Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

Measuring the Evolution of Open Source Software in E-Learning Systems

Ajlan Al-Ajlan
Information System Management, Qassim University, Buraydah, Saudi Arabia

Abstract. Measuring the evolution of any system, whether commercial or non-commercial, is important if the advantages and disadvantages of systems are to be determined. For that reason, the most important challenges are the continuously changing environment in which FOSS operates and its relationship with commercial software. This paper therefore measured the evolution of FOSS in relation to Moodle and Magento software and examined and monitored this using metric technology. It has made a compression between the rate of change of four metrics between Moodle and Magento systems. In addition, Lehman's laws were used to observe and support the scale of the evolution. Moreover, Project Code Metrics were used to measure nine versions of Moodle and four versions of Magento as a case study.

Keywords: E-Learning, Free Open Source Software, Moodle and Magento, Project Code Metrics, Lehman's laws.

1 Introduction

E-Learning was a preliminary form of online learning, used especially in higher education. Since the invention of the computer in the middle of the last century, technology has rapidly developed in all fields and must now be taken into consideration whenever possible. Computer software can be divided into two types of programs. The first type consists of commercial programs, which are owned by companies or individuals, and the second type is Free Open Source Software (FOSS). Commercial software is encrypted and not open to users, but is instead mainly owned by an individual or a company. FOSS is unencrypted and open source, which means that users are free to use, download, modify and even distribute it under the terms of the GNU license [1, 2].

Very large software packages need to continuously develop if they are to keep pace with the rapid advances in technology and to avoid losing out to market competitors. However, the preservation of these large program packages is difficult, very complicated, and time-consuming. This complexity is required to add new tools and features, and for the repair and maintenance of all the complex software, which takes a lot of time and effort and is expensive. There are two patterns in the evolution and growth of OSS: the maintenance of the software codes and the development of the requirements of the application. The evolution of FOSS can be evaluated using tools such as SuiteCX and quantitative metrics [2, 3].

It is clear that the protection and security of FOSS is very important; this has received the most attention from developers and researchers. Even with rising anxieties related to this issue, relatively few studies of FOSS have been published. There are currently many more in-

depth studies that examine how the structural design of software supports FOSS [1, 4]. Therefore, this study aims to confirm that FOSS is one of the most important aspects of the evolution of software development, because it is open source and therefore easier to access the code without restrictions or costs. Moreover, it presents the most important tools that may be used to develop FOSS.

The main aim of this study was to measure and demonstrate the evolution of FOSS. Project Code Metrics (PCM) were used to measure the evolution of nine versions of the e-learning system Moodle and four versions of the e-commerce system Magento. The focus of this study was on four areas in the Moodle and Magento systems. The differences between the nine versions of Moodle released over a period of seven years were examined. In addition, this study examined the evolution of Moodle by applying the eight Lehman laws of software evolution.

The rest of this study is organized as follows: a literature review of FOSS and the evolution of software and e-Learning systems are described in Section 2. Section 3 describes the evolution of FOSS by using Project Code Metrics with Moodle software as a case study. Section 4 presents a discussion of this work. Finally, a conclusion and suggestions for future work are presented in Section 5.

2 Literature Review

2.1 Free Open Source Software

FOSS has begun to be extensively adopted by commercial, public and academic organizations. Developers use open source codes as a language to create and develop software. These codes are free and not closed, meaning they are available to all programmers. The ease of access to the code and ability to download it for free and without restrictions has led to a revolution in the use of open source programming languages to develop software. FOSS can be used in operating systems such as Linux, email software, Internet servers such as Apache, Java's Guice, and e-Learning systems such as Moodle [5, 2].

In 1970, FOSS was first developed in multiple countries. Richard Stallman, an American software developer, was the first person to develop and suggest ideas for the development of a free version of Unix software. The GNU operating system was released under the newly created General Public License. Assurances were made that the source code must remain openly accessible to all users [1, 5, 6].

In 2000, the Organizations and Overview project was established as a FOSS project. It was released in 2001 and the first version was released in 2000. The development started within StarDivision, a German-based company acquired by Sun Microsystems in 1999. Before establishing the Organizations and Overview project, the code was closed source. The website OpenSource.org provides more information about the development, collabouration and use of OS software [7].

There are already more than 300 types of commercial e-Learning software, of which more than 70 are free e-Learning systems that use FOSS [8]. Some of this software, such as Moodle platforms is superior to commercial e-Learning software. This study focuses on Moodle and shows some of its versions in Table 1 [9, 10, 11].

Table 1. Summary of Moodle Versions Statistics [30]

No.	FOSS	No.	FOSS	No.	FOSS
1.	Moodle	2.	Bscw	3.	WebCT
4.	Sakai	5.	ProProfs	6.	LON-CAPA
7.	Ilias	8.	Udacity	9.	Spaghetti Learning
10.	Eduplone	11.	Siminars	12.	MamboLaiThai
13.	Claroline	14.	Udemy	15.	SkillShare
16.	Drupal	17.	.LRN	18.	OpenACS

The Reasons for Working with FOSS. The reasons for working with FOSS include the increased adoption of open source software. It is now considered equivalent to many proprietary software alternatives [12]. The reasons for the success of FOSS may be summarized into the following four main areas [1, 13, 14, 15]:

1) *Cost*: Most OSS is currently free of charge, and people are free to use, modify, add to and even distribute it under the GNU General Public License. Developers and researchers are not charged for using OSS, and can download, modify and even distribute it under the GNU General Public License.
2) *Auditability:* An important reason is the auditing process. FOSS publishes its source code, which supports users in terms of auditability. In contrast, commercial closed source software requires users to trust the seller, especially in terms of specifications, such as freedom, security and adherence to standards, and flexibility in the face of future changes. If the source code is not available, these claims remain simply claims. Publishing the source code makes it possible for users to have confidence that there is a basis for these claims.
3) *Openness:* Software is unlocked for any users wishing to work with FOSS. Furthermore, uses are free to use, download, modify and even distribute it under the terms of the GNU license.
4) *Flexibility and Freedom:* Open source software enables a number of projects to be carried out and for a large number of researchers from different backgrounds and countries to work together. This gives researchers more flexibility and the freedom to understand all the requirements, and speeds up the implementation of programs.
5) *Speed*: Speed is a significant aspect of all technologies, especially OSS. For this reason, when developers wish to build software quickly, they typically take on a large number of assignments and then test them as prototype software. As this is quicker than using

proprietary models, they can respond to mistakes and errors and find solutions quickly, because they have the source code.

