Journal of Medical Education, 2, 53-55. doi: 10.5116/ijme.4dfb.8dfd

WILSON, M. 2005. Constructing measures: An item response modeling approach. Mahway: Erlbaum.

YONG, A. G., AND PEARCE, S. 2013. A beginner's guide to factor analysis: Focusing on exploratory factor analysis. Tutorials in Quantitative Methods for Psychology, 9(2), 79-94.

Empirical Study of Continuous Change of Open Source System

Michael Abayomi Olatunji3, Rufus Olalere Oladele and Amos Orenyi Bajeh
Department of Computer Science
University of Ilorin, Ilorin, Nigeria
P. M. B. 1515, Ilorin, Nigeria.
mikeolaus@gmail.com, roladele@yahoo.com, bajeh_amos@yahoo.com

ABSTRACT

This paper describes empirical investigation of the first law of software evolution: law of continuing change, using four open source systems in evolution as case studies. The four systems used as case studies have 354 versions altogether with a combined 55 years of evolution and running existence. Three code analysing tools were used to parse source codes and generated

3 Author's Address: Michael Abayomi Olatunji3, Rufus Olalere Oladele and Amos Orenyi Bajeh
Department of Computer Science, University of Ilorin, Ilorin, Nigeria, P. M. B. 1515, Ilorin, Nigeria.
mikeolaus@gmail.com, roladele@yahoo.com, bajeh_amos@yahoo.com International Journal of Computing and ICT Research, ISSN 1818-1139 (Print), ISSN 1996-1065 (Online), Vol.11, Issue 2, pp. 31 - 52, June 2017.

metrics needed for quantification of continuing change property. Statistical functions and descriptive statistics were used for metrics processing and analysis. Results of the study confirm the law of continuing change. In particular, results show that: (i) in the pre versions, lot of changes do occur but when it is getting close to first release candidate of version 1, the changes reduce but in stable branch after the first official release more changes do occur close to the release of another stable or development major release. (ii) More changes occur in the function body than in the function interfaces. (iii) Changes in the stable branches are fewer than those in the development branches. (iv) Function interface changes are correlated to the function statement changes. Within the domain of the case studies investigated, there is no system that does not have shrinkages in number of functions and Lines of code in the course of evolution.

Keywords: Lehman Law, Empirical investigation, Open Source Systems, Software Metrics, Software Evolution, Quantification.

IJCIR Reference Format:

Michael Abayomi Olatunji4, Rufus Olalere Oladele and Amos Orenyi Bajeh. Empirical Study of Continuous Change of Open Source System, Vol. 11, Issue.1 pp 31 - 52.
http://ijcir.mak.ac.ug/volume11-issue1/article3.pdf

INTRODUCTION

4 Author's Address: Michael Abayomi Olatunji4, Rufus Olalere Oladele and Amos Orenyi Bajeh
Department of Computer Science, University of Ilorin, Ilorin, Nigeria, P. M. B. 1515, Ilorin, Nigeria.
mikeolaus@gmail.com, roladele@yahoo.com, bajeh_amos@yahoo.com International Journal of Computing and ICT Research, ISSN 1818-1139 (Print), ISSN 1996-1065 (Online), Vol.11, Issue 2, pp. 31 - 52, June 2017.

Software evolution is the process by which softwares are modified and adapted to their changing environment to maintain relevance. Software evolution became a field of research owing to the pioneering empirical studies done by Meir Lehman, which he started by using IBM systems as case studies about 47 years ago. Lehman aimed at formulating theory for software evolution from empirical and statistical perspective. He intended to bring out commonalities or invariants within software systems. In the course of pursuing the above aim, he came by three debatable invariants which were postulated as the laws of software evolution in 1974. After years of continual research in software evolution, more invariants were observed and postulated as laws; by 1996, the laws of software evolution became eight in number, amongst which is the law of Continuing Change which is further investigated in this study. The laws of software evolution are said to be referring only to software systems classified as Evolutionary type systems (E-type systems), these are systems that solve problems involving people or real world. They are reflections of human processes. E-type systems can be Open Source Systems (OSS) or commercial/proprietary systems which are close source systems developed for a particular organisation.

The law of continuing change was the first of the three laws of software evolution postulated in 1974 and it states that E-type systems must be continually adapted else they become progressively less satisfactory [Lehman 96b]. Since user requirements changes with time, software systems are inevitably subject to change and thus, must evolve to continually meet the needs of the users. Otherwise, with time the software will not be meeting users' needs and satisfaction, thereby losing relevance. The other laws of software evolution which are not the focus of this paper are stated as follows:

Increasing complexity (1974): as an E-type system evolves its complexity increases unless work is done to maintain or reduce it.

Self-regulation (1974): E-type system evolution process is self-regulating with distribution of product and process measures close to normal.

Conservation of organisational stability (1980): the average effective global activity rate in an evolving E-type system is invariant over product lifetime (Invariant work rate).

Conservation of familiarity (1980): as an E-type system evolves all associated with it, developers, sales personnel, users, for example must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariants as the system evolves.

Continuing growth (1980): the functional content of E-type system must be continually increased to maintain satisfaction over its lifetime.

Declining Quality (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to the operational environment changes.

Feedback system (first stated in 1974, formalised in 1996): E-type system evolution process constitute multi-level, multi-loop, multi agent feedback system and must be treated as such to achieve significant improvement over reasonable base.

Lehman studied seven commercial large software systems and came out with the laws of software evolution. There are lot of differences between commercial systems and OSS. For instance, the development of OSS is done by programmers scattered over the world. Also, the development of OSS and its evolution/maintenance are been done at same time which happen at different times in commercial systems. Some studies [Godfrey and Tu, 2000; Herraiz 2008; Fernandez-Ramil et al. 2008; Israeli and Feitelson 2010; Neamtiu et al. 2013; Pirzada 1988; Alenezi and Almustafa 2015; Eick et al. 2001; Vasa, 2010] show that some of the laws are not holding or partially hold in OSS. The laws have to be investigated with OSS in order to check the applicability of the laws with OSS and to arrive at theory for software evolution, most especially with the availability of lot of open source codes in diverse repositories on internet. This paper is an attempt towards validating the first law, law of continuing change, using a new set of metrics.

