Int J Software Informatics, Volume 7, Issue 3 (2013), pp. 469–481 Tel: +86-10-62661040
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org

Email: ijsi@iscas.ac.cn, ijsi2007@gmail.com

An Empirical Study of Lehman's Law on Software Quality Evolution

Liguo Yu[1] and Alok Mishra[2]

[1] (Computer Science and Informatics, Indiana University South Bend, South Bend, IN, USA)
[2] (Department of Computer & Software Engineering, Atilim University, Incek, Ankara, Turkey)

Abstract Software systems must change to adapt to new functional requirements and nonfunctional requirements. According to Lehman's laws of software evolution, on the one side, the size and the complexity of a software system will continually increase in its life time; on the other side, the quality of a software system will decrease unless it is rigorously maintained and adapted. Lehman's laws of software evolution, especially of those on software size and complexity, have been widely validated. However, there are few empirical studies of Lehman's law on software quality evolution, despite the fact that quality is one of the most important measurements of a software product. This paper defines a metric—accumulated defect density—to measure the quality of evolving software systems. We mine the bug reports and measure the size and complexity growth of four evolution lines of Apache Tomcat and Apache Ant projects. Based on these studies, Lehman's law on software quality evolution is examined and evaluated.
Key words: software evolution; software quality evolution; Lehman's laws of software evolution; mining bug reports; empirical study

Yu L, Mishra A. An empirical study of Lehman's law on software quality evolution. *Int J Software Informatics*, **Vol.7, No.3 (2013): 469–481. http://www.ijsi.org/1673-7288/7/i152.htm**

1 Introduction

Software systems must continually evolve to adapt to new requirements or new environments. In their series studies, Lehman et al. presented eight laws of software evolution[1–7]. Three of them are related with the evolution of the complexity/size and quality of a software system, which can be summarized as follows: the functional capability of a software system must be continually enhanced to maintain user satisfaction; and as a result (1) both the system's size and complexity will be increasing with time; (2) the system's quality will be declining with time unless the system is rigorously monitored and adapted to these changes.

Lehman's laws of software evolution have been examined and validated in many systems and many applications. For example, Israeli and Feitelson studied 810 versions of the Linux kernel and characterized the system's evolution patterns[8]. They investigated different possible interpretations of Lehman's laws, as reflected by different metrics. Barry and Kemerer presented an empirical study of commercial software applications to test and understand how software evolves over time[9].

Corresponding author: Liguo Yu, Email: ligyu@iusb.edu
Received 2011-12-09; Revised 2013-04-09; Accepted 2013-04-22.

Their results support most of the Lehman's laws of software evolution. Herraiz et al. studied the evolution of a large sample of programs, where they found the evolution patterns in both the number of lines of code and the number of files are same, and some patterns do not conform to Lehman's laws of invariant growth rate of software systems[10]. Godfrey and Tu studied the evolution of open source software systems and found that several open-source software systems appear not to obey some of Lehman's laws of software evolution, and that Linux in particular is continuing to grow at a geometric rate[11,12]. Similar results are also reported by other studies to indicate some open-source systems grow at a super-linear rate[13,14]. Simmons et al. presented a case study of Nethack, an open source game product[15]. Their results demonstrated that the evolution patterns observed in Nethack do not consistently conform to Lehman's laws of size and complexity growth.

Quality is undoubtedly one of the most important measurements of a software product. However, little research has been done to empirically study the evolution of software quality, especially the continually evolving and frequently releasing software systems. The only studies we could find are performed by Eick et al.[16] and Lee et al.[17]. In Eick et al.'s study, they defined the concept of code decay (code quality decreasing) and found that in general software maintenance becomes more time and effort consuming, which is the evidence of code decay. In a more recent study, Lee et al. investigated the evolution of JFreeChart, an open source software system. They used fan-in coupling and fan-out coupling to measure software quality and found added class group has higher fan-in coupling and lower fan-out coupling than removed class group, which indicates software quality is increasing with time. They concluded their observation is against Lehman's law of quality evolution.

Despite the fact that Lehman's laws of software evolution have been widely accepted and become the basic knowledge of software engineers, there has been no systematic work to validate its law on software quality evolution. The objective of this study is to re-examine Lehman's law on software quality evolution on two different open-source software systems using different quality metrics. To achieve this goal, a series of studies have been performed to determine the accurate software quality measurement[18]. In this paper, we first define a generic quality metric for continually evolving and frequently releasing software systems. We then mine the bug reports and measure the size and complexity of four evolution lines of open-source Apache Tomcat and Apache Ant projects. Finally, Lehman's law on software quality evolution is examined and evaluated.

