International Journal of Hybrid Information Technology
Vol.8, No.2 (2015), pp.257-266
http://dx.doi.org/10.14257/ijhit.2015.8.2.24

Empirical Analysis of the Complexity Evolution in Open-Source Software Systems

Mamdouh Alenezi and Khaled Almustafa

Prince Sultan University, Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa kalmustafa@psu.edu.sa

Abstract

When the software system evolves, its scale is increasingly growing to the degree where it is very hard to handle. Measuring the internal quality of the source code is one of the goals of making software development an engineering practice. Source Lines of Code (SLOC) and Cyclomatic Complexity (CC) are usually considered indicators of the complexity of a software system. Software complexity is an essential characteristic of a software system where it plays an important role in its success or failure. Although understanding the complexity is very important, yet it is not clear how complexity evolves in open source systems. In this paper, we study the complexity evolution of five open source projects from different domains. We analyze the growth of ten releases of these systems and show how complexity evolves over time. We then show how these systems conform to the second Lehman's law of software evolution.

Keywords: *Software Quality, Software Metrics, Software Complexity, Open Source, Software Evolution.*

1. Introduction

Software evolution is the software system dynamic behavior while it is maintained and improved over its lifetime [15]. Software systems usually evolve to x problems, accommodate new features, and improve their quality. In order for the software to survive for a long period, it needs to evolve. The changes that the software undergo are formally categorized as corrective, preventive, adaptive and perfective maintenance.

One of the maintainability characteristics of the ISO/IEC SQuaRe quality standard is Modularity[13]. In this standard, modularity is de ned as "a degree to which a given system or a program is composed of discrete components; such that a change in any component has minimal impact on the other components" [13]. Modularity is further decomposed at that standard into two main dimensions, coupling and complexity.

McCabe's Cyclomatic Complexity (CC), first introduced in 1976 [20], measures the software complexity using its flow graph. In practice, CC counts the decision points in the software. The CC metric definitions are independent of the syntax and the productivity differences among programming languages [34]. Complexity measures identify components that have the highest complexity. Most likely, these components contain bugs, because complexity makes them harder to understand [27].

Complexity measures are very useful in identifying complex program units. Numerous research studies investigated the relationship between program complexity and maintain-ability [27]. These studies have found that complex systems requires a lot of e ort in maintaining them. Simplifying your components usually leads to reducing the maintenance

ISSN: 1738-9968 IJHIT

cost. Large values for CC hinder the software evolution since they make the program both di cult to change and to understand [23].

The cyclomatic complexity measure is an internal attribute of the software code, which is different from external attributes; such as productivity and e ort. Several research studies built models from internal attributes trying to predict external attributes. Internal and external attributes and their relationships can be sometime intuitive (a.e, more complex code will require greater e ort to maintain). Certain problems have intrinsic complexity level in which developers should try harder to decompose and simplify. Especially, when solving a complex problem with software and the solution adds another level of complexity. Unfortunately, when the problem complexity increases, this leads to an increase in the complexity of the software, which results in reducing its quality [8].

Software complexity is considered a major player in the software engineering research and practice [7, 6], and it is the result of the design decision, algorithm choices, and the implementation details, also it can be used to predict several quality characteristics; such as maintainability [18, 6]. For example, high complex modules have been found to be most prone to faults [30]. Therefore, monitoring and controlling of a given software is very essential to the its project success.

Most research work that examined software complexity was focused on closed-source software systems [2]. Open-source software development, which is mostly done by volunteer developers, has some unique characteristics that may increase the importance of studying their complexity. These developers who are geographically distributed work with the lack of standards and processes that ensure reducing the software complexity. Software engineering research showed that the documentation availability usually improves the quality of both design and code of complex functions [6], most open-source projects lack traditional documentation of requirements or design [29].

Developers have difficulty in understanding complex software units, and the complexity might vary between different implementations to solve the same problem [6]. In order for the software to evolve, software has to be updated regularly with added features, which in part add more complexity to the software [6]. However, it is essential to strive for minimizing the complexity to facilitate adding features, or xing bugs in the future [6]. Manduchi and Taliercio [19] investigated the complexity evolution of closed-source systems in which they showed that almost all complexity measures increased with different pace. MacCormack et. al [18] found that restructuring the software design of Mozilla reduced the complexity which is already established by one of the Lehman's laws, which states that complexity will increase unless precautionary measures are implemented. The question remains valid, does it always true that adding more functionalities leads to increase in the complexity?

