now starting to break through, is not explored fully at this time, and it concerns a reasonably large sized software. The target users for these systems are educational institutions, including universities, various software developers who need good editor to perform their work.

First, the selection of evolution areas has been done which include the development effort and maintenance effort of the open source software. Here the thesis results in studying evolution on the selected areas and set of metrics are also selected on the above specified areas. The thesis is going to work on the development and the complexity areas and a set of metrics has been selected for studying the given areas.

4.2.1 Introduction to jEdit

jEdit is a mature programmer's text editor with hundreds of person-years of development behind it. The jEdit project was originally started by Slava Pestov. The motivation was to create an open source, high quality, text editor for programmers.

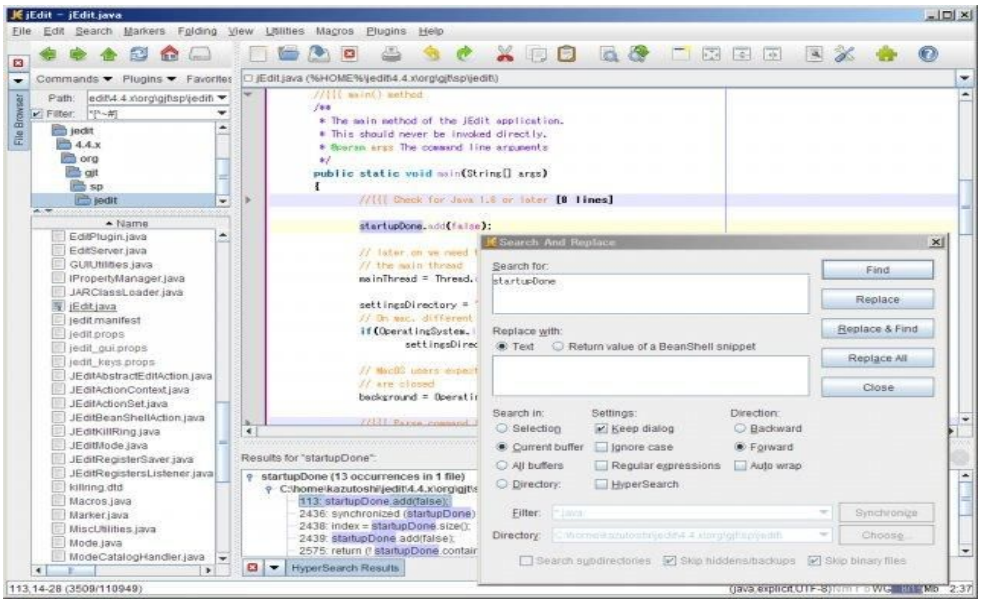


Figure 4.2 jEdit 4.4pre1 on Windows XP

While jEdit beats many expensive development tools for features and ease of use, it is released as free software with full source code, provided under the terms of the GPL 2.0. The project moved to SourceForge from the Giant Java Tree in early 2000.

Some of jEdit's features include:

- Written in Java, so it runs on Mac OS X, OS/2, Unix, VMS and Windows.
- Built-in macro language; extensible plugin architecture. Hundreds of macros and plugins available.
- Plugins can be downloaded and installed from within jEdit using the "plugin manager" feature.
- Auto indent, and syntax highlighting for more than 200 languages.

- Supports a large number of character encodings including UTF8 and Unicode.
- Folding for selectively hiding regions of text.
- Word wrap.
- Highly configurable and customizable.

4.2.2 Selection

The Open Source Software to be chosen for studying the metric evolution process must pass the following selection criteria.

The proposed selection criteria are:-

- **Functionality.** The software needs to have the basic functionality to conform to the functional requirements that were defined.
- **Release Activity.** The last stable release needs to be no older than one year.
- **Number of Versions.** The software must have more than 15 releases over a long period of time so that open source software development evolution process must have broader area of data.

jEdit under selection criteria:-

- **Functionality.** The jEdit is a text editor which provides all the basic functionality to conform to the functional requirements that were defined. The user rating for this text editor is 78% and has been successful more than 12 years of development and many developers get added to it in this span of time.
- **Release Activity.** The last stable release of jEdit under study is jEdit 4.5.1 and release date is 27th March, 2012. So the last stable release is not less than one year.
- **Number of Versions.** The jEdit under study have more than 30 releases over a long period of time. This thesis studies is over the 27 versions of the jEdit.

4.2.3 Evolution of jEdit using Version Control data

This section presents an analysis of the evolution of an open source software system, jEdit, which is an open source text editor, based on its size, complexity, and modularity metrics. The jEdit software is open source software which is under GNU license and whose source code can be read by any user needed to study it.

Mining the repository of jEdit using version control system

JEdit is a java based text editor. It is implemented in java. It is a programmer's text editor written in java. Since the registration of jEdit on sourceforge on 6th December 1999, and then there has been 34 official releases till date for which source is available. Study has considered jEdit2.3Final as the initial versions registered on 11th March 2000, of software. jEdit 5.5.1 is a result of major revisions applied to previous versions of jEdit. The latest version is jEdit 4.5.1. Each release of jEdit is a result of set of revisions made to it in response to bug fixation, feature request, adaption to new environment or preventive activity. The software has gone through 21738 revisions till 31st of May 2012. There are 126 active authors who helped in development of the jEdit releases. The software has 12 successful years of development and user rating of 78% satisfaction level. Mining the repository of the jEdit helps in finding the source code of the project.

Table 4.1 gives the details of releases of jEdit over the period. In order to conduct the study with the said objective there is requirement of source code, measure of several metrics and revision details.

The software used in the study is open source software. Therefore the availability of source code for different versions was never a problem. The computation of different metrics was done using the defined tools meant to compute metrics.

The revision details of each release of software are maintained in the subversion repositories. The revision details of jEdit are available at []. The software has gone through 21738 revisions till date and has 12 successful years of development.

As discussed in section 1.2.3 the following views have been analyzed:

- **Requirements.** Build a programmers editor.
- **Developers.** Team of 126 developers who have to coordinate with each other.
- **Implementation technology.** The Java language.
- **Software process**. Open Source Software Development, many of the program components built in parallel and distributed development.

Table 4.1 Details of releases of jEdit

Version	Release Date	Version	Release Date
2.3	2000-03-11	4.0	2002-04-12
2.4	2000-04-23	4.0.2	2002-06-15
2.4.1	2000-04-23	4.0.3	2002-06-15
2.4.2	2000-04-26	4.1	2003-05-24
2.5	2000-07-08	4.2	2004-12-01
2.5.1	2000-07-31	4.3	2009-12-23
2.6	2000-11-04	4.3.1	2010-01-28
3.0	2000-12-25	4.3.2	2010-05-10
3.0.1	2001-01-23	4.4.1	2010-12-01
3.0.2	2001-01-25	4.3.3	2011-06-21
3.1	2001-04-22	4.4.2	2011-10-13
3.2	2001-08-29	4.5.0	2012-01-31
3.2.1	2001-09-02	4.5.1	2012-03-27
3.2.2	2001-09-02		

jEdit development process details using version control system

- **Statistics of jEdit development**

Table 4.2 Statistics of jEdit development

Number of Years of Development			12
Number of authors involved in development			126
Total number of commits analyzed			19198
Total number of files changed during development			80066
Most active author			Ezust
Least active author			Yafd
	Average	Minimum	Maximum
Commits each year	1599	171	2412
Most active author	267	0	862

- **Commits by author.**

  The author is the one who makes the changes, and add those changes to the repository by using commit command. The jEdit has 126 authors who contribute in the development of the project and the Figure below show the 20 most active authors who perform actively during the 12 years of development.