The remaining of the paper is organized as follows. Section 2 describes the background knowledge of this study. Section 3 presents the generic software quality metric—*accumulated defect density.* Section 4 describes the data source and data mining process. Section 5 presents the results and the analysis of the case studies. Conclusions are in Section 6.

2 Background

Lehman's laws of software evolution state that software systems must continually grow. This is represented as the regular addition of new features, which can satisfy either new functional requirements or new nonfunctional requirements. In open-source software systems, this phenomenon is represented as the frequent

release of new versions of one product. Each release results in the increase of system size and complexity. As has been stated by Lehman and widely observed by others, in the software evolution process, both the complexity and the size of the product will increase with time. However, the quality change of a software product is unknown, or at least not conclusive. Lehman's law No 7 states "*Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining*"[3]. If this statement is carefully examined, it can be seen that the software quality decreasing with time is under certain circumstance, i.e., the software product is not rigorously adapted to changes. However, in real world software systems, adapting to changes is almost always the first priority in software maintenance. We do not know whether the quality of a software product will decline or improve even if adaption to changes is performed.

On the other hand, there are so many different definitions and metrics of software quality, i.e., software quality could mean many different aspects of a software product. For example, structure quality could be measured with couplings and cohesions between software components; code quality could be measured with readability and reusability; nonfunctional quality could be measured with its reliability, efficiency, security, and maintainability, and so on. Therefore, the major obstacle to universally validating Lehman's law on software quality evolution is the lack of a generic definition and measurement of software quality.

It is interesting to notice that Lehman did not specify what kind of software quality his law is applicable to, which could be the reason why Lehman's law on software quality evolution has not been widely validated yet. Therefore, a generic software quality measurement across domains and applications should be defined first in order to broadly validate Lehman's laws of software quality evolution.

3 A Generic Software Quality Metric

As described before, there are many different ways to defining software quality. For example, readability, reusability and maintainability are measures of design quality; usability and portability are measures of user satisfaction; and reliability and security are measures of software performance. In this paper, software quality is measured with the number of bugs detected/reported in a software product. Software bugs have been long considered the most important issue in software quality. Quality metrics based on the measurement of software bugs are universal and generic, because they are measurable in and applicable to all software products, independent of domains, applications, architectures, and implementations. In contrast, quality measurements based on design, user satisfaction, maintenance, or performance are dependent on specific products and the availability of specific documents.

To measure the evolution of software quality is to measure the quality changes of evolving software products. Continually evolving software systems, especially open-source software systems, are frequently changed and released. On the one side, a change could fix a bug or add a new feature to the system. On the other side, a change might introduce new bugs to the system. Consider a software evolution branch with six releases as shown in Fig. 1, where bugs are reported to each of the

six versions.

First, we use the number of bugs (bug reports) to represent the product quality of each release and give the following definition.

Definition 1. *For a continually evolving software product that has released n versions* $(v_1, v_2, \ldots, v_n)$*, where each new release is based on its previous release, the product quality of version* v_i $(1 \leqslant i \leqslant n)$ *can be measured with the number of bugs reported to version* v_i*.*

Based on Definition 1, the quality measures of Versions V_1 through V_6 of the product in Fig. 1 are 2, 1, 1, 3, 4, and 2, respectively. Definition 1 is based on two assumptions: (1) all the bugs reported in current version V_i are introduced during the modification to its previous version V_{i-1}; (2) all the bugs introduced in modifying current version V_i will be detected, reported, and fixed in next release V_{i+1}. In other words, we need to assume that most bugs can only live for one version and will not be carried for two or more versions. These two assumptions are unrealistic for most software products and accordingly, the number of bugs or bug reports cannot be used to represent the quality of a specific software version. This observation has been reported in our previous study[18].

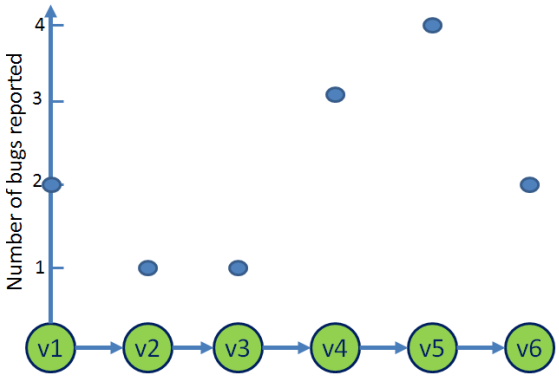


Figure 1. Number of bugs reported to each release of a software system

Next, we consider using the total number of bugs (bug reports) to represent the product quality of each release and give the following definition.