6) *Quality:* FOSS was created by thousands of developers and users working to improve and innovate new features and enhancements for the software, especially security.

2.2 Free Open Source Software Evolution

Controlling FOSSE is now the most important challenge for developers. The main challenge with FOSS is therefore how to make progress with its setting, particularly the development of the quality and security of the software. There are two main aspects to the evolution of FOSS, namely how to develop the features and tools of the software, and the maintenance of and improvements to the code [16].

Software Evolution. As the majority of firms have become more dependent on software, the useful management of software evolution has become critical to a firm's success. Therefore, the planning and development of software evolution, particularly for FOSS, has become vital.

An experiential study by Meir Lehman within IBM in 1969 aimed to improve the company's programming effectiveness. It received little attention within the company and had no impact on its development practices. The aim of Lehman's study was to formulate a scientific theory of software evolution. Some variants were found, which were first described in Lehman 1974 as the laws of software evolution. In 1996, the last version of the laws was published after several years of intense activity and refinement [17].

The eight laws of software evolution proposed by Lehman have developed into a theory for the software evolution engineering labouratory. Officially, this labouratory considers the eight Lehman laws as rules to understand software evolution and proposes the best solutions for problems. This study uses Lehman's laws to deal with the specific evolution problems of FOSS and to suggest some solutions to these problems.

[15] studied the evolution of OSS and assessed Lehman's laws to observe if it they were appropriate to OSS evolution. Linux Kernel was analysed and found to have a super-linear growth rate related to its size. Moreover, the same result was found in Vim Text Editor [14, 18]. More studies published by Lehman analysed five types of software: ICL VME Kernel, IBM OS 360, Logica FW, and two large real-time telecommunications systems. Other studies have also been carried out, establishing steady growth models, but their results have not yet been published [19].

The laws of software evolution as summarized by Lehman [20] are as follows:

Rule 1: Continuing Change: An e-type system must be continually modified to gradually become acceptable, or it will become progressively less satisfactory to use.

Rule 2: Increasing Complexity: software development increases complexity and maintenance.

Rule 3: Self-Regulation: software evolution processes are self-regulating, namely controlled by either users o r the software itself.
Rule 4: Conservation of Organizational Stability: the average effective activity rate in software does not vary throughout the system's lifetime.
Rule 5: Conservation of Familiarity: The content of successive issues is statistically invariant throughout the active lifetime of the software.
Rule 6: Continuing Growth: the growth of software code should continually increase to preserve user satisfaction throughout their lifetimes.
Rule 7: Declining Quality: The quality of software will seem to decline, unless if it is maintained and adapted to operational environmental modification.
Rule 8: Feedback System: Software evolution processes constitute multi-loop, multi-level, multi-agent feedback systems and must be treated as such if they are be successfully improved or modified.

The Differences between Open Source Evolution and Traditional Evolution.

The growing significance of OSS has helped developers to analyse how traditional software engineering differs from OSS. The important question is whether the environment of OSS is fundamentally different from that of commercially and traditional available software. Lehman and other researchers carried out a series of experiential studies that showed that traditional software grows at a linear or sub-linear rate [20, 21, 22].

Previous studies into Linux software appear to conclude that OSS builds up in a unique approach. [23] discovered some features in Linux that are increasing at a super-linear rather than a sub-linear rate. On the other hand, more studies into OSS are essential before drawing conclusions. [24, 21] analysed the evolution of Linux and FreeBSD, and found that both systems have a linear upper bound, and are consequently similar to the rates of increase for profit systems. This study did not aim to confirm the hypothesis that OSS grows at rates that exceed those of traditional systems.

[23] introduced 22 studies, of which 46% explained activities belong to the requirements process and 60% explained activities belonging to the design process. Nearly all accounted for activities linked to execution. The OSS community does not enact software engineering models. Therefore, the requirements of OSS are developing using a number of different web artefacts, as well as repeated interactions in forums and through messaging. A software system is a general feature of OSS implementation and is modular in design. The main concern in the OSS community is implementation, and any developers can make contributions, including code and designs.

2.3 E-Learning Systems

E-learning is increasingly regarded as a significant feature and tools in higher education. The advantage of e-learning is that it offers a chance for students to connect with each other and

their teachers electronically through discussion board forums, chatting and e-mail. Moreover, it leads to a culture of learning and self-training at university, which develops and improves the capacity of students to learn at a low cost and with minimum effort [25].

In e-learning, genuine benefits are to be gained from the use of the technology; it has therefore become very popular and is embedded in many institutions. The use of e-learning technology should increase the number of students in higher education. It promotes good communication and opportunities for automated assessment. In terms of widening participation, e-learning can offer resources for part-time students who cannot always travel to the institute [26, 27].

Users of e-learning can enjoy the privacy of their home environment. Internet technology supplies easy access to important information at a low price. E-learning makes interaction between instructors and students easy; it is almost free from time and location constraints. Furthermore, it facilitates the benefits of the integration of group learning facilities and individuals. It also enhances learning and teaching experiences by supporting learning at flexible times and locations. E-learning allows the online delivery of resources for both lecturers and students, with many possibilities for students to access educational resources both on and off campus [9, 28].