- **Commits by dates.**

  The "Commits by date" gives a nice overview of authors activity over time and the Figure below show the 10 most active authors who perform actively during the 12 years of development and shown the commits by them in every year of development.

- **Percent of authorship.**

  The percent of authorship is a metric to answer the question which person should be consulted to understand/ fix/improve code. Thus help in finding the developers having the knowledge of the software and its developing process

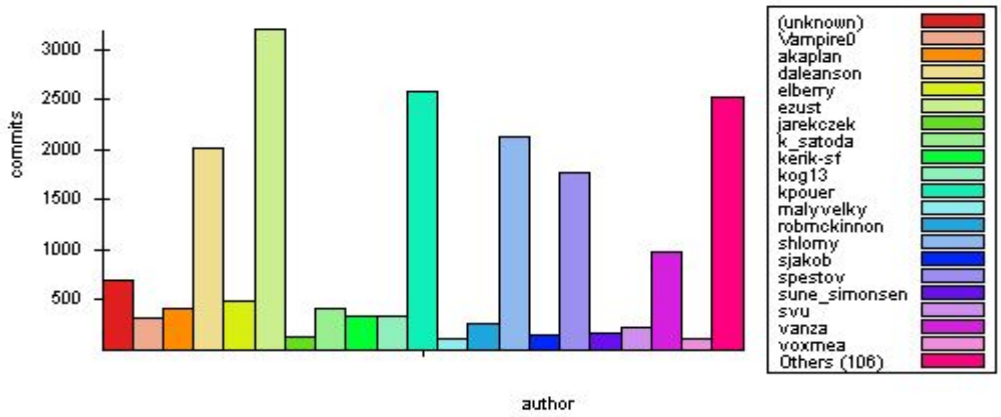


Figure 4.3 Commits by 20 most active authors in jEdit Development

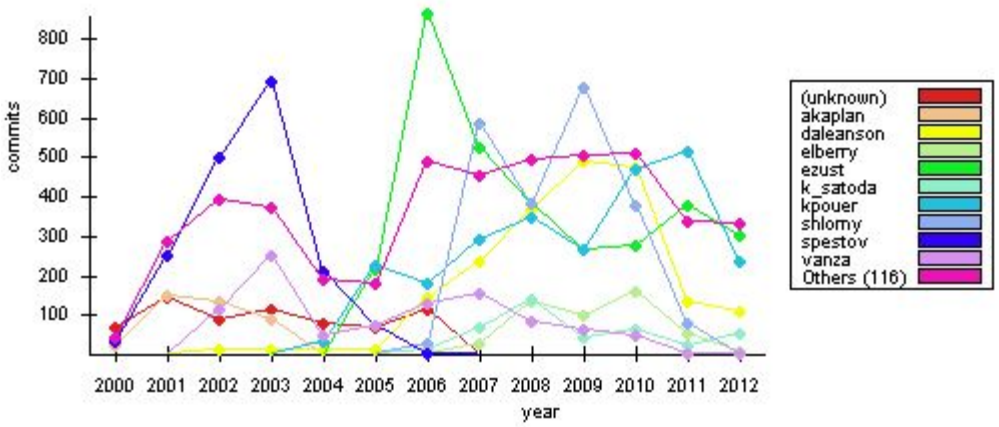


Figure 4.4 Commits by date by 10 most active authors in jEdit Development

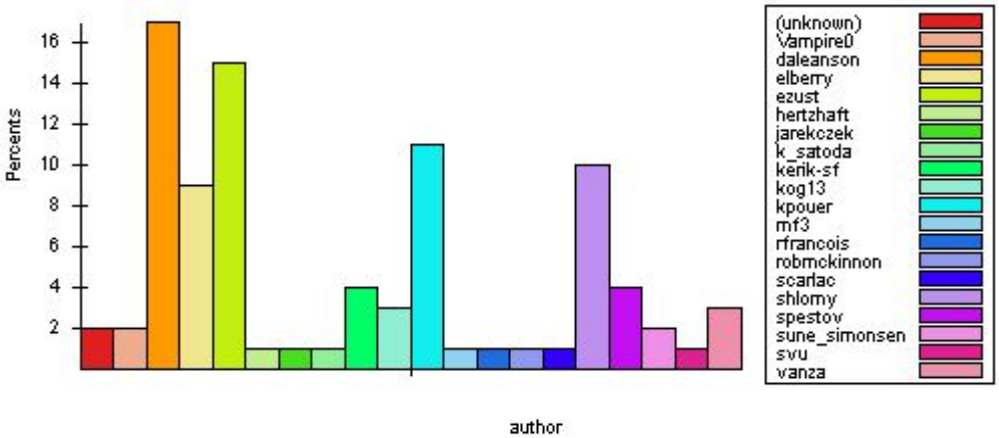


Figure 4.5 Percent of Authorship by 20 most active authors in jEdit Development

Metrics calculation using version control data

In order to conduct the study with the said objective the source code using the version control tool has been extracted and a local repository in the system has been created. The selected set of metrics has been calculated for all the twenty seven versions of the selected open source software called jEdit. The metrics are classified into two groups called the size metrics and complexity metric.

- **Size metrics**

  The metrics results in quantitatively measuring the size of the software. This study involves LocMetrics as a tool used to calculate the physical source line of code (SLOC), lines of comment (CLOC), number of source files (NOF). SourceMonitor tool is used here to calculate the given metric as number of classes (NOC), number of methods (NOM). All the twenty seven versions at the package level is analyzed and then collectively added to get the metrics at the full version level. Number of packages (NOP) is also calculated using the TortoiseSVN repository of jEdit created locally at the system. Table 4.3 show the calculated value of size metrics for the twenty seven releases of the jEdit.

Table 4.3 Size Metrics over jEdit releases

Releases	SLOC	CLOC	NOF	NOP	NOC	NOM
2.3	26787	12173	257	8	403	1487
2.4	27599	13690	281	8	415	1747
2.4.1	27602	13690	281	8	415	1747
2.4.2	27629	13731	282	9	416	1751
2.5	27629	13731	282	9	416	1751
2.5.1	27629	13731	282	9	416	1751
2.6	27629	13731	282	9	416	1751
3.0	47622	16076	251	10	483	3178
3.0.1	47877	16053	251	10	485	3201
3.0.2	47883	16053	251	10	485	3205

3.1	49775	16025	257	11	499	3313
3.2	60719	21164	328	12	504	3402
3.2.1	60770	21172	328	12	504	3402
3.2.2	61048	20838	330	12	579	4006
4.0	69374	27370	367	13	673	4435
4.0.2	69409	27379	367	13	673	4435
4.0.3	69412	27383	367	13	673	4435
4.1	78400	31234	400	14	750	4912
4.2	94604	38867	453	15	920	5869
4.3	109403	47847	531	21	1174	7302
4.3.1	109445	47855	531	21	1174	7302
4.3.2	109515	47884	531	21	1174	7314
4.4.1	109472	47035	546	21	1224	7380
4.3.3	109547	47887	531	22	1174	7314
4.4.2	109643	47070	546	22	1226	7392
4.5.0	111035	46824	554	22	1232	7379
4.5.1	111265	46861	554	22	1234	7404

- **Complexity Metric**

  In this study the LocMetrics tool is used to calculate the McCabe Cyclomatic complexity (MVG). Table 4.4 shows the calculated value of Cyclomatic Complexity for the twenty seven versions of the jEdit.