Definition 2. *For a continually evolving software product that has released n versions* $(v_1, v_2, \ldots, v_n)$*, where each new release is based on its previous release, the product quality of version* v_i $(1 \leqslant i \leqslant n)$ *can be measured with the accumulated number of bugs reported to versions* v_1 *through* v_i*.*

Based on Definition 2, the quality measures of Versions V_1 through V_6 of the product shown in Fig. 1 are 2, 3, 4, 7, 11, and 13, respectively. The basic idea under Definition 2 is that each version (say V_i) of a software product has a development history starting from the same origin, i.e., the beginning development of first version V_1. All the previous releases (V_1 through V_{i-1}) are considered preliminary releases of V_i. All the bugs reported so far are used to improve the quality of the current version V_i. In other words, according to Definition 2, when we examine the quality of one specific version, we collect all the bugs reported since the beginning and ignore all the previous releases, because they are considered premature versions of current release.

However, Definition 2 also suffers two drawbacks: (1) software quality always decreases with time as bugs will be continually detected and reported; and (2) product size or product complexity, which are important factors of software quality, are ignored in the definition; two products with the same number of bugs but different size/complexity should certainly be considered having different qualities.

To overcome these two drawbacks of Definition 2, product size and complexity factor are incorporated in this study. Conventionally, software quality has been measured with the number of faults per thousand lines of code. We adapt this idea and make the following definition.

Definition 3. *For a continually evolving software product that has released n versions ($v_1, v_2, \ldots, v_n$), where each new release is based on its previous release, the accumulated defect density of version v_i ($1 \leqslant i \leqslant n$) is the accumulated number of bugs reported to versions v_1 through v_i divided by the size or complexity of the product.*

The basic idea of Definition 3 is the incorporation of the growth of product size and complexity with the growth of number of detected bugs in order to measure software quality. In other words, Lehman's laws on software size and complexity evolution are combined with the law on quality evolution. In this paper, we will use *accumulated defect density* (ADD) to measure the quality of an evolving software product. A lower value of *accumulated defect density* (ADD) indicates a higher quality; a higher value of *accumulated defect density* (ADD) indicates a lower quality.

It is worth noting (1) in Definition 3, size and complexity of the product could be any measures, such as number of lines of code, fan-in, fan-out, number of functions, number classes, and so on; and (2) Definition 3 provides a generic measurement of software quality for continually evolving and frequently releasing software products.

4 Data Source and Data Mining Process

In this study, two open-source products are analyzed. They are Apache Tomcat and Apache Ant. The source code of these products is downloaded from their source code repositories[19]. The bug reports are mined from their Bugzilla web sites[20].

Software version system is a tree structure. There could be a trunk and zero or more branches. A trunk or a branch represents one line of evolution. In this study, it is called *an evolution line*. Four evolution lines are studied in this paper. They are Tomcat branch 5.5, Tomcat branch 6.0, Tomcat trunk, and Ant trunk, which are illustrated in Fig. 2. Because Apache Ant only has one branch of evolution (Fig. 2b), it is also a trunk. Table 1 describes the release information of these four lines of product evolution. It should be noted that only the releases with bug reports data are included in this study. Early releases without bug reports, such as Tomcat 3.0 are not included in this study.