Software complexity serves as both, indicator of quality and success of an open-source project, and contributor to its ability to attract market share and community's input [1]. Empirically investigating complexity is very beneficial in comprehending and understanding the mechanism of the evolution of complexity in open-source projects and how it is managed. Particularly, size and complexity are two aspects of complexity which known to be good predictors of external attributes; such as flexibility and maintainability [23].

The remainder of this paper is organized as follows. Section 2 states Lehman's laws of software evolution. Section 3 discusses the previous open-source projects studies. Section 4 states the metrics used in this study. The empirical study is given in Section 5. Some threats to validity are presented in Section 6. Conclusions are presented in Section 7.

2. Lehman's Laws of Evolution

Lehman is known for his introduction of laws regarding systems evolution. In Lehman's vision, as soon as you deploy the software, the underlying environment changes. Henceforth, in order for the software to operate in the real word, it should evolve to adapt to the environment as it evolves in responding to the user's feedback, which further drive the software evolution. According to Lehman, the task of successfully evolving a given software system is not an easy task. Lehman outlined his views about E-type software systems and came up with Lehman's Laws of Software Evolution [11]:

1. Continuing change - A software system will turn into gradually less satisfying to its users over time, if not continually adapted to meet new needs.
2. Increasing complexity - A software system will turn into gradually more complex over time, if not explicit work is done to reduce its complexity.
3. Self-regulation - The software evolution process is self-regulating, very close to normal distribution of the product and process artifacts that are produced.
4. Conservation of organizational stability - The average effective global activity rate on an evolving software system does not change over time; that is, the amount of work that goes into each release is about the same.
5. Conservation of familiarity - New changes in each successive release of a software system inclines to either stay constant or decrease over time.
6. Continuing growth - Functionality of a software system will keep growing to satisfy its users.
7. Declining quality - A software system will be perceived as declining in quality over time, if not carefully maintaining its design and adapted to new operational constraints.
8. Feedback System - Successfully evolving a software system requires recognition that the development process is a multi-loop, multi-agent, multi-level feedback system.

In this paper, we only focus on the second law of Lehman's laws. Several studies investigated these laws and how they apply to open-source systems. A recent systematic literature review [31] was conducted to review e orts in that direction. Based on the results of that study, the results revealed that open-source systems both conform and refute the second Lehman's law (Increasing Complexity) [31]. The second Lehman law for software evolution states that "as a software evolves, its complexity increases, unless proactive measures are taken to reduce or stabilize the complexity". We selected 10 releases of well-known open-source systems in order to investigate if the second law actually conforms with open-source systems.

3. Studies on Open-Source Projects

Open-source projects are usually software development projects based on a methodology that has several properties, which are not found in commercial software development. The availability of the source code to everyone [24]. If you contribute to any of the open-source systems, you will join their community. The open-source community usually shares

information with its members through email, mailing list, forum etc. Usually the development process of open source project is an ad hoc process; and no formal process is followed [5].
Several studies were carried to investigate and find the success factors of open-source projects. These studies can be classified into three different categories. The first category studied some different successful open-source projects, OpenBSD [16], FreeBSD [9], Apache [21], and Debian GNU/Linux [28]. The second category studied the similarities of the process used by successful open-source projects, Arla and Mozila Projects [4], Apache and Mozilla [22], and fifteen OSS Projects [32]. Saini and Kaur [25] reviewed and compared different open source software life cycles found in the literature. The third category focuses on the community side of open source projects [5, 35]. Several research studies pointed that in a certain phase of open-source projects, the complexity will increase to a level where it would not be possible for the open-source community to handle [12, 28]. The nature of the ad hoc development is another factor for the quality to decline [28], the coupling to increase [3], the source code to degrade [10], the lack of a formal process [5] and the poor architectural design [9]. For the open-source community, several problems faces their communities, frequent turnovers of volunteers [10] and only few open-source projects attract enough help to be developed appropriately [35].