Table 4.4 Complexity Metrics over jEdit releases

Version	MVG	Version	MVG
2.3	3202	4.0	11269
2.4	3295	4.0.2	11279
2.4.1	3295	4.0.3	11280
2.4.2	3298	4.1	12524
2.5	3298	4.2	14449
2.5.1	3298	4.3	16253

2.6	3298	4.3.1	16263
3.0	7700	4.3.2	16268
3.0.1	7730	4.4.1	16080
3.0.2	7731	4.3.3	16275
3.1	8136	4.4.2	16105
3.2	9888	4.5.0	16310
3.2.1	9899	4.5.1	16343
3.2.2	9917		

Visualizing the data extracted and System observations

Graph-like representations turned out to be adequate for visualizing this kind of data. The various extracted models of source code and release history data can be directly mapped to graphs. The metrics value is mapped to show the growth patterns of the jEdit which helps in analyzing the Software Evolution. Various Figures in this section shows the historical development of the size, complexity, and modularity attributes of the examined system using the version numbers in the x-axis. The system has a high growing rate: it consisted of 26787 SLOC initially and of 111265 at the end. This means that the total size of the jEdit increased by over 315 percent which is a high growth rate. After the release of jEdit 2.3 the first six releases are minor releases with very short time span of 8 months, with the two releases 2.4 and 2.4.1 on same day of development also. Then there is a major release jEdit 3.0 and again some minor stable releases. There are some major releases between 4.0.3 and 4.3 and after that the growth of the system is stable. The time span between major releases is more than the minor releases. The first seven releases are minor releases after the time span of seven releases there is a major release, again there are four minor releases and then a major release and same pattern is followed. At the end now after release of 4.3 versions in 2009 onwards the development gets stable. All the releases between these 4 years are now minor means the system gets stable now.