Why Choose the Moodle Platform? [8] argues that the Moodle platform is the best in terms of security, performance, support, interoperability, flexibility, communications and metrics for course delivery tools. In addition, [9] showed Moodle to be the best platform. This was a comparative study between 10 VLE systems and Moodle that compared the features and capabilities of VLE tools in the first phase, and the technical aspects of VLE systems in the second. Moreover, [10] reports that the result of the evaluation shows that Moodle had the best rating in the adaptation category and in terms of adaptation issues.

[11] provides a comparison between four VLE systems based on categories. This study showed that Moodle outperformed all other systems and scored 4.467 out of 5. Finally, Moodle is free and open source, which enables developers and researchers to use it and even to distribute it under the terms of the GNU General Public License.

3 The Evolution of FOSS Using Project Code Metrics with Moodle as a Case Study

This section focuses on an analysis of FOSS using Project Code Metrics (PCM) with Moodle Software as a case study. The study examined the differences between the nine versions of Moodle, and also examined the following four areas in the Moodle code: 1) Statistical Labour Distribution; 2) Quality Measurements; 3) Project Code Meter Time; and 4) Quantitative Metrics.

PCM is a professional software tool used to measure the complexity, quality and maintainability of software projects as well as estimating the time and cost of software

projects. It enables users to be continuously aware of the health of their source code. In addition, it exports metrics to HTML format for public display and to CSV for additional study [30]. PCM has some standard metrics, some of which are used in this study to measure the evolution of Moodle, as shown in Appendix A, B, C and D below.

This study used Moodle software as a case study and examined nine versions of it. To provide an outline summary of this study, there is a huge difference between Version 1.6 and Version 3.2, which are separated by a seven-year gap. Version 1.6 has just 2,180 files, compared to 11,106 in version 3.2. Version 1.6 has just 183,563 lines of code, compared to 1161993 in version 3.2. Version 2.6 has the maximum number of lines of code (1504693), as shown in Appendix A. This shows that there has been huge evolution in Moodle software in terms of the FOSS it uses.

Table 2 shows all the versions of Moodle examined in this study. These nine versions are 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0 and 3.2. These versions were released between 28 January 2009 and 23 May 2016, with the last version under preparation during this study.

Table 2. Summary of Moodle Versions Statistics [30]

No.	Version	Year
1.	Moodle -1.6.9	28 January 2009
2.	Moodle -1.8.14	3 December 2010
3.	Moodle -2.0.10	9 July 2012
4.	Moodle -2.2.11	8 July 2013
5.	Moodle -2.4.11	14 July 2014
6.	Moodle-2.6.11	11 May 2015
7.	Moodle -2.8.12	9 May 2016
8.	Moodle -3.0.3	14 March 2016
9.	Moodle -3.2	10 November 2016

3.1 Statistical Labour Distribution

Statistical Labour Distribution (SLD) in all versions of Moodle is described in Appendix A and Figure 1. SLD indicates the number of hours for the 11 standard metrics. This method allows programming time to be measured for current software projects, according to the Weighted Micro Function Points Algorithm (WMFPA). This is useful in weighing up the effort (work hours) of a developer or team, either in-house or outsourced. SLD indicates which of the five standard metrics will appear in both the minutes and in the percentage of total file development time, as shown below [29]:

1. *Time:* display the calculated of programmer time required for developing, coding, testing and debugging the software.
2. *Coding:* display the calculated programmer time involved in developing only the coding in the file program.

ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145
ORIGINAL ARTICLE

3. *Debugging:* display the calculated programmer time involved only in developing debugging in the file program.
4. *Testing:* display the calculated programmer time developed only in the testing of the file program.
5. *Object Vocabulary, Flow Complexity, Data Transfer, Inline Data, Object Conjuration, Code Structure, Arithmetic, Comments*: indicate the linking of the WMFPA source code metric measured for the file program.

In the SLD, Version 2.6 appears to require the highest number of hours in Debugging (139724), Arithmetic Intricacy (12,026), Data Transfer (52,263), Code Structure (36,852) and Inline Data (11,251), whereas version 3.0 is preferred in terms of Coding (175662), Testing (89,499), Flow Complexity (144614) and Comments (3,937). In addition, Version 3.2 appears to require the highest number of hours in Object Conjuration (75,117) and Object Vocabulary (88,565), as shown in Appendix A. The best evolution was in 2.6, which showed growth in five standards, followed by 3.0, which showed growth in four standards, and 3.2, with growth in two standards.

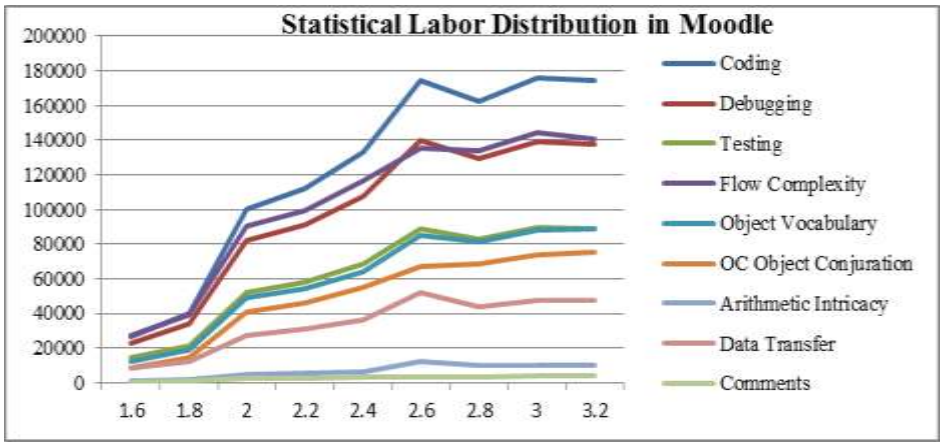

Figure 1. The evolution of SLD in nine versions of Moodle by time in hours.