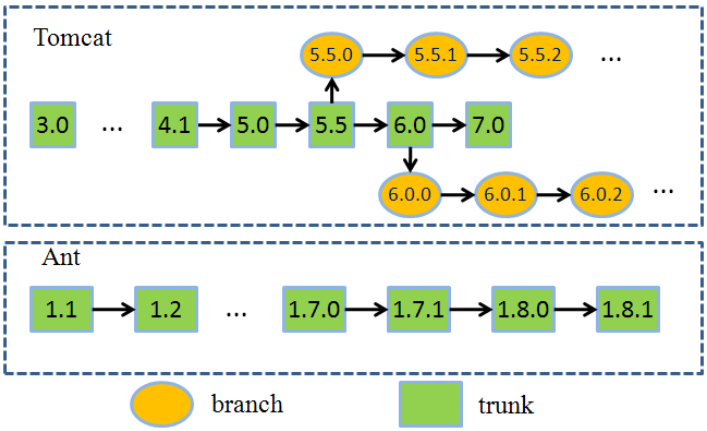


Figure 2. The four evolution lines studied in this paper

Table 1 Descriptions of the four evolution lines studied in this paper

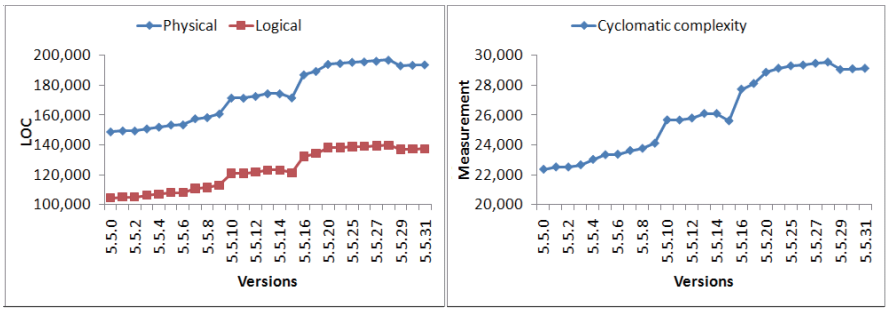
Evolution line	Number of releases	First release (date)	Latest release (date)
Tomcat branch 5.5	27	5.5.0 (8/31/2004)	5.5.31 (9/16/2010)
Tomcat branch 6.0	18	6.0.0 (10/21/2006)	6.0.29 (7/22/2010)
Tomcat trunk	9	3.1 (4/18/2000)	7.0.0 (6/29/2010)
Ant trunk	20	1.1 (7/19/2000)	1.8.1 (5/7/2010)

In measuring the size and the complexity of each product, CASE tool *LocMetric* is used[21]. Because both the two products are written in Java, only ".java" files are considered as the source code files. Three measurements of each version of four evolution lines are recorded. They are *physical lines of code*, which includes comment lines but no blank lines; *logical lines of code*, which only includes statement lines; and *McCabe Cyclomatic complexity*.

In mining bug reports, only confirmed and fixed bug reports are mined. Unconfirmed and duplicated bug reports are not included. The bugs reported to each release are collected since the release date of that version until September 29, 2010.

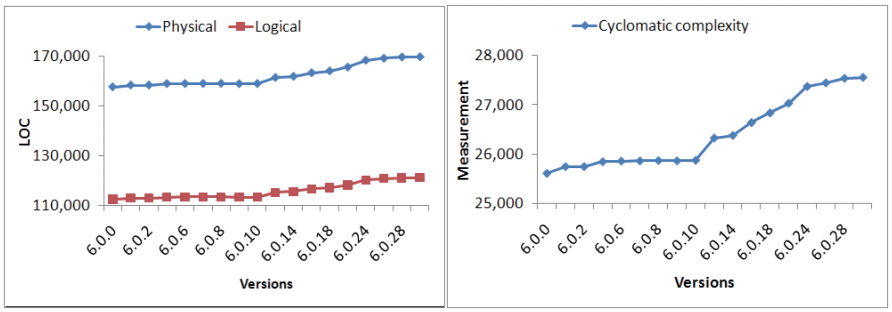
5 Analysis and Results

Figure 3 through Fig. 6 illustrate the growth of the size and complexity of Tomcat branch 5.5, Tomcat branch 6.0, Tomcat trunk, and Ant trunk, respectively. It can be seen that all these four lines of evolution obey Lehman's law Number 6 (continuing growth) and Number 2 (increasing complexity). More specifically, we can see (1) both the *physical line of code* (LOC) and the *logical line of code* (LOC) are increasing with time, which indicates that the size of the products is growing during the software evolution process; (2) the *McCabe Cyclomatic measurement* is also increasing with time, which indicates the increasing of system complexity during the software evolution process.



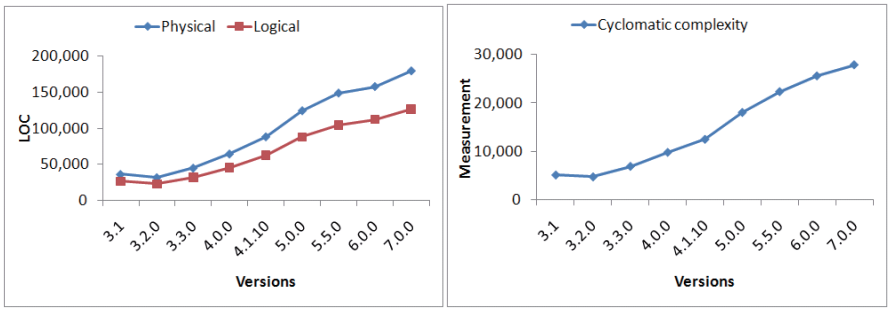
(a) (b)

Figure 3. The growth of (a) size and (b) complexity of Tomcat branch 5.5



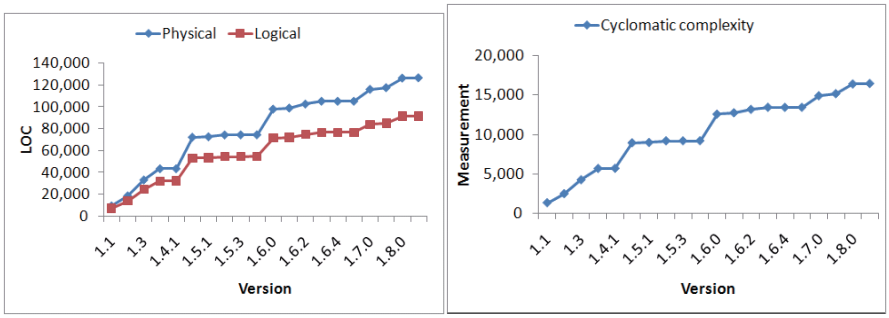
(a) (b)