4. Studies on Open-Source Projects

We used mainly in this study two metrics Source Lines of Code (SLOC) and Cyclomatic Complexity (CC). We further explain these metrics in more details:

1. Source Lines of Code (SLOC): it is whichever a line of code that has programing text (not a comment, or blank line) irrespective of the number of fragments of statements on that line. This includes all line that has any text in them, executable and non-executable statements [14].
2. Cyclomatic Complexity (CC): it is the largest number of linearly independent circuits in the control flow graph of a certain program, where each exit point is connected with an additional edge to the entry point. CC is computed by calculating forks (if, while, etc) in a control flow graph and adding 1 [14]. In any structured program, the cyclomatic complexity is equal to the number of decision points contained in that program plus one. In other words, it calculates how many decisions caused by conditional statements in the source code.

5. Empirical Study

This section discusses the empirical work done in this study. We discuss the selection criteria for the systems, and then we report the findings.

5.1. Selected Systems

To select the systems for the empirical analysis, four selection criteria have been used. First, the selected systems had to be well-known systems that are very widely used. Second, the systems had to be sizeable, so the systems can be realistic and have multi-developers. Third, the systems had to be actively maintained. Finally, the data of these systems had to be publicly available.

Table 1. Selected Software Systems

System	Versions	Years	SLOC	Classes

JabRef	2.1-2.10	Aug 2006-Mar 2014	46357-93158	376-694
CheckStyle	5.2-6.1	Sep 2010-Nov 2014	22739-27556	269-301
Jajuk	1.1-1.10	Jan 2006-Aug 2012	25300-53155	167-404
jEdit	3.0-5.0	Dec 2000-Nov 2012	26868-109201	139-538
Cobertura	1.1-2.0	Mar 2005-May 2013	3249-50948	29-97

Five various-sized systems have been chosen from different domains. Characteristics of the selected software systems are listed in Table 1. JabRef is a reference management software that provides a user-friendly interface to edit BibTeX les, to import data from online scientific databases, and to manage and search BibTeX les. Checkstyle is a medium-sized, development tool that checks whether a code conforms to a coding standard by automating the process of checking Java code. It enforces developers to follow certain coding standards. Jajuk, supports all platforms, is a music organizer that provides a plenty of features help users manage their large or scattered music collections. jEdit is a medium-sized, text editor. It focuses on providing different features for developers, including macro scripting, syntax highlighting, and a comprehensive plug-in environment. Cobertura is a code coverage tool that calculates the percentage of code accessed by tests. It identifies that parts of your Java program which are lacking test coverage.

5.2. Data Collection Process

10 releases for each selected system were downloaded from the sourceforge repository. Scitools Understand (version 3.1) was utilized in extracting several measures from the source code. Size (SLOC) was determined after summing the total number of SLOC. To determine a release level average cyclomatic complexity measure, the class level average complexity measures were averaged across all classes for the given release of the project. To determine a release level maximum cyclomatic complexity measure, the class level maximum complexity measures were averaged across all classes for the given release of the project.

5.3. Size Evolution

Figure 1 shows the size evolution of the five selected systems. Figure 1 (a), (b) and (c) graphs show almost linear change in the SLOC size of JabRef, CheckStyle and Jajuk systems, and it can be estimated as an increase of 97.7 %, 21 % and 108 % of the SLOC size respectively, and that is over the period of ten releases for these systems. Figure 1 (d) graph shows rapid increase in the first seven releases for jEdit software, for an estimated increase of 354 % of the SLOC size, then a linear increase for the last three releases with an estimated increase of 10 %, and for a total estimated increase of SLOC size of 400 % over the ten releases of the jEdit software. Figure 1 (e) graph shows no increase in the SLOC size for the first seven releases of the Cobertura software, and then a rapid increase in the SLOC size in the last three releases for an estimated and overall increase of 1566 % in the SLOC size for the entire period.

We can conclude from the mentioned analysis that the studied systems witnessed a great deal of changes in their averaged SLOC size over the period of ten releases for each one of the mentioned systems.