Figure 1 clearly displays the evolution of SLD in nine versions of Moodle over the past decade. The coding standard had the highest evolution, and Version 3.0 the highest number of hours (175662), followed by Version 3.2, with 174677 hours. The second standard is Flow Complexity, for which Version 3.0 had the highest number of hours (144614), followed by Version 3.2 with 140428 hours. The debugging standard had the highest number of hours in Version 3.0 version (89,499), followed by Version 3.2 (89,499 hours). In contrast, the Arithmetic Intricacy standard had the lowest evolution; the highest number of hours in this standard was in Version 2.6 (11,251), followed Version 3.0 with 7,128.

As shown in Figure 1, the Coding, Flow Complexity and Debugging standards underwent the most evolution in SLD in the nine versions of Moodle over the past decade. In contrast, the Testing, Object Vocabulary, Object Conjuration and Data Transfer standards had low evolution, and Code Structure and Arithmetic Intricacy had weak evolution.

ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145
ORIGINAL ARTICLE

3.2 Quality Measurements

Quality Measurements (QLMs) in all versions of Moodle are described in Appendix B and Figure 2. These refer to some essential source code qualities that affect maintainability, re-use and peer review. QLMs refer to the eight standard metrics described below [29]:

1. *Code Quality Notes Count:* this displays the number of warnings indicating quality issues. This should preferably be zero; higher values suggest that the code will not be easy to maintain.
2. *Code to Comment Ratio and Essential Comment Factor:* this indicates the balance between code statements and comment lines. If the value is 100, this means each code has a comment, whereas if the value is lower than 100, this means that only some of the code lines have comments. If the value is higher than 100, this means that each code line has more than one comment.
3. *Code Structure Modularity:* this shows the degree of the code, which is divided into functions and classes. If the values are around 100, the balance is good. However, if the values are lower than 100, they indicate a low code, and if the values are higher than 100, they indicate fragmented code.
4. *Logic Density:* this shows how the logic is condensed within the program code. If the values are higher, this means that code is more likely to have been generated by a person. If there is a low value, this indicates that the code is not guaranteed.
5. *Source Divergence Entropy:* this displays the degree to which objects are manipulated by logic. The higher the values, the greater the amount of manipulation.
6. Information Diversity Factor: this displays the degree of re-use of objects. Higher values indicate more re-use.
7. *Object Convolution Factor:* this displays the degree to which objects work together with each other. If the values are higher, there is more interaction and thus more complex data flow.

Appendix B shows that Version 2.6 appears to have scored highest in terms of Code Quality Notes Count (4275) and Code Structure Modularity (188), but Version 3.2 is preferred for the Object Convolution Factor (29) and Code to Comment Ratio (25). Version 1.6 appears to have the highest number for Logic Density (104) and Source Divergence Entropy. Version 2.8 appears to have the highest number in Essential Comment Factor and in Code to Comment Ratio (25). Version 2.4 appears to have the high number in Information Diversity Factor (548). Version 3.0 appears to have the highest number for the Code to Comment Ratio (25).

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

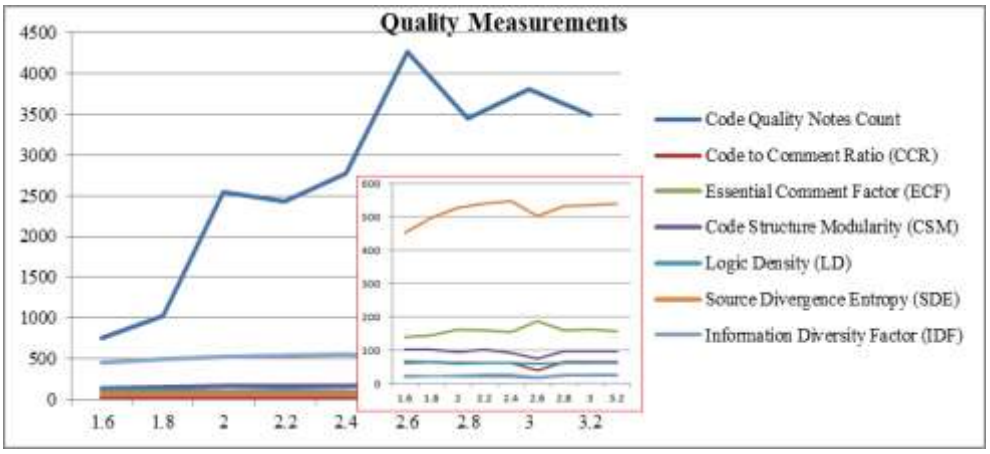

Figure 2. The evolution of nine versions of Moodle in by time in hours.

Figure 2 displays the evolution of the nine versions of Moodle in QLMs over the past decade. The Code Quality Notes Count standard had the highest evolution, whilst Version 2.6 had the highest number of notes (4,275), followed by 3.0 with 3,814 notes. The second standard is Information Diversity Factor, for which Version 2.4 version had the highest number of files (548), followed by Version 3.2 with 540 files and Version 2.2 with 540 files. The Information Diversity Factor, Code to Comment Ratio and Code Structure Modularity show low evolution at this stage. In contrast, the Logic Density and Source Divergence Entropy standards have not evolved.