- **Growth of SLOC in twenty seven official versions of jEdit**

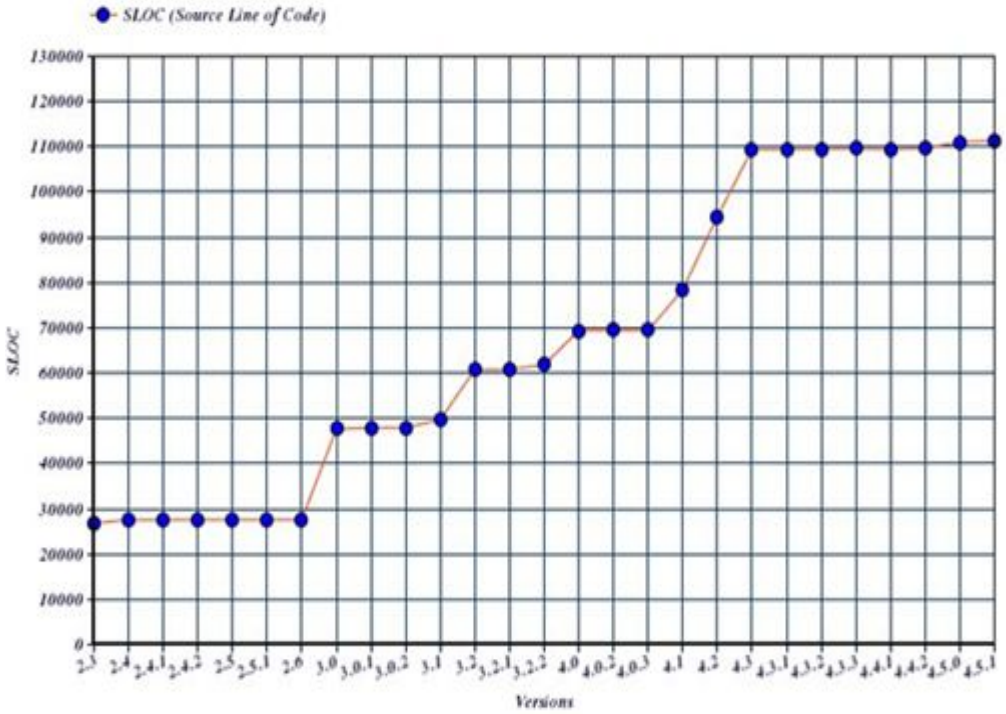


Figure 4.6 Number of Source Lines of Code Growth

- **Growth of CLOC in twenty seven official versions of jEdit**

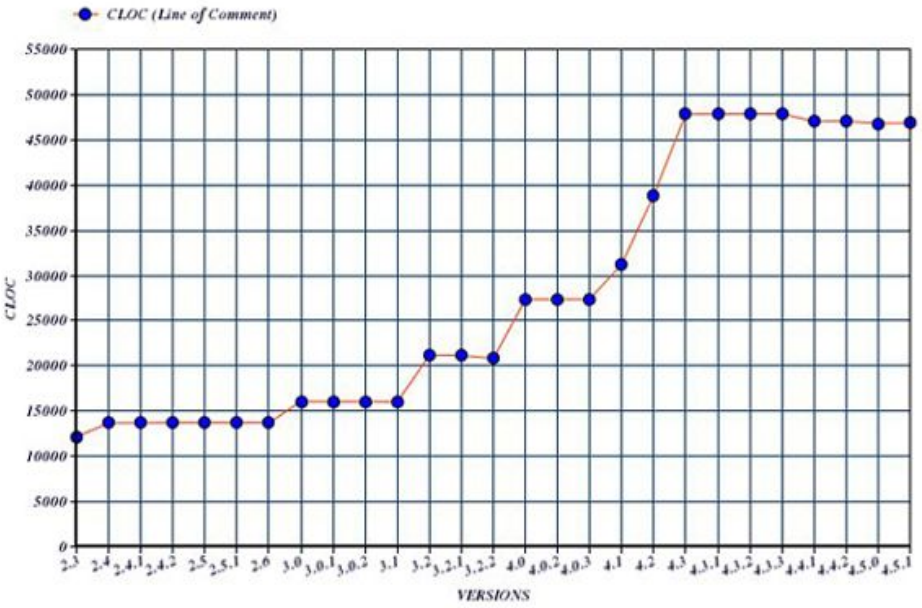


Figure 4.7 Number of Comment Lines Growth

- **Growth of MVG in twenty seven official versions of jEdit**

Figure 4.8 McCabe Cyclomatic Complexity Growth

- **Growth of NOC in twenty seven official versions of jEdit**

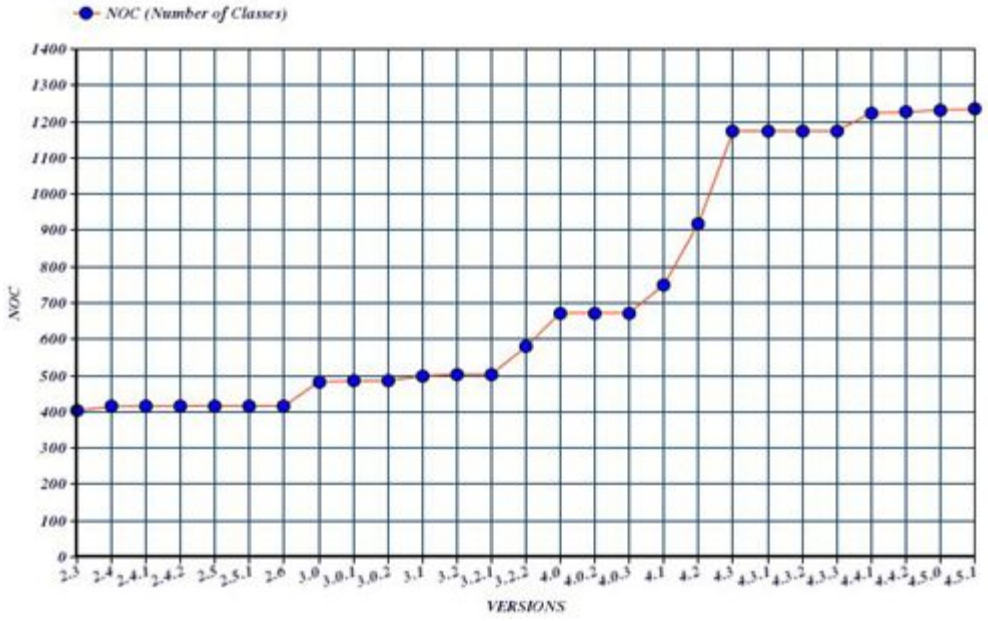


Figure 4.9 Number of Classes Growth

- **Growth of NOP in twenty seven official versions of jEdit**

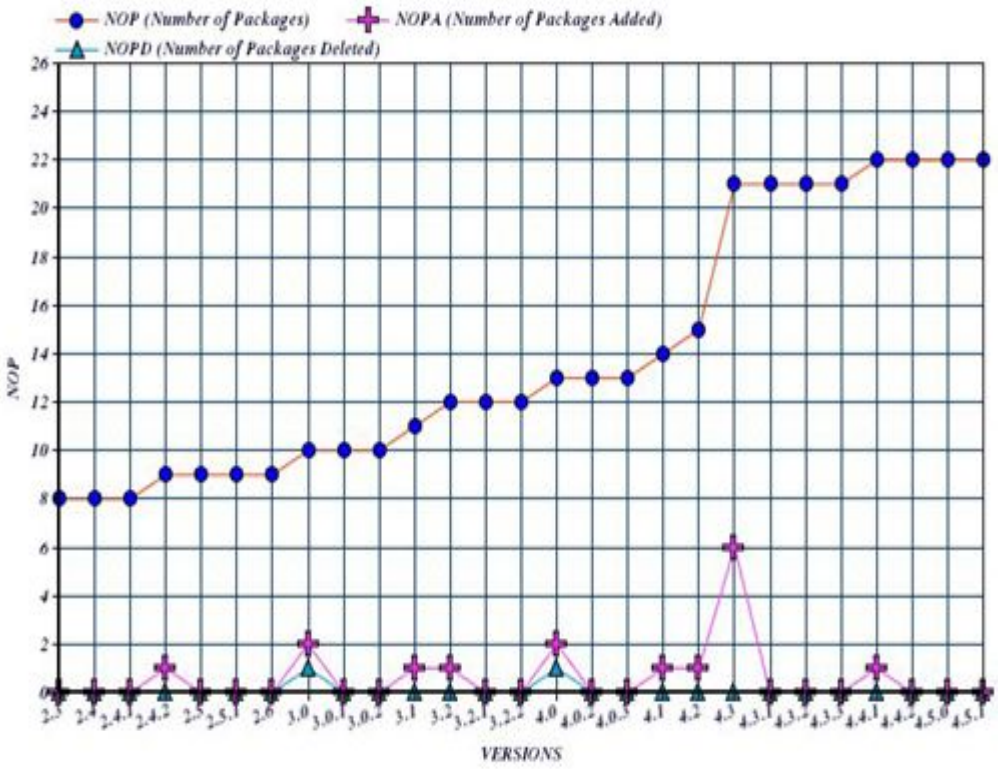


Figure 4.10 Number of Package Growth

- **Growth of NOF in twenty seven official versions of jEdit**

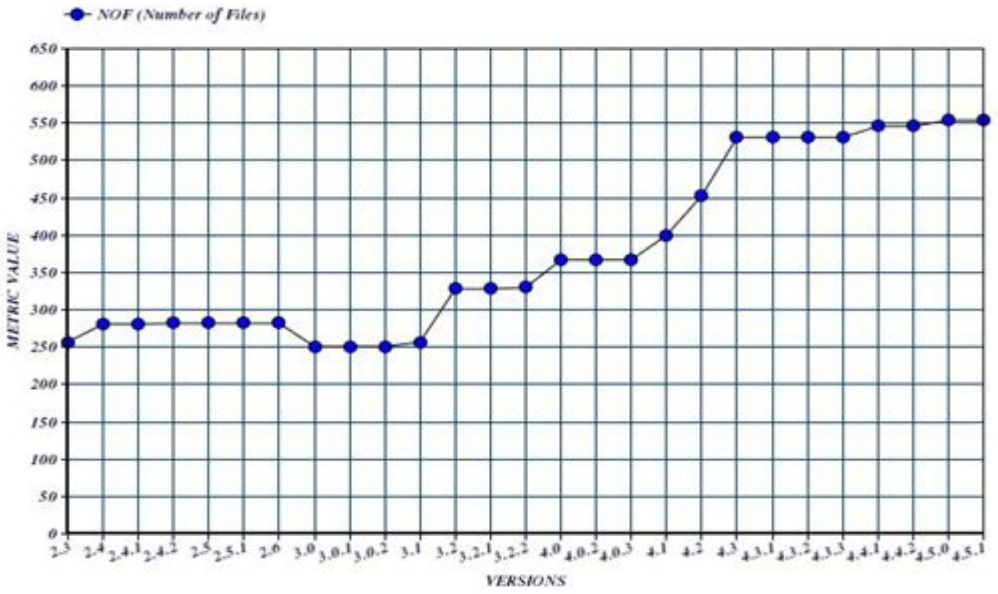


Figure 4.11 Number of Source Files Growth

- **Growth of NOM in twenty seven official versions of jEdit**

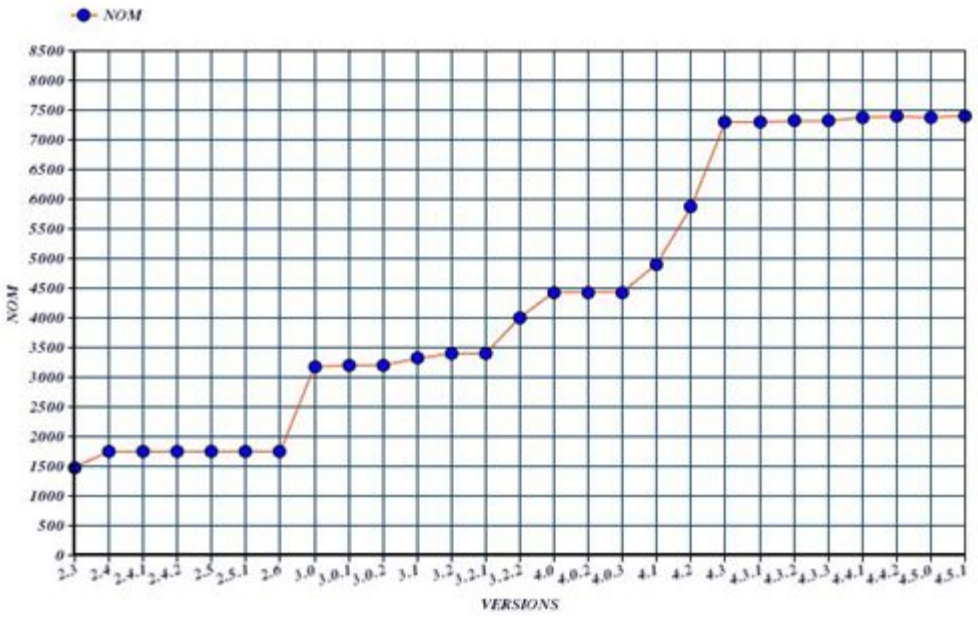


Figure 4.12 Number of Methods Growth

4.2.4 Observing relationship between various product attributes

This section presents an analysis which help in validating the software product metrics using the correlation concept of statistics. Various areas of software development can be studied and their relationship can be concluded using the concept of correlation. Here in this thesis, the development effort and maintenance effort relationship in open source software development has to be studied in terms of metrics validation approach. So the correlation analysis has to be performed using the product moment correlation with the help of a free tool called OpenStat which is providing help to various researchers. For predicting relationship among the development effort and maintenance effort the study use SLOC and MVG the metrics under analysis test. SLOC is a good measure of the development as it indicates that higher its values mean higher the development effort. MVG is another metric which is a good indicator of maintenance as higher the complexity higher will be the effort needed for maintenance and testing of the software. The following table can be given as a result of the correlation analysis over these two metrics.