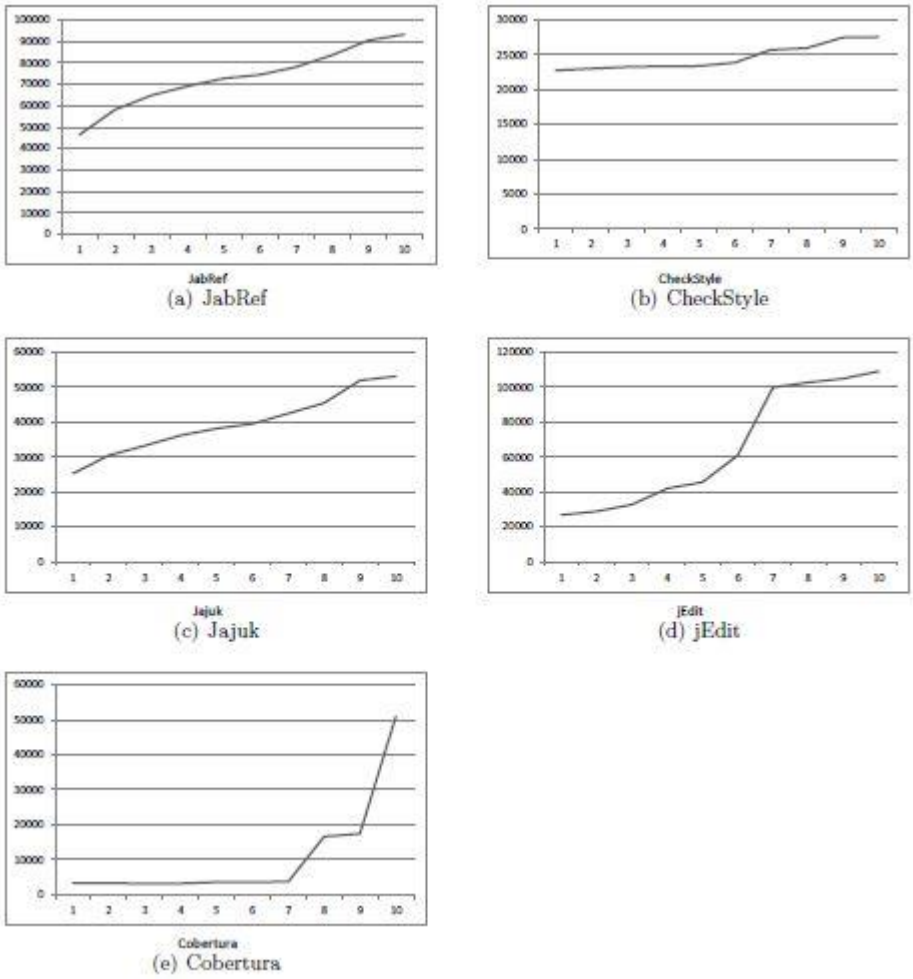

Figure 1. SLOC Evolution of the Selected Systems

5.5. Discussion

Based on the obtained results, it is clear that the functionality of these system are continually increasing which conforms with sixth law of Lehman's laws (Continuing growth). The constant increase in SLOC indicates that these systems become more complex while they evolve. The cycolmatic complexity stated almost the same while the system evolves, but this indicates that the complexity did not decrease. Having almost the same complexity level with a constant increase in size indicates that there is no extra precaution measures to decrease the complexity of these systems. A recent study by Singh and Bhattacherjee [26] studied the complexity evolution of only one open source system namely JFreeChart. They have found similar conclusion. However, relying on the results of one system cannot be generalized to other systems from different domains.

5.4. Complexity Evolution

Figure 2 shows the Cyclomatic Complexity (CC) evolution of the five selected systems. All graphs in Figure 2 show no change in the average CC for all the selected systems, and no change in the maximum CC evolution for JabRef, CheckStyle systems as seen in Figure 2 (a) and (b) throughout the ten releases. Jajuk and jEdit software show some variation in the maximum CC evolution over the same period as seen in Figure 2 (c) and (d) for an estimated change of 10 % and 14 % respectively. Cobertura software shows a large increase in the maximum CC evolution after the seventh release, as seen in Figure 2 (e), and that is due to the sudden and rapid increase in the SLOC size as it was seen in Figure 1 (e).