3.3 Quantitative Metrics

Quantitative Metrics (QTMs) are the conventional metrics used in the Legacy Sizing Algorithms (LSA) approach and are specific in order to obtain general data. They are given for each file and each entire project based on the context. There are seven standard metrics for, as listed below [29]:

1. *Files:* this indicates the number of files that metrics measure only on a per-project basis.
2. *Logical Lines of Code:* the stated number of lines of code.
3. *Multi Line Comments:* the number of comments that cross more than one text line.
4. Single Line Comments: the number of comments with a width of just a single line of text.
5. *High Quality Comments:* these indicate the number of comments that look verbally adjectival irrespective of their length in lines of text.
6. *Strings:* the number of text strings set in source code.
7. Numeric Constants: the number of hard-coded numbers embedded in the code.

In the QTMs, Version 2.6 appears to have the highest number of Logical Lines of Code (1504693), Strings (1351206) and Numeric Constants (675927), whereas Version 3.0 is preferable in terms of High Quality Comments (296103) and Multi Line Comments

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

(104758). Moreover, Version 3.2 has the highest number of Single Line Comments (224271) and Files (11,106), as shown in Appendix C.

Figure 3 illustrates the evolution over the nine versions with seven standards of QTMs. This comparison shows that there is a big difference between the first (1.6) and the last (3.2) versions, which shows that there have been huge evolution in Moodle software. Version 2.6 version shows the most evolution in most of the standards, as shown in Figure 4.

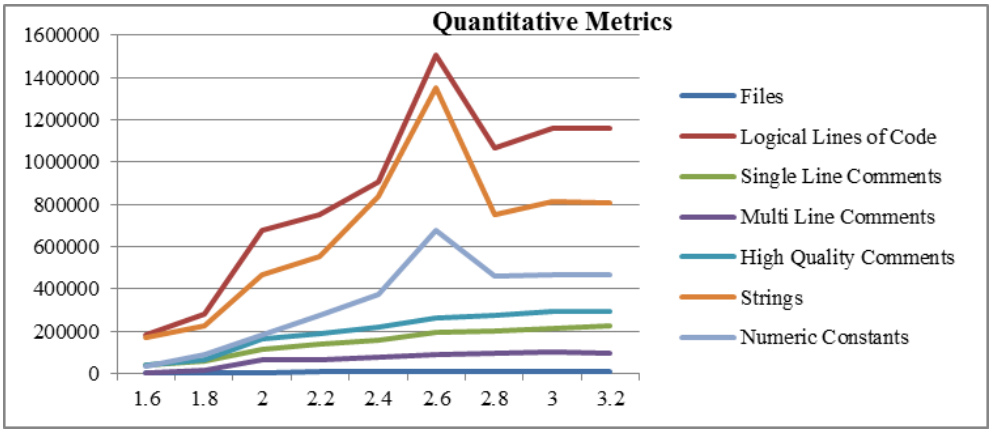

Figure 3. The evolution of nine versions of Moodle in QTMs.

3.4 Project Code Meter Time

For Project Code Meter Time (PCMT), Version 3.0 appears to have the highest number of hours in Total Time in all versions (24272044 hours). This is because it has 10539736 hours in the Coding standard, which is the highest number of hours in all version of Moodle. In addition, it has a high number in Testing (5369991), Flow Complexity (8676890) and Comments (236255), as shown in Appendix D.

Version 2.6, the second version, has 24180581 hours in all versions. This is because it has a high number in Debugging (8383478 hours). It also has the highest number of hours in Arithmetic Intricacy (721588), Data Transfer (3135816), Code Structure (2211166) and Inline Data (675097), as shown in Appendix D.

Version 3.2 is the third version, and has 24076766 hours in all versions. This is because it has a high number in Object Conjuration (10480668 hours) and 5313902 hours in Object Vocabulary, as shown in Appendix D.

Version 1.6, the last version, has 3820200 hours in Total Time in all versions. This is because it has the lowest number of hours in all standards, as shown in Appendix D.

Figure 4 shows the number of hours in PCMT in the nine versions of Moodle over the past decade. The coding standard has the highest number of hours, and Version 3.0 has the highest number of hours (10539736), followed by 3.2 with 10480668 hours. The second standard is Debugging, for which Version 2.6 has the highest number of files (8383478 hours), followed by Version 3.0 with 8362317 hours.

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

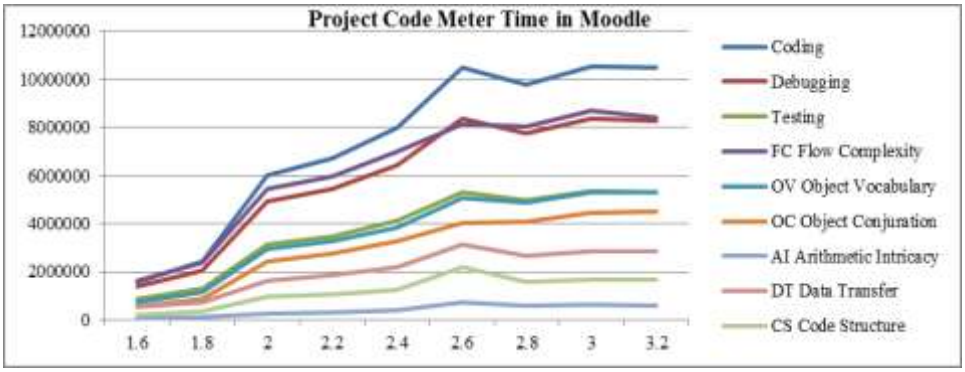

Figure 4. The evolution of PCMT in nine versions of Moodle.

4 Discussion and Results

In Section 3 above, PCM was measured as a case study in the nine versions of Moodle. The study focused on the nine versions, and discovered differences between them through the four selected areas. These four areas are described in detail in Section 3 above. Version 2.6 had the highest evolution in this study because this was the first version in the new style of Moodle. The history of Moodle software consists of two stages. The first stage included Versions 1.6 to 2.4, and the second stage includes Version 2.6 onwards. There is a big gap between Versions 2.4 and 2.6 because Moodle started to use the new style.