Figure 4. The growth of (a) size and (b) complexity of Tomcat branch 6.0



(a) (b)

Figure 5. The growth of (a) size and (b) complexity of entire Tomcat trunk



(a) (b)

Figure 6. The growth of (a) size and (b) complexity of entire Ant trunk

The size and complexity of a product can be represented as *physical line of code*, *logical lines of code*, or *McCabe Cyclomatic Complexity*. To determine if these measurements are equally in calculating defect density, Spearman's correlation tests are performed on these three measurements. The results are listed in Table 2. It can be seen that for all 12 tests in four evolution lines, the correlation coefficients are near 1 and at the 0.001 significance level, which means *physical lines of code*, *logical lines of code*, and *McCabe Cyclomatic Complexity* have nearly perfect positive relations. Therefore, the evolution of any of these three metrics can represent the evolution of the other two metrics. In the following analysis, we will use *physical line of code* to represent product size and *McCabe Cyclomatic Measurement* to represent product complexity.

Figure 7 shows the number of bug reports mined from each version of these two branches and two trunks. It can be seen that the distribution of number of bugs are irregular in each version. It should be noted that the number of bugs reported for each version of a product does not represent the quality of that version of the product, as briefly discussed in Section 3. To recapitulate: (1) each version is based on a previous release and some of these bugs have been removed; (2) the bug reported in one version might be introduced in previous releases and has no relation with the work done in current version[18]. Therefore, we need to use accumulated bugs to study the evolution of software quality.

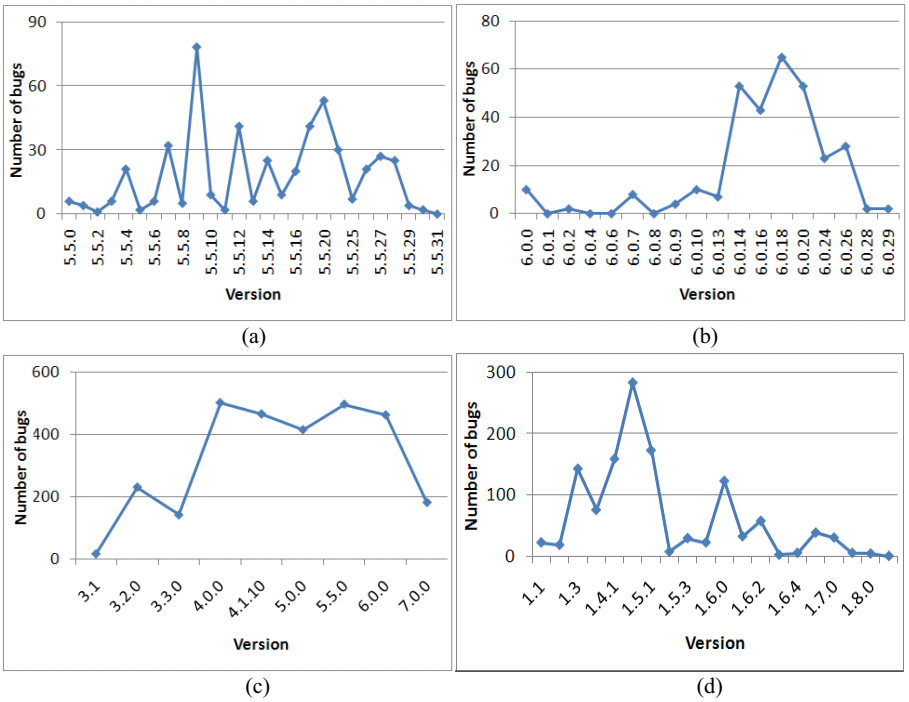


Figure 7. The number of bugs reported in each version of (a) Tomcat branch 5.5; (b) Tomcat branch 6.0; (c) Tomcat trunk; and (d) Ant trunk

Table 2 Spearman's correlation coefficients (All correlations are significant at the 0.001) level

	Tomcat 5.0	Tomcat 6.0	Tomcat trunk	Ant trunk
Physical LOC & Logical LOC	1.000	1.000	1.000	1.000
Physical LOC & Complexity	0.999	1.000	0.997	0.999
Logical LOC & Complexity	0.999	0.997	0.998	0.999

Figure 7 also shows a similar pattern of the number of bugs (bug reports) of each release: fewer bugs are reported for the first several releases; the number of bug reports gradually increases with new releases; it then decreases for the most recent releases. Based on previous study[18], this pattern is not only related to version quality. Instead, it is also related to the popularity of the product: the more users a product has, the more bugs it could be reported.