Table 4.5 Correlation Matrix between SLOC and MVG

Correlations		
	SLOC	MVG
SLOC	1.000	0.992
MVG	0.992	1.000
Means		
Variables	SLOC	MVG
	67730.444	10173.444
Standard Deviations		
Variables	SLOC	MVG
	32858.672	5100..535
Number of Valid Cases = 27		

4.2.5 Lehman's Laws of Software Evolution

In this section we analyze the Lehman's laws of software evolution using the measures of several metrics, computed for the 27 different versions of jEdit released through evolution process. Some of the laws have a direct relevance to the computed metrics whereas some of the laws cannot find direct relevance to software metrics.

Law I: Law of Continuing Change

The first law postulates that a program must continually adapt to its environment, otherwise it becomes progressively less useful. According to this law, evolving software systems must be continually adapted to the changes else they become progressively less satisfactory or the evolving software must be adapted to the changing environment. Software may change in response to bug fixing activity or in response to change in the environment. The changes in usage environment may include some addition to classes which will increase the size of software leading to growth. In this case study

of the jEdit it can easily be recognized that the number of classes increases gradually as the new version of the program evolves and the significant class growth occurred between some set of versions like version 4.2 and version 4.3 which shows that the changes lead to increase in functionality. Figure 4.9 shows the variations in the growth of classes and visualized that the jEdit supports the first law of software evolution.

Law II: Law of Increasing Complexity

The second law postulates that as a program evolves, its complexity increases, unless proactive measures are taken to reduce or stabilize the complexity. According to this law, the complexity of software tends to increase over several releases unless some measures are taken to keep the complexity under check. In this case study of the jEdit it can easily be recognized that the complexity increases gradually as the new version of the program. Figure 4.8 shows the variations in the growth of the McCabe's Cyclomatic Complexity and visualized that the complexity increases over the subsequent version releases which states that jEdit supports the second Lehman's law of software evolution.

Law III: Law of Self Regulation

According to this law, the software evolution process is self regulating; leading to a steady trend i.e., the system will adjust its size throughout its lifetime. The observations made from the growth curve of jEdit given in Figure 4.6 shows that there is steady increase in size of the software following the Lehman's 3rd law of software evolution with some exceptions when there are major releases. There is not an evidence of very huge growth and the successive versions show a steady change in the development so the law can be followed but with some exceptions also that some major releases are also in growth process. There are no peaks in the graph which shows that the major changes in whole structure has been take place.

Law IV: Conservation of organizational stability

According to this law, the average global rate of activity on an evolving system is invariant over the product life time. Lehman et al. [11] suggest using the number of changes per release as possible work rate indicator. Measures called change and growth rates, i.e., the number of program additions and program changes. Using this metric it was found that the change and growth rates is not in the initial phase of development but after sometime it increases abruptly which suggests larger efforts as programs grow. Intuitively, these high changing and growing rates make sense since the programs are open source, and the number of developers tends to increase over a program's lifetime. But open source development involves community effort and this community generally grows for open source software. So don't have enough data to determine if this law is applicable to our examined open source programs.

Law V: Conservation of Familiarity

According to this law the changes made to evolving software during successive releases is limited. This allows conserving familiarity for developer and maintainers. A metric we used was the growth rate (percentage of new programs added to a release). The study using Figure 4.11 observed that the number of source file involved in single revision were not very large. There was only one or two instance where whole of the package structure of jEdit was changed and therefore familiarity was maintained following the law with some exceptions.

Law VI: Law of Continuing Growth

According to this law, the functionality provided by the software should continually grow so that the users get satisfied over a long period of time which is an advantage of the evolving systems. The various size and function metrics can be used to study the growth of the system. Growth can be interpreted as increase in the size of code or increase in the functionality being

provided by the software. In this case study of the jEdit Figure 4.6 shows the continuous growth of the software project. jEdit has a steady growth over maximum number of versions but it has one or two time the significant growth which shows that the software is growing continuously following the sixth laws of software evolution given by Lehman.

Law VII: Law of Declining Quality

According to this law, the quality of evolving software will decline unless some substantial efforts are made to improve it. The decrease in quality can be predicted by increase in the complexity as increase in complexity itself is an indicator of declining quality. In case of jEdit, the law is clearly supported by the Figure 4.8 where Figure itself shows the continuously increase in the complexity of the software.

Law VIII: Law of Feedback system

According to this law, the evolution process is a multi–level, multi-agent system. For open source software the law seems to be true since feature request and reporting of bug comes from user community. The existence of feedback system is true for jEdit since there is the user mailing system in the website hosting jEdit development. Total number of bugs reported till March 2012 is 3509332 which mean a proper feedback system is there to support the jEdit development. User mailing list also defines that the request of new features and changes in existing features is also present in this software. According to collected data from email lists up to March 2012; there are total 25,143 email's send by jEdit users in the 12 years of development and total of 45,153 emails send by the jEdit developers which shows that there is a strong feedback system.

Chapter 5

Interpretation of the data

5.1 Evolution Patterns

Based on the structural information shown in the previous chapter the historical development of the structure of jEdit is tracked. This concerned, for example, the evolution of the size of the system and its complexity and the number of classes which changed from one release to the next is shown. With the information gathered this way, current structure of the system can be shown which helps in finding the evolution patterns also. In this section we summarize the characteristics of the jEdit system that we have observed on the system. The results briefly summarize our findings from the above facts:

- The size of the system is increasing sub linearly. Figure 5.1 visually explains the growth of source lines of code sub linearly.