This discussion summarises the results of this study by dividing the results into the following four areas: 1) Statistical Labour Distribution; 2) Quality Measurements; 3) Project Code Meter Time; and 4) Quantitative Metrics.

4.1 Statistical Labour Distribution and Project Code Meter Time

Figure 5, Appendix A and Appendix D show the SLD and PCMT areas in all nine versions of Moodle. The rate of change in the SLD and PCMT areas was 530.27% and 562.85% respectively, which demonstrates that there has been a high degree of evolution in these areas. The most evolution in SLD was in Comments, which increased by 949.04%, and the best evolution in PCMT was in Code Structure, which increased by 946.54% between Version 1.6 and Version 3.2. Version 3.0 had the highest number of hours in SLD and PCMT (809063 and 24272044 respectively).

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

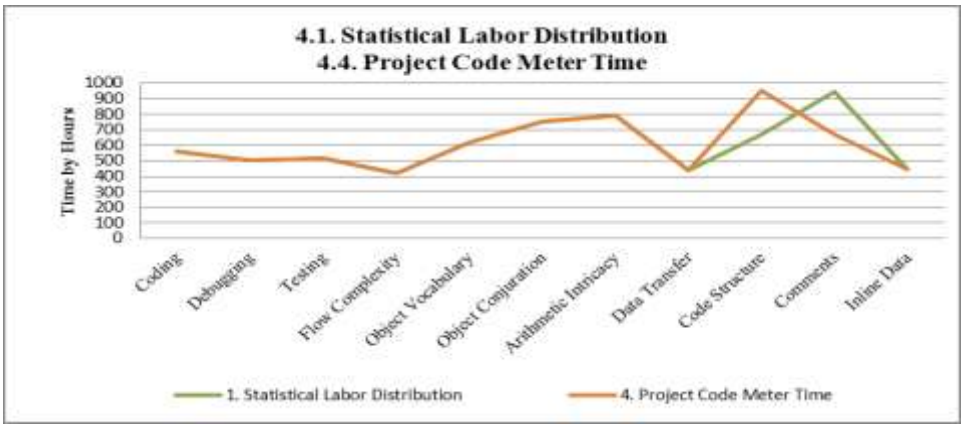

Figure 5. Percentage change between V1.6 and V3.2.

4.2 Quality Measurements

Figure 6 and Appendix B show the QLM area in all versions of Moodle. The change rate in QLMs was 175.28%, which shows that there was an evolution in this area despite the increase in evolution in some standards, as shown in Appendix C. The highest amount of evolution was in the Code Quality Notes Count, which increased by 364.67% between Versions 1.6 and 3.2. Version 2.6 had the highest number (5,179), as shown in Appendix C.

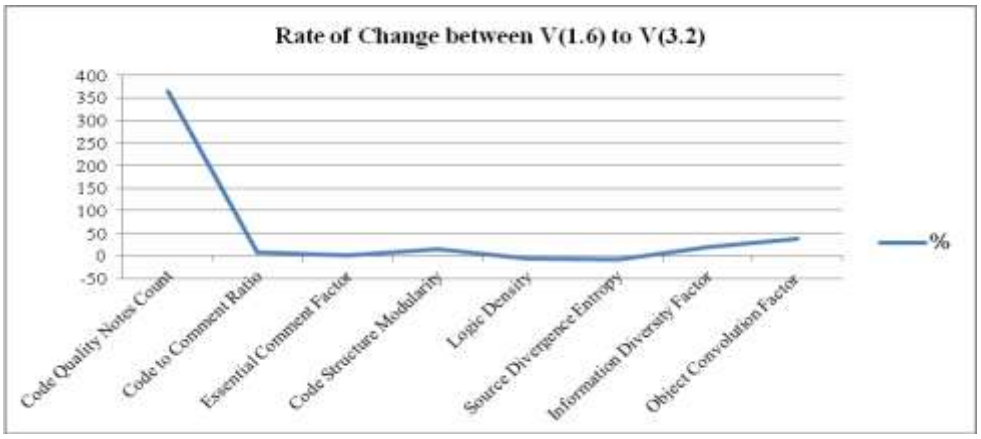

Figure 6. Percentage change between V(1.6) and V(3.2)

4.3 Quantitative Metrics

Figure 7 and Appendix C show QTMs in all versions of Moodle. The rate of change in QLMs was 534.52%, which shows that there was substantial evolution in this area. The highest amount of evolution was in Multi Line Comments, which increased by 1815.11% between Versions 1.6 and 3.2. Version 2.6 had the highest number (4091086), as shown in Appendix C.

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

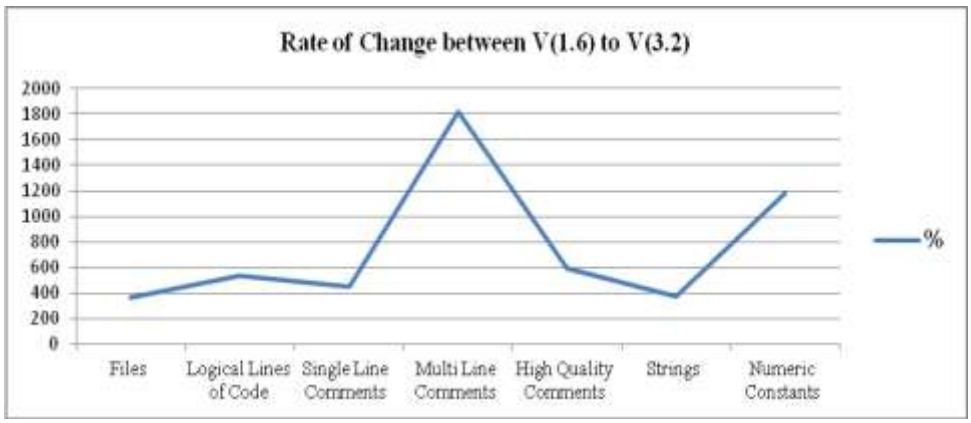

Figure 7. The Percentage change between V(1.6) and V(3.2).