RELATED WORKS

Since the promulgation of the Lehman laws [Lehman 74; Lehman 78; Lehman 79; Lehman 80; Lehman 85a; Lehman 85b; Lehman 85c; Lehman 89; Lehman 90; Lehman 96], several studies have been carried out that attempts to validate and also refine the laws. The last expanded version

of the laws published in 1996 [Lehman 96b] remains unmodified since then. However, it was since 1996 that most studies questioning or confirming their validity have been published.

Lawrence [1982] in his study on software evolution dynamics, validated the law of continuing change using number of modules handed in each release as metric.

Pirzada [1988] addressed the validity issue of the Lehman's laws of software evolution using statistical approach in his PhD thesis. In the study, the evolution of several flavours of UNIX was investigated. One of the results of Pirzada's study showed that all the UNIX flavours verified the first law of continuing change. He considered number of modules in release as metric.

Godfrey and Tu [2000] studied the case of the Linux kernel in an attempt to verify the laws of software evolution in the context of Open Source Software (OSS) and found out that it was evolving at an accelerating pace and obeys the law of continuing change as well. Godfrey and Tu used system and module size as metrics for quantifying the Law of Continuing Change.

Koch [2007] addressed the validity question with a large sample of software projects, precisely 8,621 projects from SourceForge.net repository. Koch used SLOC as metric and graphical statistical evaluation. He observed that most of the projects grow either linearly or with a decreasing rate, according to the laws. But, about 40% of the projects showed a super linear pattern which disincline with the laws. Koch speculated that the cause of the super linear growth might be a certain organizational model, that he called the chief programmer team, a term originated in IBM in the seventies.

Vasa [2010] studied 40 large open source systems using size and complexity metrics at the class level in his study. Inclinations were found for the law of continuing change, law of self-regulation, law of conservation of familiarity and law of continuing Growth.

More so, Israeli and Feitelson [2010] also studied the Linux kernel, they investigated if they fulfil laws or not. They found that linear kernel growth in evolution is initially super linear and later became linear from 2.5 release. Therefore, the law of continuing change is verified but at a different change rate. Israeli and Feitelson concluded that Linux confirmed most of the Lehman's

laws; especially those of growth, continuing change and stability. They used number of source files counted in arch and drivers directories of Linux kernel as metric to quantify continuing change law.

In a study of 705 releases of 9 open source software projects, Neamtiu et al. [2013] reported that only the laws of continuing change and continuing growth are confirmed for all programs. Neamtiu research group are the collectors of the datasets used in this study. In their study, they used the cumulative number of changes to program elements (i.e., functions, types, and global variables) as metrics for this law.

Alenezi and Almustafa [2015] conducted empirical investigation on OOS evolution by considering five different sized and different domain OSS using Cyclomatic Complexity and SLOC as metrics. They found support for applicability of Law of Continuing Change, Increasing Complexity and Continuing Growth for OSS.

In this paper, aside function count metric used in previous studies, the quantification of the law of Continuing Change is also measured using the following new metrics: parameters in functions, returns in functions, statements in function body, and incremental differences in the following: function count, parameters, returns and function statements.

RESEARCH METHOD

SAMPLE OSS (CASE STUDIES)

We downloaded the datasets which are used as case studies for this paper from UCR online repository. The repository contains 9 merged source codes of evolutionary OSS written in C language. The main reason for using a dataset placed on internet repository by a group of evolution researchers and not trying to download all the publicly available official releases of the applications that were used as case studies is that quantification of law of software evolution must be based on

empirical results, verifiable and repeatable, and made on a large scale, so that conclusions with statistical significance can be achieved [Sjøberg et al.2008]. If software evolution is analysed with data that is not available to third parties, it cannot be verified, repeated and replicated. It is dangerous to build a theory on empirical studies that do not fulfil those requirements.

The datasets used as case studies in this paper are well evolved open source applications, namely: Bison systems, Samba systems, SQLite Systems and Vsftpd systems. The above dataset are selected because they have long term software evolution, sizable and actively maintained.

OVERVIEW OF THE APPLICATIONS USED AS CASE STUDY

Table 1 presents the sample OSS used for the empirical analysis.

Table 1: Sample OSS

OSS					Description
Bison: Version	v1	v1.3	v1.5	v2.4.3	Bison is a general-purpose parser generator, which can be use to develop a wide range of language parsers, from those used in simple desk calculators to complex programming languages.
No. of Functions	128	223	572	843	
No. of parameters	71	249	1003	1631	
No. of returns	223	456	1062	1481	
Statements	3646	7665	13214	19449	
Samba: Version	v1.5.14	v1.9.18	v2.2.12	v3.3.1	Samba is a tool suite that facilitates Windows-UNIX/Linux interoperability. It is based on the common client/server protocol of Server Message Block (SMB) and Common Internet File System (CIFS)
No. of Functions	120	940	4001	13157	
No. of Parameters	291	2071	10902	37292	
No. of returns	379	2850	17142	69626	
Statements	3039	25373	118597	506158	
VSFTPD: Version	v0.0.9	v1.1.0	V1.2.2	v2.1.0	Vsftpd stands for "Very Secure FTP Daemon" and is the FTP server in major Linux distributions like Ubuntu, Centos, Unix, Fedora and the likes. It is licensed under the GNU General Public License. It supports IPv6
No. of Functions	325	401	450	590	
No. of Parameters	414	641	724	939	
No. of returns	461	564	719	997	

Statements	2987	4342	5470	7098	
SQLite: Version	v1.0.0	v2.0.7	v3.1.1	v3.6.11	SQLite is an implementation of a self-contained SQL database engine. It is as a library in C programming language. It is not a client-server database management system and it comes with a "shell" that can be used for command–line interaction.
No. of Functions	167	303	391	1284	
No. of Parameters	393	677	886	3028	
No. of returns	648	1197	1479	3556	
Statements	6926	11527	13814	31126	

Table 1 gives information on some selected versions of the sample OSS. We analysed the entire versions of the sample OSS available on the UCR online repository as at the time of carrying out this study. Bison has 33 versions (v1.0 - v2.4.3) released over a period of 22 years. 89 versions of Samba released over a period of 15 years and grew from 5514 LOC to more than 1,000,000 LOC were analysed. 60 versions of VSFTPD were analysed. 172 versions of SQLite were analysed; starting with its first version v1.0 of 17723 LOC to v3.6.11 of 65108 LOC.