The accumulated number of bugs of a continually releasing software product will increase with no doubt. But, the slope of the increase might be different between systems and between branches of the same system. Figure 8 shows the growth of accumulated number of bugs in these two branches and two trunks. It can be seen that the growth slope of Ant trunk are lower than other three evolution lines, which indicates the quality of Ant truck might be improving. To further validate this speculation, we need to study their evolution of *accumulated defect density* (ADD), which is a generic and more accurate indicator of product quality.

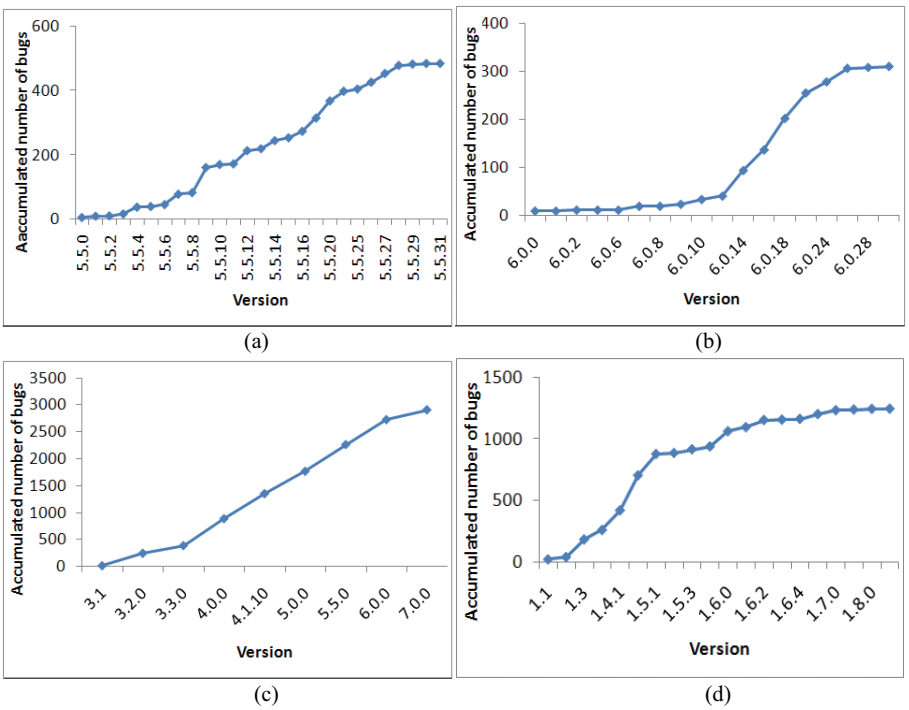


Figure 8. The accumulated number of bugs reported in each version of (a) Tomcat branch 5.5; (b) Tomcat branch 6.0; (c) Tomcat trunk; and (d) Ant trunk

Figure 9 shows the evolution of *accumulated defect density* based on product size (accumulated number of bugs divided by thousand physical lines of code) of the two branches of Tomcat and two trunks of Tomcat and Ant. Figure 10 shows the evolution of *accumulated defect density* based on product complexity (accumulated number of bugs divided by McCabe Cyclomatic Complexity) of the two branches of Tomcat and two trunks of Tomcat and Ant. In all these four lines of evolution, generally speaking, *accumulated defect density* (ADD) values increases with the release of new versions, which means, the software qualities are generally in declining trend. These observations support Lehman's Law Number 7: the quality of a software product decreases with time unless it is restructured.

In Fig. 9 and Fig. 10, we can also see some different behaviors. First, the two branches (Fig. 9a, Fig. 9b, Fig. 10a, and Fig. 10b) have higher slopes of the evolution of ADD values. Tomcat Version 5.5.0 and Version 6.6.0 are based on based on previous branches, whose defects are not included in this study, because there is no such long history data available. Second, the ADD values of the two trunks (Tomcat and Ant) are approaching stable trend, which means their qualities are approaching stabilized state and might begin to increase. For Ant trunk, we can even see some indications of declining of ADD value, which means, its quality is improving in recent releases. This behavior could be due to the system restructuring of Ant.