4.4 The Rate of Change of Four Metrics between Moodle and Magento System

This discussion focuses on two types of comparison in terms of the rate of change. The first comparison is between Versions 1.6 to 3.2 in Moodle system and the second is between Versions 1.1 to 1.9 in Magento system. Magento is an e-Commerce platform that has open source developed by Varien Inc. This platform has well-organized business user tools speed up build up time and improves productivity [31].

The Rate of Change between V1.6 to V3.2 in Moodle System. Data analysis from Table 3 shows that the rate of change between V1.6 to V3.2 was high in the SLD, QTMs and PCMT areas, compared to QLMs being quite good in some standards, with negative changes in others. The high rate in SLD area was in the Comments standard (949.03%), whereas the high rate in QTMs is for Multi Line Comments (1815.11%). Moreover, the highest rate in the PCMT area was for Comments (946.54%), whereas the highest rate in QLMs was for Code Quality Notes Count (364.67%). On the contrary, the lowest rate in SLD area was for the Flow Complexity standard (417.66%), whereas the lowest rate in QTMs was for Files (363.57%). Moreover, the lowest rate in PCMT area was for Flow Complexity (417.65%), whilst the lowest rate in QLMs was for Source Divergence Entropy (-8.82%).

Table 3. Rate of Change between V(1.6) to V(3.2) of Moodle System

1. Statistical Labour Distribution		2. Quality Measurements		3. Quantitative Metrics		4. Project Code Meter Time	
Package	%	Package	%	Package	%	Package	%
Coding	562.86	Code Quality Notes Count	364.67	Files	363.57	Coding	562.85
Debugging	502.03	Code to Comment Ratio	8.69	Logical Lines of Code	533.02	Debugging	502.02
Testing	515.49	Essential Comment Factor	1.56	Single Line Comments	447.61	Testing	515.48
Flow Complexity	417.66	Code Structure Modularity	13.57	Multi Line Comments	1815.11	Flow Complexity	417.65
Object Vocabulary	616.77	Logic Density	-6.73	High Quality Comments	591.83	Object Vocabulary	616.76
Object Conjuration	754.47	Source Divergence Entropy	-8.82	Strings	369.53	Object Conjuration	754.39
Arithmetic Intricacy	792.68	Information Diversity Factor	19.20	Numeric Constants	1180.37	Arithmetic Intricacy	792.43
Data Transfer	436.62	Object Convolution Factor	38.09			Data Transfer	436.60
Comments	949.03					Comments	946.54

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

Code Structure	667.20	Code Structure	667.15
Inline Data	445.63	Inline Data	445.48

Table 3 above shows that all metrics (SLD, QTMs and PCMT) indicate that evolution occurred between V(1.6) and V(3.2), except for QLMs, in which has some standards showed evolution and others did not.

The Rate of Change between V(1.1) and V(1.9) in Magento System. Data analysis from Table 4 shows that the rate of change between V1.1 to V1.9 was high in the SLD, QTMs and PCMT areas, compared to QLMs being quite good in some standards, with negative changes in others. The high rate in SLD area was in the Arithmetic Intricacy standard (205.21%), whereas the high rate in QTMs is for High Quality Comments (196.14%). Moreover, the highest rate in the PCMT area was for Arithmetic Intricacy (204.91%), whereas the highest rate in QLMs was for Code Quality Notes Count (98.80%). On the contrary, the lowest rate in SLD area was for the Data Transfer standard (120.70%), whereas the lowest rate in QTMs was for Single Line Comments (93.74%). Moreover, the lowest rate in PCMT area was for Data Transfer (120.70%), whilst the lowest rate in QLMs was for Code Structure Modularity (-6.84%).

Table 4. Rate of change between V(1.1) to V(1.9) in Magento System

1. Statistical Labour Distribution		2. Quality Measurements		3. Quantitative Metrics		4. Project Code Meter Time	
Package	%	Package	%	Package	%	Package	%
Coding	139.94	Code Quality Notes Count	98.80	Files	123.49	Coding	136.93
Debugging	132.84	Code to Comment Ratio	15.78	Logical Lines of Code	129.31	Debugging	132.83
Testing	133.84	Essential Comment Factor	22.72	Single Line Comments	93.74	Testing	133.90
Flow Complexity	128.07	Code Structure Modularity	-6.84	Multi Line Comments	187.55	Flow Complexity	128.06
Object Vocabulary	138.10	Logic Density	-1.09	High Quality Comments	196.14	Object Vocabulary	138.10
Object Conjuration	154.47	Source Divergence Entropy	-4.83	Strings	142.50	Object Conjuration	154.45
Arithmetic Intricacy	205.21	Information Diversity Factor	6.87	Numeric Constants	102.17	Arithmetic Intricacy	204.91
Data Transfer	120.70	Object Convolution Factor	11.11			Data Transfer	120.70
Comments (CM)	141.66					Comments	125.87
Code Structure	125.87					Code Structure	141.46
Inline Data	141.97					Inline Data	141.84

This section has made a compression between the rate of change of four metrics between Moodle and Magento systems. Table 3 and 4 above show that all metrics (SLD, QTMs and PCMT) indicate that evolution occurred between all versions in Moodle and Magento, except for QLMs, in which has some standards showed evolution and others did not such as (Code Structure Modularity and Logic Density) in Moodle system and (Code Structure Modularity, Source Divergence Entropy and Logic Density) in Magento system.