Measurement Tools and Metrics

Two static code analysing tools are selected namely Resource Standard Metrics (RSM) and PMCCABE. They consume C/C++ and Java source code files and generate diverse metrics. RSM tool was used to determine file function count, total function parameters, total function returns and number of statements in functions.

PMCCABE is a command line tool in Linux environment, it calculates McCabe-style Cyclomatic Complexity for C/C++ source code, statements in function, lines of code in function, blank lines and the likes. PMCCABE uses per-function metrics and was run with Unix scripts in the Debian command line. Metric values from PMCCABE tool was used for tracing individual functions and their metrics values for analysis purpose. The metrics values of the two tools are correlated and this gives confidence that the metrics values are correct; however, visual inspection was still done.

As regards Continuing Change law metrics, others including Neamtiu et al.[2013] have used metrics like number of functions in each release, system size, module size, function modification,

function complexity, changes to types, global variables and the likes. Function count, parameters in functions, returns of functions, statements in function body, incremental difference in the following: function count, parameters, returns and statements in function are used in this study. The above metrics have not been used in the previous studies except the function count metric. The metrics are defined as follows.

i. Function Count (FC): this is the number of functions in a software version

ii. Functions Parameters (FP): this is the sum of all the parameters in functions of a software version

iii. Functions Returns (FR): this is the sum of the return statements in functions of a software version.

iv. Functions Statements (FS): the sum of executable expressions that terminates by semi-colon.

v. Incremental difference in the following: function count, parameters, returns and statements in function.

vi. Function Count Incremental Difference (FC-ID): this is the difference between the sum of functions in a system version and the immediate preceding one.

vii. Functions Parameters Incremental Difference (FP-ID): this is the difference between the sum of parameters in functions of a software and its immediate preceding version.

viii. Functions Returns incremental difference (FR-ID): this is the difference between the sum of returns in functions of a software and its immediate preceding version.

ix. Functions Statements Incremental Difference (FS-ID): this is the difference between the sum of statements in functions of a software and its immediate preceding version.

Continuing change: E-type software systems must be continually adapted else they become progressively less satisfactory. E-type software systems adapt to their environment and to changing/increasing requirement from the users. The software systems considered as case studies are widely used and well maintained, so they are good candidate softwares to investigate the law

of continuing change. Also, changes in kind of incremental difference in functions interfaces (parameters and returns) and in the functions body like functions statements, depict continuing changes occurring to the software systems in evolution. As the softwares are being continually adapted to their environment, modifications to the function interfaces and body are inevitable.

Results and Discussion

Results

Having measured the sample OSS using the measurement tools, the following graphs depict the relationship between the selected metrics under consideration. Figure 1 to Figure 9 depicts the change behaviour of the four OSS. They show how incremental difference in Functions Count , Parameters, Returns and Statements changes in the four sample OSS.

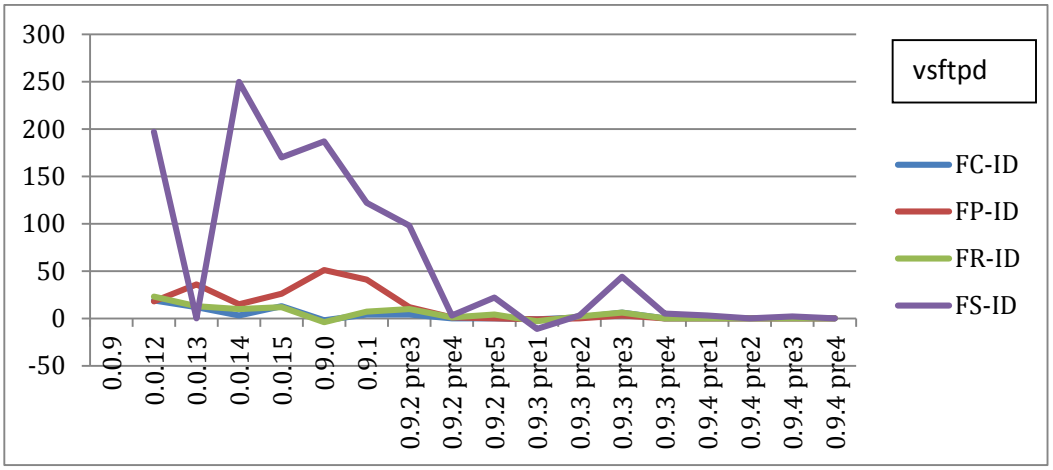

Figure 1. Graph of Function Count Incremental Difference(FC-ID), Function Parameters Incremental Difference(FP-ID), Function Returns Incremental Difference(FR-ID), Function Statements Incremental Difference(FS-ID) against vsftpd pre versions of v1.0 on horizontal axis.