It is clear from this analysis that the maximum CC evolution for the mentioned systems is not necessary evolving with the increase of the SLOC size, and the average CC evolution is not being effected over the entire life-span of the mentioned releases for all the studied systems.

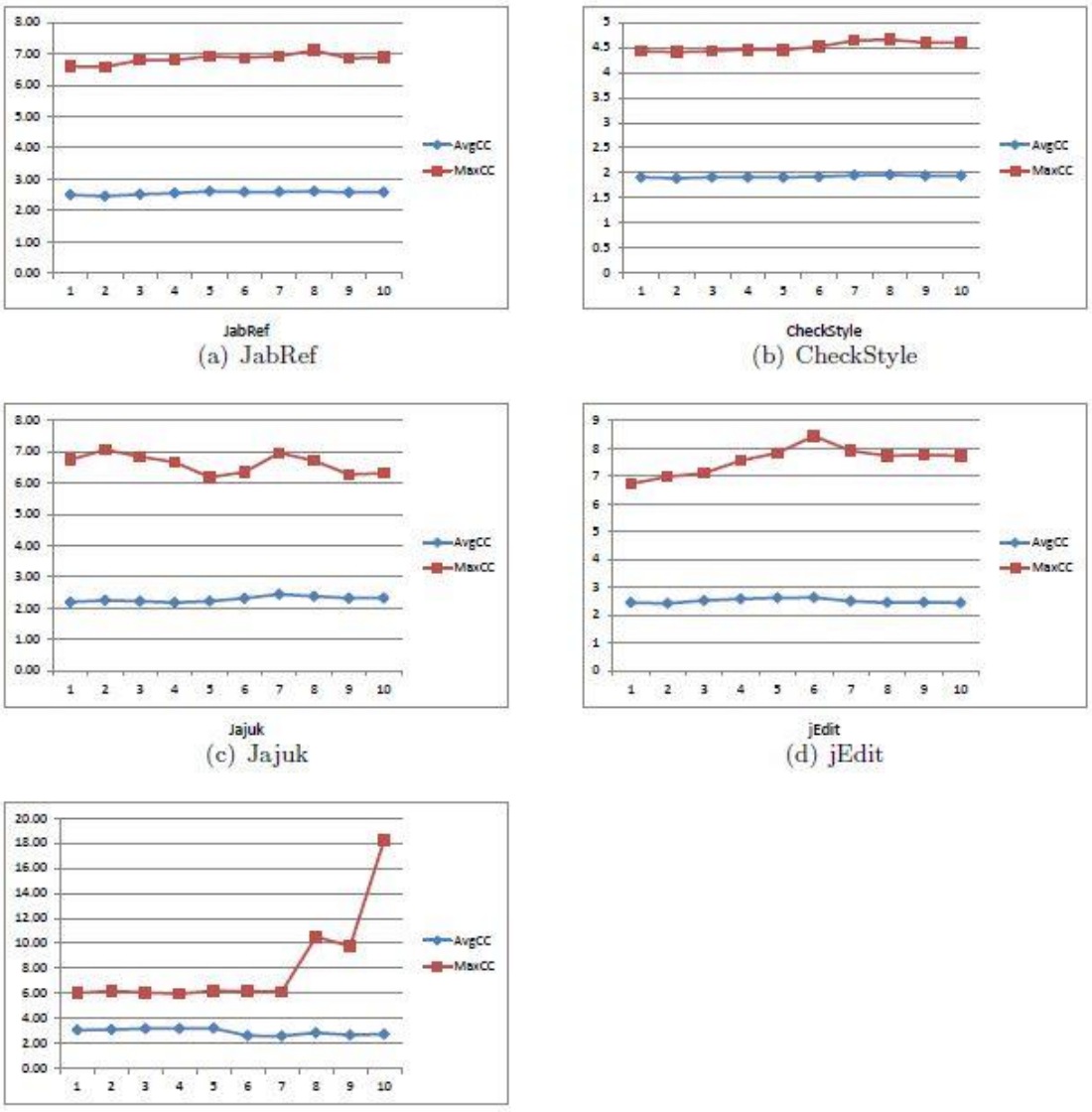

Figure 2. Cyclomatic Complexity Evolution of the Selected Systems

6. Threats to Validity

Threat to validity is very common in empirical studies. In this section, we report threats to validity based on the framework for software engineering empirical studies presented in [33]. The validity of the results obtained in this work is constrained by a number of aspects.