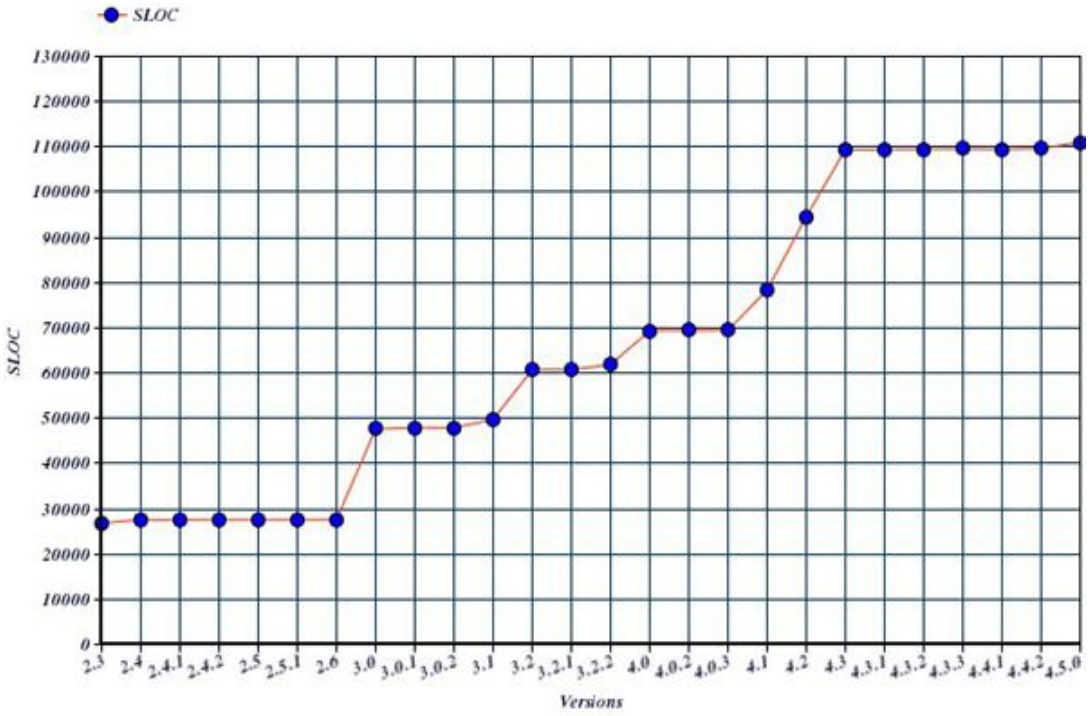


Figure 5.1 Growth of the size of jEdit

- Between releases 4.0.3 and 4.3 and in releases 3.0, 3.2, 4.0 some major activities can be observed. Figure 5.1 shows the significant changes between versions.
- The code and comments coevolves, both are changed in the same revision: most of the comment changes are done in the same revision as the associated source code change.
- In general the Growing rate of the whole system also decreases. Figure 5.2 shows that the growing rate of the system in the final versions goes on decreasing.
- In general the changing rate is always more than the growing rate with few exceptions when the changing environment demands the growth of the system.
- In the last examined releases only a few new files has been added and deleted which is shown by the growing rate. Growing rate here is defined as the number of files added plus number of files deleted.
- The structure of the whole system has become stable in the final versions after releases of version 4.3 in 2009, the system gets stable.

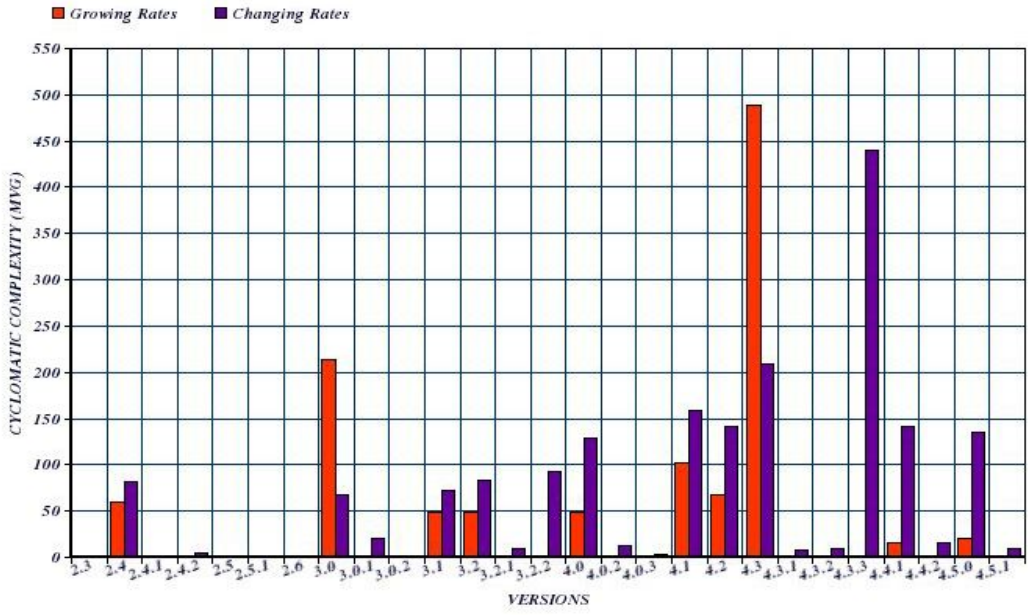


Figure 5.2 Growing and changing rate of jEdit

The above given data interpret that, observations on the system level of the jEdit system indicate that the system supports evolution in a satisfactory way, because the growing rates decrease over time and also the releases become stable over the time. This means that the number of changes in the structure of the programs significantly decreased over the last observed releases making the system stable. So open source software evolution studied depicts that in the starting years of development the growth and changing rate is more than the last years of the development making the system stable.

5.2 Relationship between the size and complexity metrics in open source

The various size and maintenance metrics have been studied and the relationship between them is analyzed in case of open source software. Correlation method has been used and the data given in the previous chapter is analyzed. The product moment correlation coefficient can be used to tell us how strong the correlation between two variables is. Table 5.1 shows the results of correlation test of the size and complexity metrics showing the value of correlation among the five major attributes of the system.

A positive value indicates a positive correlation and the higher the value, the stronger the correlation means that the variable are strongly in relationship in a positive side. With the increase in one variable the other variable is also increased and with the decrease in one variable the other variable is also decreased. Generally, correlations above 0.80 are considered pretty high which means that the size and maintenance are highly correlated in open source software. The complexity metrics and the SLOC metric are positively correlated with a value of 0.992 which is a high value means in the open source software the evolution trends for the SLOC is same as the complexity. Similarly the complexity metrics is positively correlated with other size metrics also such as the complexity is correlated with the Number of methods with a value of coefficient equal to 0.986 which also shows that the evolution trend of one is same as the trend of other. Number of package has maximum correlation with number of packages. As the number of packages increase the number of classes increases in

the same fashion. So number of packages can be validated using the method of correlation. Analyzing the table of correlation the complexity has the highest relation with Source line of code.

Table 5.1 Correlation Matrix

Product Moment Correlations Coefficient					
	MVG	NOP	SLOC	NOM	NOC
MVG	1.000	0.943	0.992	0.986	0.933
NOP	0.943	1.000	0.974	0.980	0.991
SLOC	0.992	0.974	1.000	0.997	0.970
NOM	0.986	0.980	0.997	1.000	0.976
NOC	0.933	0.991	0.970	0.976	1.000
Number of valid cases		27			

The observations help in concluding that the growth pattern of the complexity is similar to the growth pattern of the source line of code. Thus the increase in development effort increases the maintenance effort also. Summarizing the observations, the answer to the problem statement dictates that the maintenance is directly related to the development effort. As the developers effort increases the functionality, the complexity of the system increase in turn increasing the maintenance. The results can be shown using the graphically method also. Figure 5.3 shows that the growth trends of the source line of code are similar to that of the complexity metrics.

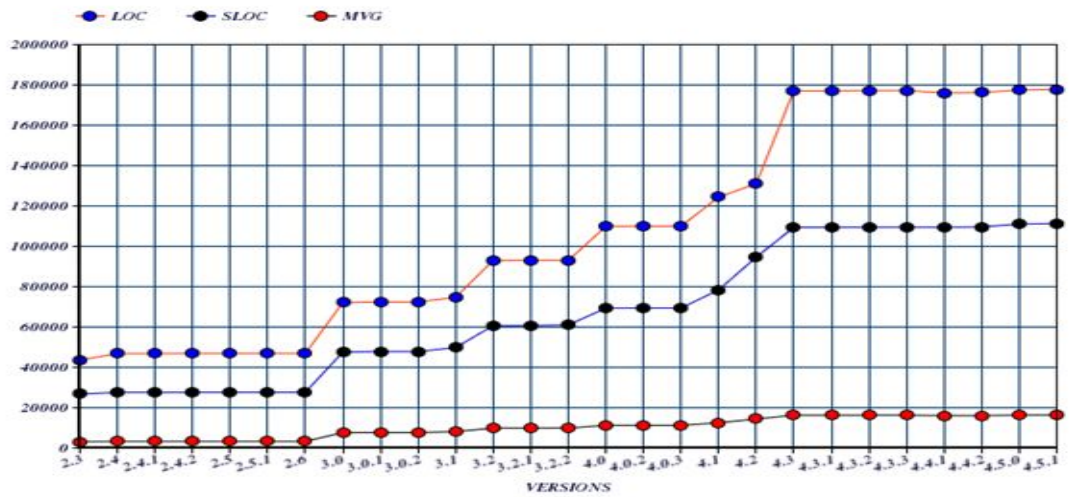


Figure 5.3: Relationship among SLOC, LOC and Complexity of jEdit

The line with red circle shows the growing behavior of the system and line with black circle shows the variation in source lines of code. LOC here depicts the actual lines of code including the blank and comment lines whereas the SLOC is the executable source lines of code which is equal to the code lines without blank and comment lines. The graph shows that when there is significant growth in the lines of code the complexity also shows the significant growth like during the release of version 3.0 the increase in SLOC increases the Complexity.