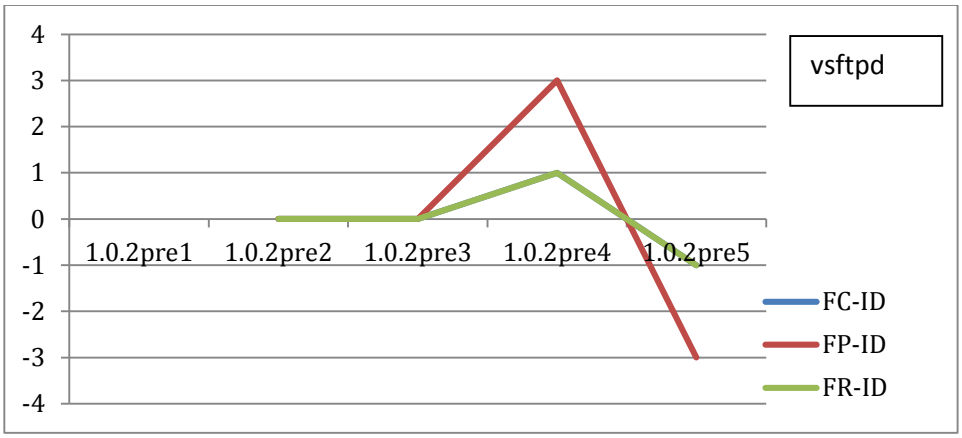

Figure 2a. Graph of Function Parameters Incremental Difference (FP-ID), Function Returns Incremental Difference(FR-ID), function counts incremental difference(FC-ID) against vsftpd version 1.0 releases on horizontal axis.

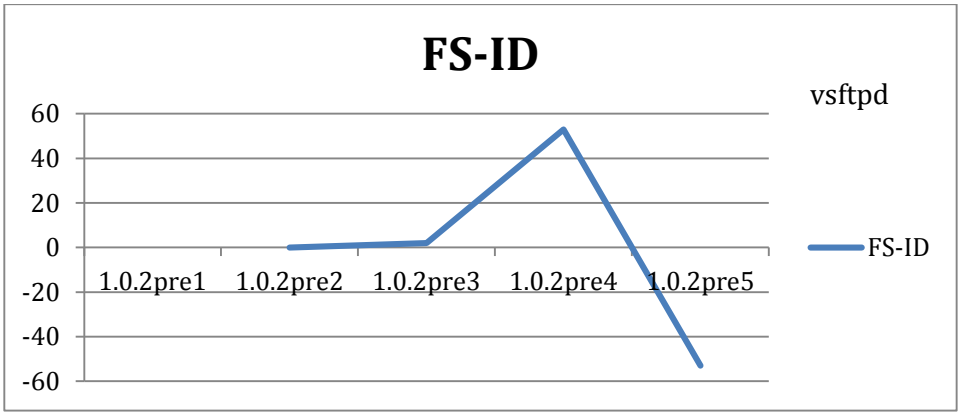

Figure 2b. Graph of Function Statements Incremental Difference (FS-ID) against vsftpd version 1.0 releases on horizontal axis.

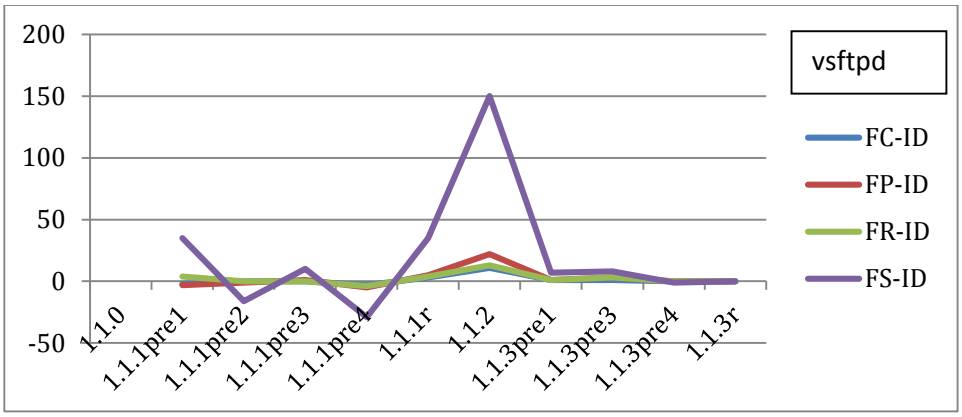

Figure 3. Graph of Function Counts Incremental Difference (FC-ID), Function Parameters Incremental Difference(FP-ID), Function Returns Incremental Difference(FR-ID), Function Statements Incremental Difference(FS-ID) against vsftpd version 1.1 releases on horizontal axis.

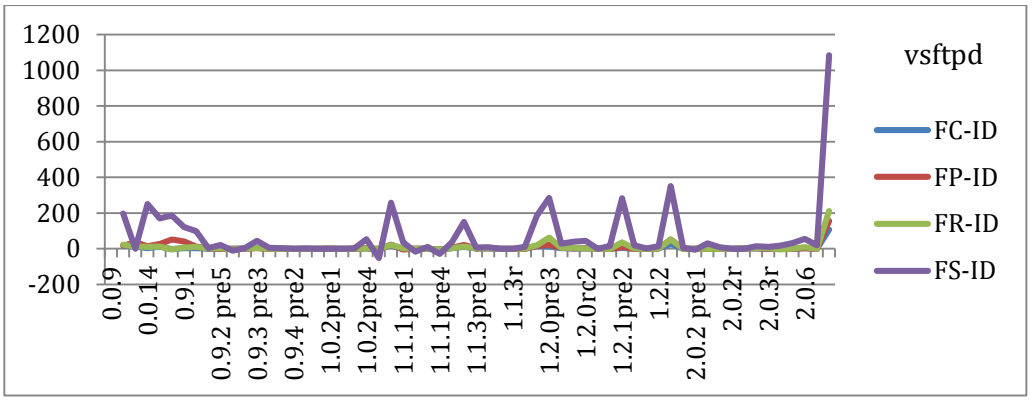

Figure 4. Graph of Function Counts Incremental Difference (FC-ID), Function Parameters Incremental Difference(FP-ID), Function Returns Incremental difference(FR-ID),Function Statements Incremental Difference(FS-ID) against vsftpd version 0.0.9 to version 2.1.0 on horizontal axis.