Since our research is categorized as action research, the major threat is external validity. Since we chose five systems from different domains, we conjecture that this study results can be generalized to more contexts of software development. As for the internal validity, we used a very popular software tool to collect the metrics from the software releases. This tool has been empirically validated and compared with other tools [17]. For the construct validity, we identify classes to be les in the release. It is well-know that Java classes correspond to actual les in the source code. Several releases have been manually checked in order to make sure that the tool actually identifies the right number of classes.

7. Conclusion

Software complexity is an important software property for the software engineering researches. The complexity has been identified as the source of most software bugs. In this work, we conducted an empirical investigation of complexity evolution on five well-known different-sized different-domain open source systems. We showed that these systems conform to the second law of Lehmans laws of evolution. We studied both the increase in size (SLOC) and the increase in cyclomatic complexity (CC). The results indicate that Continuing Change, Increasing Complexity, and Continuing Growth are applicable to open-source systems. Future directions include investigating how the complexity of software correlate with fault-proneness. Does an increase in complexity result in increase in number of faults?

References

[1] S. Androutsellis-Theotokis, D. Spinellis, M. Kechagia, G. Gousios, S. Evdokimov, B. Fabian, O. G•unther, L. Ivantysynova, H. Ziekow and M. A Lapre, "Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management*, vol. 4, nos. 3-4, (**2011**), pp. 187-347.
[2] E. J. Barry, C. F. Kemerer, and S. A Slaughter, "How software process automation affects software evolution: a longitudinal empirical analysis", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 1, (**2007**), pp. 1-31.
[3] S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler, "A feedback based quality assessment to support open source software evolution: the grass case study", In *22nd IEEE International Conference on Software Maintenance*, 2006. ICSM'06., pages 155-165. IEEE, (**2006**).
[4] A. Capiluppi and J. F. Ramil, "Studying the evolution of open source systems at different levels of granularity: Two case studies", In Proceedings. *7th International Workshop on Principles of Software Evolution*, 2004., pages 113-118. IEEE, (**2004**).
[5] S. Christley and G. Madey, "Analysis of activity in the open source software development community", In *40th Annual Hawaii International Conference on System Sciences*, 2007. HICSS 2007, pages 166b-166b. IEEE, (**2007)**.
[6] D. P. Darcy, S. L. Daniel, and K. J. Stewart, "Exploring complexity in open source software: Evolutionary patterns, antecedents, and outcomes. In *43rd Hawaii International Conference on System Sciences* (HICSS), 2010, IEEE, pp. 1-11, (**2010**).
[7] D. P. Darcy and C. F. Kemerer, "OO metrics in practice", *IEEE Software*, vol. 22, no. 6, (**2005**), pp. 17-19.
[8] D. P. Darcy, C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko, "The structural complexity of software an experimental test", *IEEE Transactions on Software Engineering*, vol. 31, no. 11, (**2005**), pp. 982-995.
[9] T. Dinh-Trong and J. M. Bieman, "Open source software development: a case study of freebsd", In *Proceedings. 10th International Symposium on Software Metrics*, IEEE, (**2004**), pp. 96-105.
[10] H. JC Ellis, R. A Morelli, T. R de Lanerolle, J. Damon, and J. Raye, "Can humanitarian open-source software development draw new students to cs?", In *ACM SIGCSE Bulletin*, vol. 39, ACM, (**2007**), pp. 551-555.