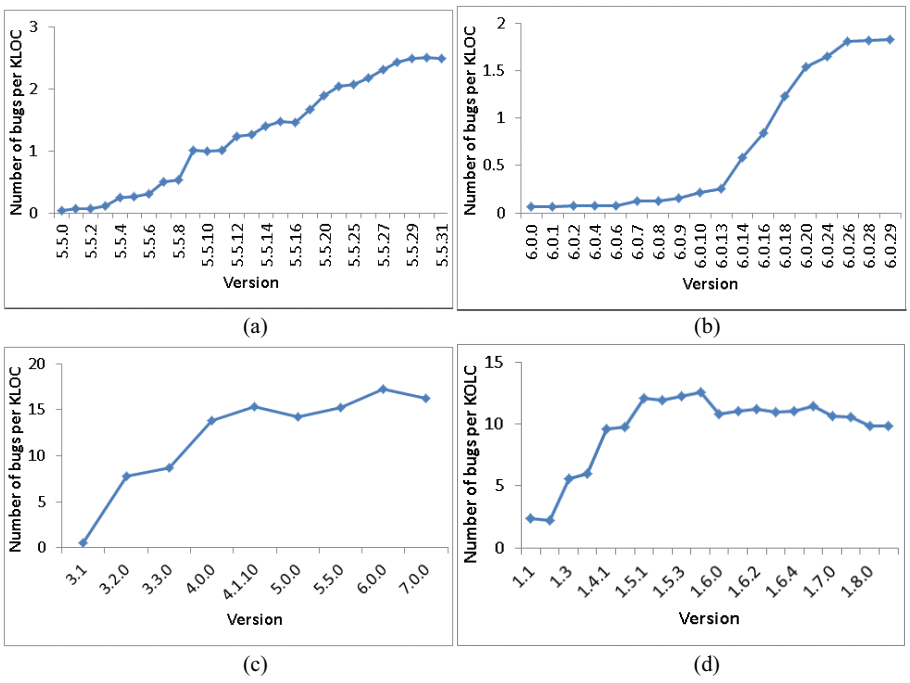


Figure 9. The evolution of accumulated defect densities based on product size of (a) Tomcat branch 5.5; (b) Tomcat branch 6.0; (c) Tomcat trunk; and (d) Ant trunk

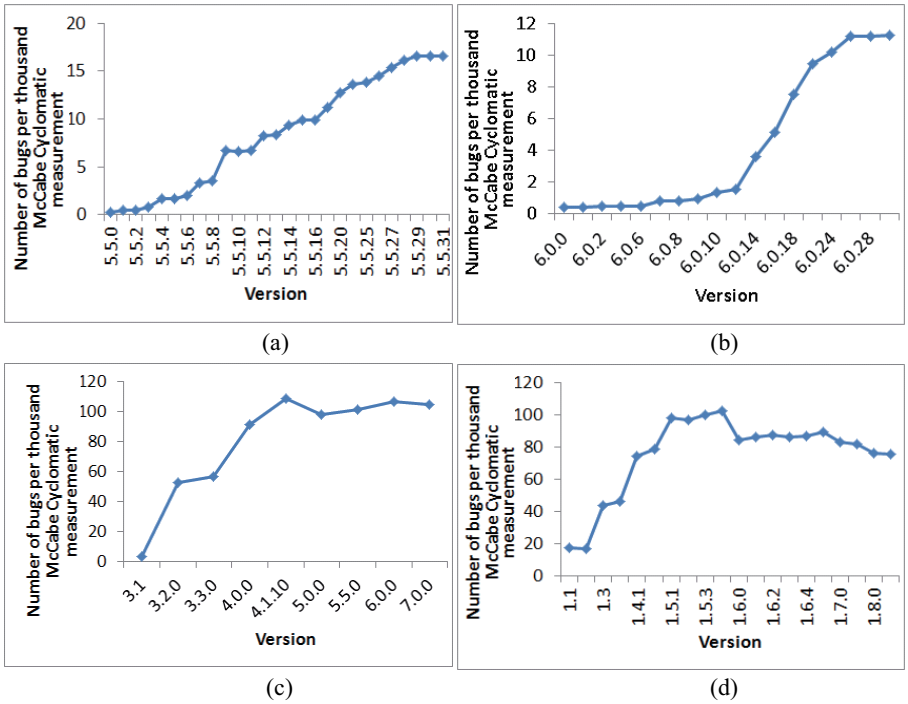


Figure 10. The evolution of accumulated defect densities based on product complexity of (a) Tomcat branch 5.5; (b) Tomcat branch 6.0; (c) Tomcat trunk; and (d) Ant trunk

Further examining Fig. 9 and Fig. 10, we can see they present similar information: (1) The quality of a continually evolving software product tends to decrease with the release of new versions; (2) As the bugs are reported and corrected, the product's quality becomes stable, which is the case of Tomcat; (3) If restructuring is performed, the product quality could even be improved, which is demonstrated by the latest releases of Ant.

Combining Fig. 7 through Fig. 10, we can also see the different behavior of quality evolution of Tomcat and Ant: the quality of Ant is better managed than Tomcat, at least for the recent releases (Version 1.51 to Version 1.80). Also, the quality evolution of three lines of Tomcat is consistent, i.e. the branches have the same behavior as trunk.