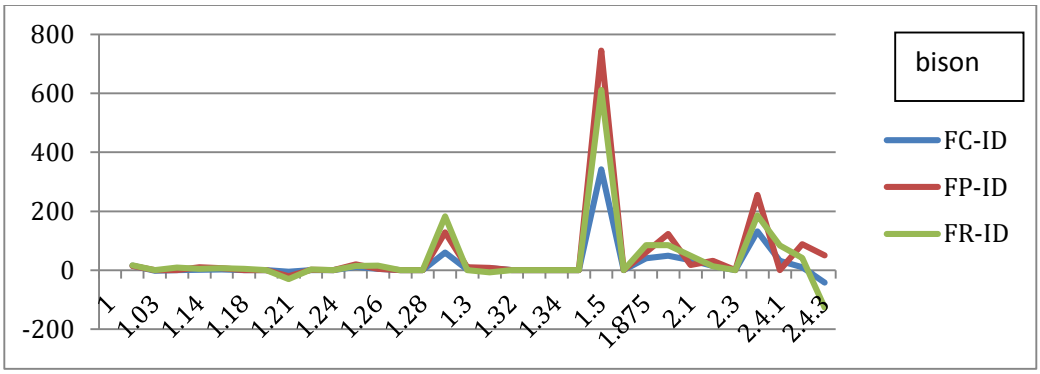

Figure 5a. Graph of Function Counts Incremental Difference(FC-ID), Function Parameters Incremental Difference(FC-ID), Function Returns Incremental Difference(FR-ID) against bison version 1.0 to version 2.4.3 on horizontal axis

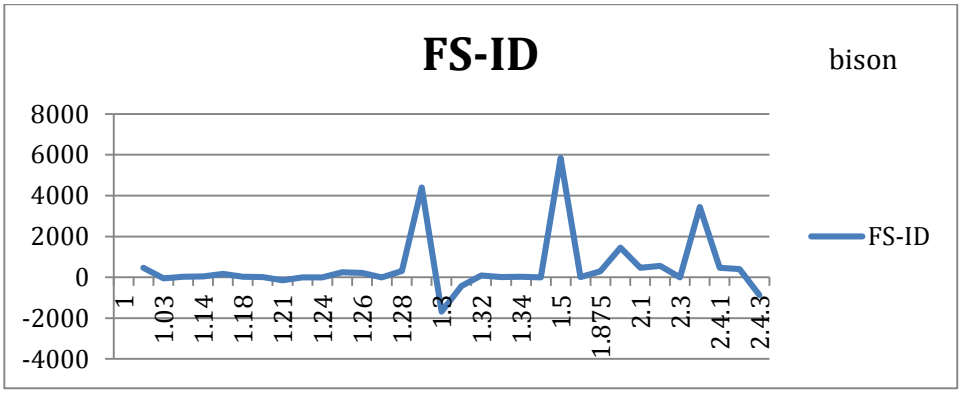

Figure 5b. Graph of Function Statements Incremental Difference (FS-ID) against bison version 1.0 to version 2.4.3 on horizontal axis

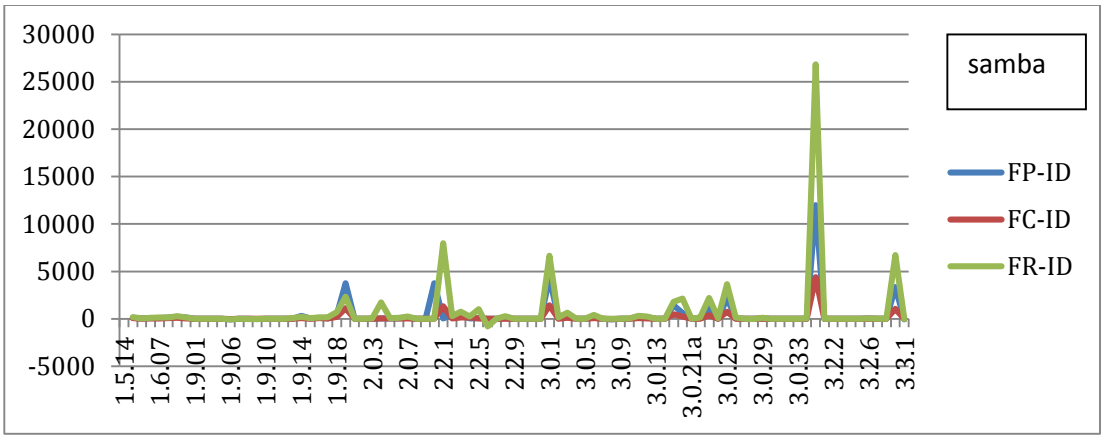

Figure 6a. Graph of Function Parameters Incremental Difference(FP-ID) ,Function Counts Incremental Difference(FC-ID), Function Returns Incremental Difference(FR-ID) against samba version 1.5.14 to version 3.3.1 on horizontal axis

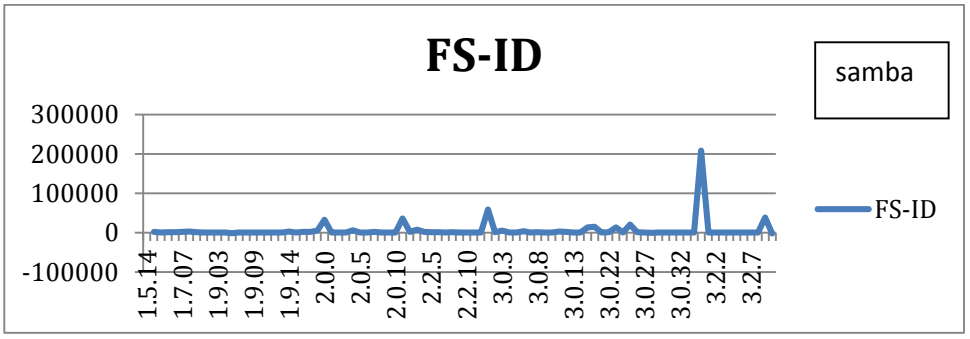

Figure 6b. Graph of Function Statements Incremental Difference (FS-ID) against samba version 1.5.14 to version 3.3.1 on horizontal axis