[11] M. W. Godfrey and D. M German, "On the evolution of lehman's laws", *Journal of Software: Evolution and Process*, (**2013**).
[12] V. K. Gurbani, A. Garvert, and J. D. Herbsleb, "A case study of open source tools and practices in a commercial setting", In *ACM SIGSOFT Software Engineering Notes*, ACM, vol. 30, (**2005**), pp. 1-6.
[13] ISO/IEC. Systems and software engineering - systems and software quality requirements and evaluation (square). ISO/IEC 25010 - System and software quality models, (**2011**).
[14] D. Landman, A. Serebrenik, and J. Vinju. 'Empirical analysis of the relationship between cc and sloc in a large corpus of java methods", In *IEEE International Conference on Software Maintenance and Evolution* (ICSME), IEEE, (**2014**), pp. 221-230.
[15] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle", *Journal of Systems and Software*, vol. 1, (**1980**), pp. 213-221.
[16] P. L. Li, J. Herbsleb, and M. Shaw, "Finding predictors of field defects for open source software systems in commonly available data sources: A case study of openbsd", In *11th IEEE International Symposium Software Metrics*, IEEE, (**2005),** pp. 10.
[17] R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools", In *Proceedings of the 2008 international symposium on Software testing and analysis*, ACM, (**2008**), pp. 131-142.
[18] A. MacCormack, J. Rusnak, and C. Y Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code", *Management Science*, vol. 52, no. 7, (**2006**), pp. 1015-1030.
[19] G. Manduchi and C. Taliercio, "Measuring software evolution at a nuclear fusion experiment site: a test case for the applicability of oo and reuse metrics in software characterization", *Information and Software Technology*, vol. 44, no. 10, (**2002)**, pp. 593-600.
[20] T. J McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol. 4, (**1976**), pp. 308-320.
[21] A. Mockus, R. T Fielding, and J. Herbsleb, "A case study of open source software development: the apache server", In Proceedings of the *22nd International Conference on Software engineering*, ACM, (**2000**), pp. 263-272.
[22] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology* (TOSEM), vol. 11, no. 3, (**2002**), pp. 309-346.
[23] I. Neamtiu, G. Xie, and J. Chen, "Towards a better understanding of software evolution: an empirical study on open-source software", *Journal of Software: Evolution and Process*, vol. 25, no. 3, (**2013**), pp. 193-218.
[24] Er. S. Raymond and T. Enterprises, "The cathedral and the bazaar", (**2012**).
[25] M. Saini and K. Kaur, "A review of open source software development life cycle models", *International Journal of Software Engineering & Its Applications*, vol. 8, no. 3, (**2014**).
[26] V. Singh and V. Bhattacherjee, "A new measure of code complexity during software evolution: a case study", *International Journal of Multimedia & Ubiquitous Engineering*, vol. 9, no. 7, (**2014**).
[27] I. Sommerville, "Software Engineering", Addison-Wesley, vol. 9, (**2010**).
[28] S. Spaeth and M. Stuermer, "Sampling in open source development: The case for using the debian gnu/linux distribution", In Proceedings of the *40th IEEE Hawaii International Conference on System Sciences*, (**2007**), p. 166a.
[29] I. Steinmacher, M. A. Graciotto Silva, M. Aurelio Gerosa, and David F Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects", *Information and Software Technology*, (**2014**).
[30] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects", *IEEE Transactions on Software Engineering*, vol. 29, no. 4, (**2003**), pp. 297-310.
[31] M. Syeed, I. Hammouda, and T. Syatä. "Evolution of open source software projects: A systematic literature review", *Journal of Software*, vol. 8, no. 11, (**2013**), pp. 2815-2829.
[32] G. von Krogh, S. Spaeth, and S. Haefliger, "Knowledge reuse in open source software: An exploratory study of 15 open source projects", In Proceedings of the *38th Annual Hawaii International Conference on System Sciences*, 2005. HICSS'05, IEEE, O(**2005**), pp. 198b-198b.
[33] C. Wohlin, P. Runeson, M. Höst, M. C Ohlsson, B. Regnell, and A. Wesslen, "Experimentation in software engineering", Springer, (**2012**).
[34] S. Yu and S. Zhou, "A survey on metric of software complexity", In The 2nd *IEEE International Conference on Information Management and Engineering* (ICIME), 2010, IEEE, (**2010**), pp. 352-356.
[35] F. Zou, "A model of bug dynamics for open source software", In 2008 *Second International Conference on Secure System Integration and Reliability Improvemen*t, (**2008**), pp. 185-186.