Although similar information is provided in Fig. 7 through Fig. 10, we can see that Fig. 9 and Fig. 10 are more clear than Fig. 7 and Fig. 8 to illustrate the evolution of product quality. That is the benefit of defining *accumulated defect density* (ADD).

Based on the above observations and discussions, we can state that our study supports Lehman's law on software quality evolution: the quality of evolving software products will be declining unless restructuring is performed. Specifically, we found the quality of initial releases of a product tends to decrease and the quality could be improved if restructuring is performed on later releases.

6 Conclusions

In this paper, we defined a generic software quality metric called accumulated defect density for continually evolving software systems. Using this metric, we validated Lehman's law of software quality evolution. In particular, we mined the bug history of two open-source systems, Apache Tomcat and Apache Ant and studied the growth of size, complexity, and quality of two trunks and two branches of these two software systems. Our results support Lehman's laws of software evolution, especially, Law Number 7—declining quality.

Because quality is such an important issue in software development, measuring software quality is thereby an important task in any software project. We hope this study can provide software engineers with a generic metric to measuring the quality and monitor the evolution of continually evolving and frequently releasing products.

Acknowledgement

This study is supported in part by the Faculty Research Grant of Indiana University South Bend.

References

[1] Lehman MM. Programs, life cycles, and laws of software evolution. Proc. of IEEE, 1980, 68(9): 1060–1076.

[2] Lehman MM. On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software, 1980, 1(3): 213–221.

[3] Lehman MM. Laws of software evolution revisited. Proceedings of the 5th European Workshop on Software Process Technology. Springer Verlag. 1996. 108–124.

[4] Lehman MM, Perry DE, Ramil JF. Implications of evolution metrics on software maintenance. Proc. of the 14^{th} International Conference on Software Maintenance. November 1998. 208–217.

[5] Lehman MM, Perry DE, Ramil JF. On evidence supporting the FEAST hypothesis and the laws of software evolution. Proceedings of the 5th International Software Metrics Symposium, November 1998. 84–88.

[6] Lehman MM, Ramil JF. The impact of feedback in the global software process. Journal of Systems and Software, 1999, 46 (2–3): 123–134.

[7] Lehman MM, Ramil JF, Wernick PD, Perry DE, Turski WM. Metrics and laws of software evolution – the nineties view. Proceedings of the 4th International Software Metrics Symposium. November 1997. 20–32.

[8] Israeli A, Feitelson DG. The Linux kernel as a case study in software evolution. Journal of Systems and Software, 2010, 83(3): 485–501.

[9] Barry EJ, Kemerer CF, Slaughter SA. How software process automation affects software evolution: a longitudinal empirical analysis. Journal of Software Maintenance and Evolution: Research and Practice, 2007, 19(1): 1–31.

[10] Herraiz I, Robles G, Gonzalez-Barahon JM. Comparison between SLOCs and number of files as size metrics for software evolution analysis. Proc. of the Conference on Software Maintenance and Reengineering. 2006. 206–213.

[11] Godfrey MW, Tu Q. Evolution in Open Source software: A case study. Proc. of the International Conference on Software Maintenance. October 2000. 131–142.

[12] Godfrey M, Tu Q. Growth, evolution, and structural change in open source software. Proc. of the 4th International Workshop on Principles of Software Evolution. September 2001. 103–106.

[13] Robles G, Amor JJ, Gonzalez-Barahona JM, Herraiz I. Evolution and growth in large libre software projects. Proc. of the 8th International Workshop on Principles of Software Evolution. September 2005. 165–174.

[14] Succi G, Paulson J, Eberlein A. Preliminary results from an empirical study on the growth of open source and commercial software products. Proc. of the 3rd International Workshop on

Economics-Driven Software Engineering Research. May 2001.
[15] Simmons MM, Vercellone-Smith P, Laplante PA. Understanding open source software through software archaeology: the case of Nethack. Proc. of the 30th Annual IEEE/NASA Software Engineering Workshop. April 2006. 47–58.
[16] Eick SG, Graves TL, Karr AF, Marron JS, Mockus A. Does code decay? assessing the evidence from change management data. IEEE Transactions on Software Engineering, 2001, 27(1): 1–12.
[17] Lee Y, Yang J, Chang KH. Metrics and evolution in open source software. Proc. of the 7th International Conference on Quality Software. October 2007. 191–197.
[18] Yu L, Ramaswamy S, Nair A. Using bug reports as a software quality measure. Proc. of the 16th International Conference on Information Quality. November 2011. 277–286.
[19] Apache Archive. http://archive.apache.org/dist.
[20] ASF Bugzilla. https://issues.apache.org/bugzilla/.
[21] LocMetrics. http://www.locmetrics.com/.