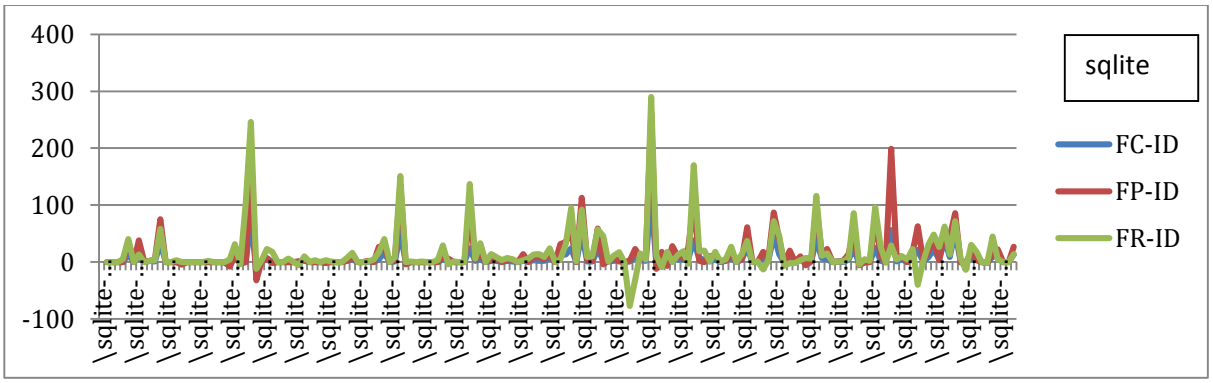

Figure 7a. Graph of Function Counts Incremental Difference(FC-ID), Function Parameters Incremental Difference(FP-ID), Function Returns Incremental Difference(FR-ID) against sqlite version 1.0.0 to version 3.6.11 on horizontal axis

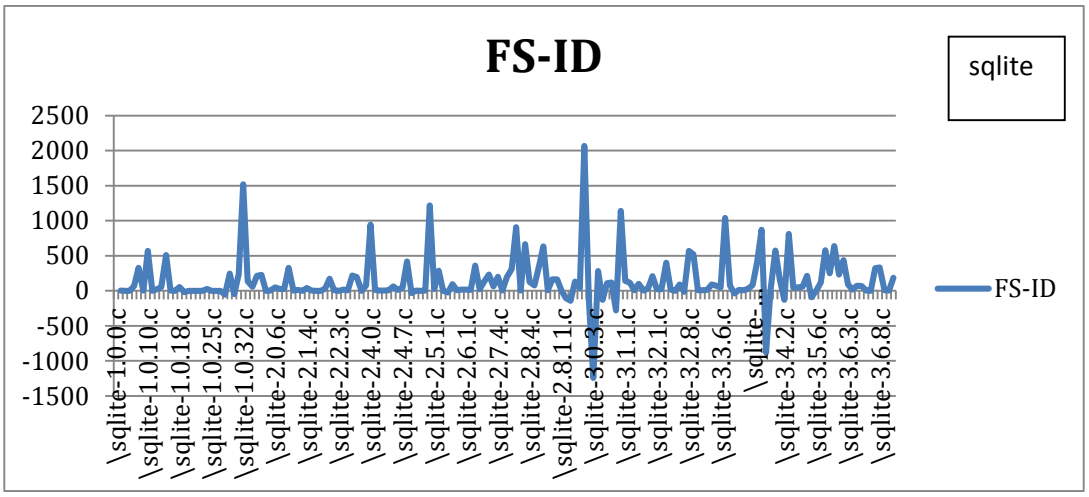

Figure 7b. Graph of Function Statements Incremental Difference against sqlite version 1.0.0 to version 3.6.11 on horizontal axis

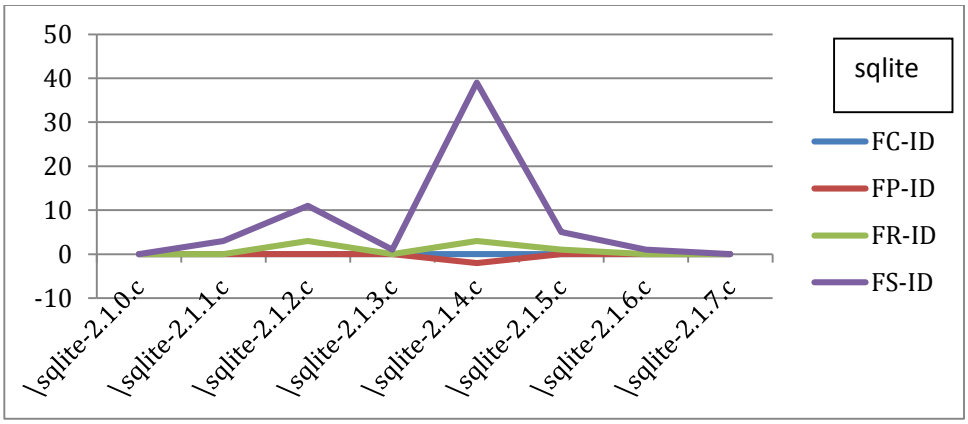

Figure 8. Graph of Function Counts Incremental Difference(FC-ID), Function Parameters Incremental Difference(FP-ID), Function Statements Incremental Difference(FS-ID), Function Returns Incremental Difference(FR-ID) against sqlite version 2.1.0 to version 2.1.7 on horizontal axis