5.3 Support to Lehman's Law of Evolution in Open Source Software

Following table shows the laws and the evidence to support the law.

Table 5.2: Support to Lehman's Laws of Software Evolution

	Brief Name	**Support**	**Indicator**
1	Continuing Change	Yes	Figure 4.9 clearly indicates continuing growth.
2	Increasing complexity	Yes	Figure 4.8 clearly indicates continuing growth.
3	Self Regulation	Yes	Figure 4.6 proof this law (with exception).
4	Conservation of Organizational Stability	No	No data that provides evidence for or against the law is applicable.
5	Conservation of Familiarity	Yes	Figure 4.11 proof this law (with exception).
6	Continuing Growth	Yes	Figure 4.6 clearly proof this law.
7	Declining Quality	Yes	Figure 4.8 clearly proof this law
8	Feedback System	Yes	Email system is used as feedback system and total of 70296 emails has been send by both the group of jEdit users and jEdit developers till March 2012 which supports the law.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

This thesis presents the study based on open source software called jEdit. The evolution patterns of the open source software are studied using the version control data by mining the software repositories. The repositories are mined using the version control system tools. The relationship between the development effort and maintenance effort is studied. The applicability of Lehman's laws of software evolution to open source software was studied in the light of number of metrics. The validity of the study can be ascertained as the source code of the system under study is available on internet and can be used for any research work. This thesis demonstrated a methodology to recover the evolution of a software project using its version control data. Version control data, such as version releases, source code, version control logs and mailing lists were used to recover the evolution of jEdit, a free programmer's editor. The analysis of these versions allowed the discovery of interesting facts about the history of the project: its growth in size, its growth in complexity, the size of the contributions, and important statistics in its development. The analysis of the system results to make some possible observations about the system under study with respect to the area of software evolution.

- Although the number of people who have contributed to the project is equal to 126, most of the work is done by a small number of contributors. Among the maximum number of commits per year the 64% of the commits is done only by the most active author called Ezust.
- At the system level the growth of system is increasingly sub linear.
- The code and comment grows in similar manner means understandability is also enhanced with the increase of functionality.
- There is a strong positive relationship among the development and the maintenance efforts that is with the increase in development effort of system the maintenance effort also increases.

- The reflection of Lehman's law 1, law 2, and law 6 was easily determined using the metrics approach and the law 3, law 4, law 5 was hard to determine for open source software because of its unlimited resources and open environment. Most of Lehman's law of software evolution supports open source software under study but law 4, law 5 needs more empirical study to relate with open source software.

Thus the thesis presents the evolution behavior of jEdit and proposed that the size of the system grows gradually over most of the releases and the size metrics have positive relation with the complexity metrics. The study confirmed that the evolution of system under study is consistent to Lehman's 1st, 2nd, 6th, 7th, 8th laws of software evolution. While some laws need more research to be observe their behavior in open source software development process. Thus applying the metrics over the open source software can provide a method to software engineer to understand how the system evolved over the time.

6.2 Future Scope

This study here involves the metrics approach to quantify the evolution of open source software and involves development and maintenance attributes to study its evolution behavior. This study has opened up more opportunities for research in the field of software evolution using the metrics evolution. This study provides the method to the researchers about measuring the open source software. In future the researchers can add more set of metrics that are applicable in measuring the quality and reusability of the open source software based on software evolution. Regarding the category of evolvability characteristics, changeability, development, modularity and complexity are addressed in this case study. However, there are some more evolvability characteristics that are not covered such as reusability, extensibility, testability. Thus further studies can use these characteristics to show the evolution patterns with respect to all the evolvability characteristics. The evolutions patterns of subsystems can also be studied to find that subsystem behave in a similar way or not as system behave. Thus, software evolution can be studied at subsystem level also.

List of Publications

Nindya Kotwal & Vineeta Bassi, "*A comprehensive study of Version Control System in Open Source Software*", International Journal of Scientific & Engineering Research, Volume 3, Issue 4, April-2012, ISSN 2229-5518.

References

[1] A. Brown and G. Booch, "*Reusing Open Source Software and Practices: The impact of open source on commercial vendors*", Software Reuse: Methods, Techniques, and Tools, Proceedings, vol. 2319, pp. 123-136, 2002.

[2] G. Polancic, M. Hericko, R.V. Horvat , "*Open Source Software Usage Implications in the Context of Software Development*", Informatica ,vol. 29, pp. 483–490, 2005.

[3] M. Elliott, M. S. Ackerman, W. Scacchi, "*Knowledge Work Artifacts: kernel cousins for free/open source software development*," Proceedings of the 2007 international ACM conference on Supporting group work, November, 2007.

[4] S. Jeschke, C. Muller, S. Grottke, "*Lecture on Free / Libre and Open Source Software (FLOSS)*", Institute of Information Technology Services, University of Stuttgart, 23 April, 2009.

[5] R. Roets, M. Minnaar, K. Wright, "*Towards Successful Systems Development Projects in Developing Countries*", Proceedings of the 9th International Conference on Social Implications of Computers in Developing Countries, Sao Paulo, Brazil, May 2007.

[6] V. Tiwari, "*Software Engineering Issues in Development Models of Open Source Software*", IJCST Vol. 2, Issue 2, June 2011.

[7] A. Beard, H. Kim, "*A survey on open source software licenses*", Journal of Computing Sciences in Colleges, vol. 22 no. 4, April 2007.

[8] E. Sink, *Version Control by Example*, Pyrenean Gold Press, July 2011.

[9] T. Ball, K. Adam, H.P.Siy, "*If Your Version Control System Could Talk* ", 2011.

[10] Y. Ren, T. Xing, Q. Quan, Y. Zhao, "*Software Configuration Management of Version Control Study Based on Baseline*", 3rd International Conference on Information Management, Innovation Management and Industrial Engineering, 2010.

[11] M. Lehman, J. Ramil, "*Towards a theory of software evolution - and its practical impact"*. Invited Lecture, Proceeding of International Symposium on principles of Software Evolution , pp. 2–11,2000.

[12] K. H. Bennett, V.T. Rajlich," *Software Maintenance and Evolution: a Roadmap*", Anthony Finkelstein (Ed.), ACM Press, 2000.

[13] Y. Lee, J. Yang, K. H. Chang, "*Metrics and Evolution in Open Source Software*", Seventh International Conference on Quality Software, pp. 191 – 197, 2007.

[14] Mattewkennedy,"Evaluating open source software ", Defence AT&L,2010.