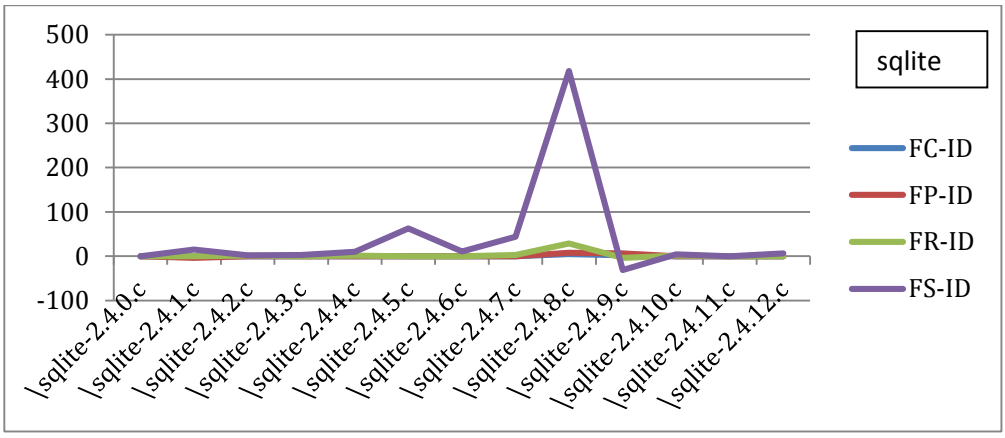

Figure 9. Graph of Function Counts Incremental Difference(FC-ID), Function Returns Incremental Difference(FR-ID), Function Parameters Incremental Difference(FP-ID), Function Statements Incremental Difference(FS-ID) against sqlite version 2.4.0 to version 2.4.12 on horizontal axis

Discussion

In Figures 1, it is seen in vsftpd systems evolution that in pre versions before the first main version 1.0, there were lot of early changes but close to the first major release candidate of v1.0 the changes got vastly reduced as the trend becomes straight on the horizontal axis.

Figures 2a and Figure 2b show changes within vsftpd version 1.0 releases, it is noted that the changes within stable branch of version 1.0 is within a short range i.e. no much changes; precisely in Fig 2a, FC-ID the range is (1- (-1)) = 2; FP-ID is of range (3- (-3)) = 6; FR-ID is (1- (-1)) = 2; FS-ID of Fig.2a is (53 - (-53)) = 106.

Figures 3a Show changes in function components of vsftpd v1.1 releases; it is seen that changes within development branch of vsftpd v1.1 is more than that of vsftpd v1.0. The range of the changes in the components/elements is wider i.e. the highest peak in FS-ID trend in v1.1 is 150 while its lowest trough is -30, the range is (150 - (-30)) = 180, while corresponding range of FS-ID in v1.0 is (53 - (-53)) = 106, range of FC-ID in v1.1 is (12 - (-3)) = 15 while that of v1.0 is (1 - (-1)) = 2, range of FR-ID in v1.1 is (13 - (-4)) = 17 while that of v1.0 is (1-(-1)) = 2 ,this is same with the trend of FP-ID. Also there are peaks and troughs in the Incremental Difference trends of v1.1 than those of v1.0. However, there are continuing changes in both development branch and stable branch.

In Figure4, it is shown that, more changes occur in the function statements than in the interfaces of the function in term of parameters and return. Also, all trends from vsftpd version 0.0.9 to v2.1 is depicted, in contrast to vast reduction in changes prior or close to the first main version 1.0, in the pre versions of a stable release which is to lead to the stable release, more changes do take place close to the stable release candidate version. It is noted that the changes within the stable branches are lower compared to the development branches, but there is continuing change in both.

The standard deviation of the incremental difference of statements in functions of vsftpd from v0.0.9 to v2.1.0 is 161. The Skewness is 4.6 while Kurtosis is 27. Mean is 69. The standard deviation of incremental difference of returns of vsftpd from version 0.0.9 to version 2.1.0 is 29, skewness is 5.99 and kurtosis is 40. Standard deviation is a measure of how widely values are dispersed from average value, so standard deviation of 161 and 29 for incremental difference of functions' statements and returns respectively shows that there is variance/dispersion which depicts continuing change. Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive kurtosis indicates a relatively peaked distribution. Negative kurtosis indicates a relatively flat distribution. The incremental difference of the statements and returns are of 27 and 40 respectively, which shows relative peakedness of

the metrics values distribution. Skewness characterizes the degree of asymmetry of a distribution around its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values. Negative skewness indicates a distribution with an asymmetric tail extending toward more negative values. The skewness of the incremental difference of statements and returns in vsftpd systems are 4.6 and 5.9. The asymmetrical tails for both distributions extend towards positive value. In all the descriptive statistics that have been considered for vsftpd version 0.0.9 to version 2.1.0 i.e. graphs, standard deviation, skewness and kurtosis; changes have been depicted.

Figure 8 above shows graphs of Incremental difference for function counts, parameters, returns, statement for sqlite v2.1 releases (development branch) .The changes in the development branch of sqlite 2.1 is more than what happens in Figure 9 above for stable branch of sqlite 2.4, going by the trends oscillations and variance, more continuing change occurred in sqlite v2.1 releases.

It can be seen in all the trends from Figure 1 to Figure 9, continuing change occurs to all the systems in evolution. Also more functions and invariably more function components are added to the systems as they are changing. Additions are more than deletions. In very large systems like samba (there are versions with more than 1 million LOC), shrinkages/deletions occur less in comparison to averagely large systems like sqlite systems. Changes within a major version amongst the minor versions e.g. within major version 2.4 having minor versions v2.4.1, v2.4.2, v2.4.3 and so on are less compare to changes that occur from major version to major regardless of either development or stable major version i.e. from v2.4 to v2.5 to v3.0 and so on.

Incremental difference of samba (v1.5.14 to v3.6.11) functions statements, returns, parameters, function counts are of mean 5717, 786, 420, 148 respectively and of standard deviation 23805, 3140, 1502, 533 respectively. This shows dispersion or variance which is invariably continuing change. These statistical parameters values are also observed in other softwares of this project case study. In the study of Neamtiu et al. [2013], cumulative number of changes to program elements (i.e., functions, types, and global variables) of 9 evolutionary software systems were quantified

and continuing changes were found in all. Similarly, components of Linux Operating systems (arch and drivers directories) were measured using number of files and Lines of Code metrics by Israeli and Feitelson [2010], they also confirmed continuing changes occuring to in Linux Operating system.

Stable major versions are systems like v1.0, 1.2, 2.2, 2.0, 2.4, 3.0, 3.4, 3.8 etc. Development major versions have the second number in the versioning system to be odd number e.g. v2.1, 2.3, 3.5, 3.7 etc

CONCLUSION

In the pre versions, lot of changes do occur but when it is getting close to first release candidate of version 1, the changes reduce but in stable branch after the first official release more changes do occur close to the release of another stable or development major release.

Changes within stable branch is within a short range i.e. no much changes because simple enhancements, updates, bug fixings are the major things that are usually done in the stable branch.

Changes within development branch are more than that of stable branch because many new small files are created due to new features and functionalities in the development branch.

More changes occur in the function statements than in the interfaces of the function in term of parameters and returns.

Changes within the stable branches are lower compared to the development branches, but there is continuing change in both.

In all the trends, continuing change occurs to all the systems in evolution. Also more functions and invariably more function components are added to the systems as they are changing. Additions are more than deletions. In very large systems like samba (there are versions with more than 1million

LOC), shrinkages/deletions occur less in comparison to averagely large systems like sqlite systems.

Changes within a major version amongst the minor versions e.g. within major version2.4 having minor versions v2.4.1, v2.4.2, v2.4.3 and so on are less compare to changes that occur from major version to major regardless of either development or stable major version i.e. from v2.4 to v2.5 to v3.0 and so on.

Incremental difference of system components such as function statements, returns, parameters and function counts show dispersion from their mean going by their standard deviation parameter. This shows dispersion or variance which is invariably continuing change.

The systems under consideration observed versioning order as follows: stable major versions are systems like v1.0, 1.2, 2.2, 2.0, 2.4, 3.0, 3.4, 3.8 e.t.c. Development major versions have the second number in the versioning system to be odd number.

REFERENCES

ALENEZI, M., AND ALMUSTAFA, K. 2015. Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology*, *8*(2), 257-266.

EICK, S. G., GRAVES, T. L., KARR, A. F., MARRON, J. S., AND MOCKUS, A. 2001. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, *27*(1), 1-12.

FERNANDEZ-RAMIL, J., LOZANO, A., WERMELINGER, M., AND CAPILUPPI, A. 2008. Empirical studies of open source evolution. In *Software evolution* (pp. 263-288). Springer Berlin Heidelberg.

GODFREY, M.W., & TU, Q. 2000. Evolution in open source software: A case study. In *Software Maintenance, 2000. Proceedings. International Conference on* (pp. 131-142). IEEE.

HERRAIZ, I. 2008. A Statistical Examination of the Evolution and Properties of Libre Software. *PhD thesis, Universidad Rey Juan Carlos*, Retrieved from *http://purl.org/net/who/iht/phd.*

ISRAELI, A., AND FEITELSON, D. G. 2010. The Linux kernel as a case study in software evolution. *Journal of Systems and Software*, *83*(3), 485-501.

KOCH, S. 2007. Software evolution in open source projects—a large-scale investigation. *Journal of Software: Evolution and Process*, *19*(6), 361-382.

LAWRENCE, M. J. 1982. An examination of evolution dynamics. In *Proceedings of the 6th international conference on Software engineering* (pp. 188-196). IEEE Computer Society Press.

LEHMAN M. M. 1969. The Programming Process. *IBM Res. Rep. RC 2722, IBM Res. Centre*, Yorktown Heights, NY 10594

LEHMAN, M. M. 1974. Programs, cities, students—limits to growth?. In *Programming Methodology* (pp. 42-69). Springer New York.

LEHMAN, M.M. 1978. Laws of Program Evolution - Rules and Tools for Programming Management, Proc. Infotech State of the Art Conf., Why Software Projects Fail, pp. 11/1-11/25.

LEHMAN, M.M.1979. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software* , 213 – 221.

LEHMAN, M. M. 1980. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076.

LEHMAN, M.M. 1985a. Program Evolution. In *Program Evolution. Processes of Software Change*, M. M.Lehman and L. A. Belady (Eds.). Academic Press Professional, Inc., San Diego, CA, USA, 9–38.

LEHMAN, M.M. 1985b. The Programming Process. In *Program Evolution. Processes of Software Change*, M. M. Lehman and L. A. Belady (Eds.). Academic Press Professional, Inc., San Diego, CA, USA, 39–84.

LEHMAN, M.M. 1985c. Programs, Cities, Students: Limits to Growth? In *Program Evolution Processes of Software Change*, M. M. Lehman and L. A. Belady (Eds.). Academic Press Professional, Inc., San Diego, CA, USA, 133–164.

LEHMAN, M.M. 1989. Uncertainty in computer application and its control through the engineering of software. *Journal of Software Maintenance: Research and Practice,* 1,1, 3–27.

LEHMAN, M.M. 1990. Uncertainty in Computer Application. *Communications of ACM* 33, 5, 584–586. Technical letter.

LEHMAN, M.M., PERRY, D.E., AND TURSKI, W. M. 1996.Why is it so hard to find Feedback Control in Software Processes?, Invited Talk, Proc. of the 19th Australasian Comp. Sc. Conf., Melbourne, Australia, pp. 107-115.

LEHMAN, M.M. 1996b. Laws of Software Evolution Revisited. In Proceedings of the European Workshop on Software Process Technology. Springer-Verlag, London, UK,
108–124.

NEAMTIU, I., XIE, G., & CHEN, J. 2013. Towards a better understanding of software evolution: an empirical study on open-source software. *Journal of Software: Evolution and Process*, *25*(3), 193-218.

PIRZADA, S. S. 1998. *A statistical examination of the evolution of the UNIX system* (Doctoral dissertation, Imperial College London (University of London)).

SJØBERG, D. I., DYBÅ, T., ANDA, B. C., &HANNAY, J. E. 2008.Building theories in software engineering. In Guide to advanced empirical software engineering (pp. 312-336).
Springer London.

VASA, R. 2010. Growth and change dynamics in open source software systems.
Ph.D. Dissertation. Swinburne University of Technology, Melbourne, Australia.