[15] K. Crowston, Kangningwei, J. Howison, A. Wigggins, "*Free/Libre Open Source Software Development What We Know and What We Do Not Know*",Syracuse University School of Information Studies, 2009.

[16] OSTG Open Source Technology Group, Inc. 2007, Source Forge [online]: *http://sourceforge.net/*

[17] J. Feller, B. Fitzgerald, *"A Framework Analysis of the Open Source Software Development Paradigm"*, University College Cork, Ireland, ACM, 2000.

[18] G. Schryen, R. Kadura, "*Open Source vs. closed source software: towards measuring security*", Proceeding SAC '09 Proceedings of the 2009 ACM symposium on Applied Computing, pp. 2016-2023, ISBN: 978-1-60558-166-8.

[19] Sourceforge [Online]: *http://www.sourceforge.net/*

[20] Freecode [Online]: *http://www.freecode.com/*

[21] E. E. Kim ,"*An Introduction to Open Source Communities*", Blue Oxen Associates Technical report, BOA-00007, 2003.

[22] E.S. Raymond, *The Cathedral and the Bazaar Musings on Linux and Open Source by an Accidental Revolutionary*, 1st ed. Cambridge, MA: O'Reilly, 1999.

[23] A. Deshpande and D. Riehle, "*The Total Growth of Open Source*", Open Source Development, Communities and Quality, pp. 197-209, 2008.

[24] K. Crowston and J. Howison, "*The social structure of free and open source software development*", First Monday, 10(2), 2005.

[25] A.H. Maslow, "*Motivation and Personality*", 2nd edition, Harper and Row, New York, 1970.

[26] F. Kon, P. Meirelles, N. Lago, A. Terceiro, C. Chavez, M. Mendonca, "*Free and Open Source Software Development and Research: Opportunities for Software Engineering*", 25th Brazilian Symposium on Software Engineering, 2011.

[27] A. Khanjani, R. Sulaiman,"*The Process of Quality Assurance under Open Source Software Development*",IEEE Symposium on Computers & Informatics, pp. 548 – 552, 2011.

[28] Q. Zhao, H. Wang, G. Feng, "*Software Evolution Method Considering Software Historical Behavior*", Fourth International Conference on, pp. 36 – 41, 2009.

[29] C.F. Kemerer, S. Slaughter, "*An empirical approach to studying software evolution*", IEEE Transactions on Software Engineering, vol.25, no. 4, pp. 493 - 509, 1999.

[30] L. A. Belady, M.M. Lehman, "*A Model of Large Program Development*", IBM Systems Journal, vol. 15, no. 1, pp. 225±252, 1976.

[31] M. M. Lehman, D. Perry, J. E. Ramil, "*Implications of evolution metrics on software maintenance.*" Proceeding of the International Conference on Software Maintenance (ICSM'98), Bethesda, Maryland, pp. 208-217, 1998.

[32] M. M. Lehman, L. A. Belady, *Program evolution. Processes of software change*, Academic Press Professional, Inc., San Diego, CA, USA, ISBN: 0-12-442440-6 ,1985.

[33] N. H. Madhavji, "*Introduction to the panel session Lehman's laws of software evolution*", Proceedings of International Conference on Software Maintenance, pp. 66-67, 2002.

[34] W. G. Michael, Q. Tuang, "*Evolution in Open Source Software: A Case Study*", Proceedings of International Conference on Software Maintenance, pp.131-142,2000.

[35] K. Nakakojil, Y. Yarnamoto, Y. Nishinaka , K. Kishida, "Evolution Patterns of Open-Source Software Societies and communities", Proceedings of the International Workshop on Principles of Software Evolution, pp. 76 – 85,2002.

[36] H. Gall, M. Jazayeri, R. Kloesch and G. Trausmuth, "*Software evolution observations based on product release history*", Proceeding of International Conference on Software Maintenance (ICSM '97), 1997.

[37] A. P. Nikora, "*Understanding the nature of software evolution*", Proceedings of International Conference on Software Maintenance, pp. 83-93, 2003.

[38] I. Herraiz, "*A statistical examination of the evolution and properties of libre software*":, IEEE International Conference on Software Maintenance, pp. 439 - 442, 2009.

[39] H. Pei,"*A Systematic Review of Studies of Open Source Software Evolution*" , Breivold Industrial Software Systems ABB Corporate Research 721 23.

[40] L. Ping, "*A Quantitative Approach to Software Maintainability Prediction*", Conference publication on Information Technology and Applications (IFITA), Vol. 1,pp.105-108, 2010.

[41] G. Ramesh, *Managing Global Software Projects*, Tata McGraw Hill.

[42] S. Ali, A. Ghafoor, R. A. Paul, *"Metrics-guided quality management for component-based software systems*",25th Annual International Conference on Computer Software and Applications Conference, pp.303-308,2001.

[43] R. Gunnalan, M. Shereshevsky, H. H. Ammar, "*Pseudo dynamic metrics [software metrics]"*, the 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.

[44] T. McCabe, "*A software Complexity Measure*", IEEE Transactions on Software Engineering, Vol. 2, Issue 4, pp:308-320,1986.

[45] A. D. Carleton, E. Robert and B. W. Goethert, "*The SEI Core measures*", Journal of the Quality Assurance institue,2004.

[46] D. German, "*Using software trails to rebuild the evolution of software*.",Journal of Software Maintenance and Evolution: Research and Practice, November 2004.

[47] A. Bieniusa, P. Thiemann, S. Wehr, "*The Relation of Version Control to Concurrent Programming*", International Conference on Computer Science and Software Engineering, vol. 3,pp. 461-464, 2008.

[48] B. C. Sussman, B. W. Fitzpatrick, C. M. Pilato, *Version Control with Subversion For Subversion 1.6*, 2010, ISBN 9781440495878.

[49] Daniel M. German, "*An empirical study of fine-grained software modifications*.", In Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004.

[50] H. Gall, M. Jazayeri, and J. Krajewski, "*CVS release history data for detecting logical couplings.*", In Sixth International Workshop on Principles of Software Evolution, 2003.

[51] J. M. Gonzalez-Barahona, L. Lopez, and G. Robles, "*Community structure of modules in the Apache project.*", In Proceedings of the 4th International Workshop on Open Source Software Engineering, 2004.

[52] R. Robbes, M. Lanza, ”*Versioning Systems for Evolution Research*”, Eighth International Workshop on Principles of Software Evolution,pp. 155 – 164,2005.

[53] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, *Version Control with Subversion For Subversion 1.5,*2008, ISBN 9780596510336.

[54] J. Santore, T. Lorenzen, R. Creed, D. Murphy, and R. Orcutt, “*The Software Engineering Class Builds a GUI for Subversion*,” Department of Mathematics and Computer Science Bridgewater State College Bridgewater, Massachusetts 02325 USA, vol. 41, December 2009.

[55] C Pilato, Ben Collins-Sussman and Brian Fitzpatrick, *Version Control with Subversion 2*, O'Reilly Media, Inc. ISBN: 0596510330 9780596510336.

[56] Tigris.org, Open Source Software Engineering Tools [Online]: http://www.tortoisesvn.tigris.org/

[57] LocMetrics [Online]: *http://www.locmetrics.com/*

[58] SourceMonitor [Online] : *http://www.campwoodsw.com/sourcemonitor.html*

[59] OpenStat [Online] : *http://www.statprograms4u.com/OpenStatMain.html*

[60] G. R. Martinez et. al.,”*Studying the Evolution of Libre Software Projects using publicly available data*", Proceedings of the 3rd Workshop on Open Source Software Engineering, pp. 111-115, 2003.

[61] jEdit Programmer’s Text Editor [Online] : *http://www.jedit.org*