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

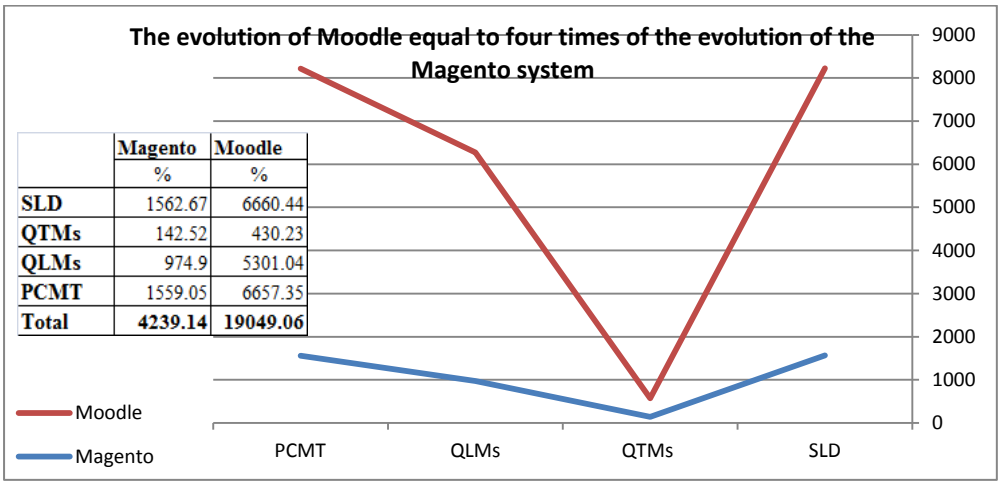

Figure 8. The rate of change of four metrics between Moodle and Magento system

As we can see in Table 3 and 4, the rate of change of evolution in Moodle equal to four times of the evolution of the Magento system. The total proportion of four metrics in Moodle is 19049.06 and in Magento is 4239.14. This means Moodle system has high evolution and has 4.49 times comparing with Magento system as in Figure 8.

4.5 Applying Lehman's Laws to the nine versions of Moodle

In this section, Lehman's laws were applied in order to observe and determine the scale of the evolution of the nine versions of Moodle software. For most software, including Moodle, these laws indicate the correct means of achieving rapid evolution. To ensure accuracy, data from the four areas described above were obtained in order to observe whether the nine versions of Moodle were compatible with the eight laws.

After reviewing data, as described in Appendix E, a more detailed picture emerges, as shown in Figure 9. This suggests that Laws 1 and 6 are compatible with the overall trends in the Moodle data and obtained a high number between 400 and 450. This demonstrates that Moodle is being continually modified, is satisfactory for use and continually aims to increase and maintain lifetime user satisfaction.

Laws 2 and 3 are generally compatible with overall trends in the Moodle data and also had a high number of between 350 and 400. This shows that Moodle has undergone development, increasing its complexity and providing more time for maintenance; software evolution processes are also self-regulating and controlled by itself.

However, Laws 4, 5 and 8 are compatible with some trends in the Moodle data and had a score of between 250 and 300. This shows that the average effective activity rate has been shortened in Moodle, and does not vary through the lifetime of the system. It also has limitations in terms of the active lifetime of software and the content of successive issues is

statistically invariant. Finally, the result of law 7 is nearly zero, which shows that Moodle has been maintained and adapted to achieve operational environmental modification.

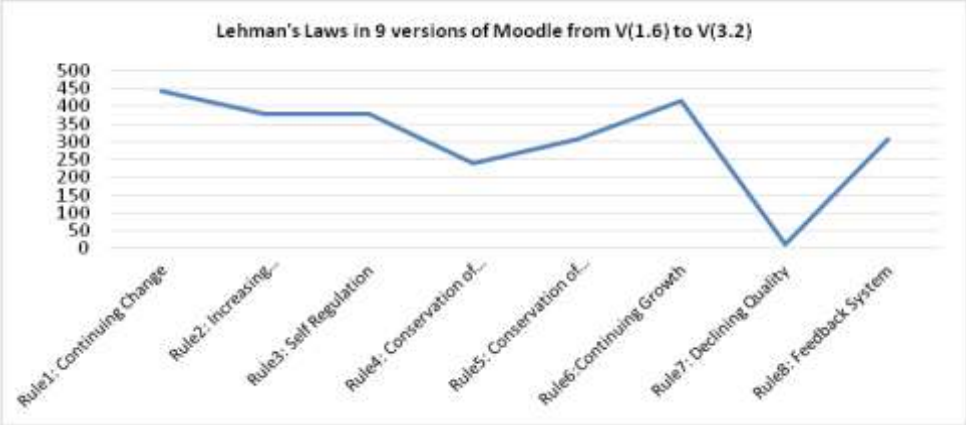

Figure 9. Applying Lehman's Laws on 9 versions of Moodle

The results show that the Moodle system has successfully applied Lehman's laws to facilitate a high degree of evolution over the nine versions released during the past decade. Lehman's laws provide the Moodle system with the correct means to achieve rapid evolution.

4.6 Recommendations

As a result, this study has made the following recommendations:

1. Measuring the evolution of any system, whether commercial or non-commercial, is very important, and helps determine the following:
   - Advantages, in order to provide a good impression and to show that the system is working in correctly and in a way that supports these advantages.
   - Disadvantages will help the developers to spot any mistakes or errors and to work hard to correct them in the next version.
2. As described in this case study, the process of evolution is essential for any system. Understanding this will aid efficacious planning and coordination, not just in the short term, but also much further ahead, especially for large software packages such as Apache, Mozilla or Linux.
3. Controlling large FOSS is a now an important challenge, and a major part of this is how its environment should evolve, and in particular how improvements in the security and quality of these systems should be made. The evolution of FOSS should focus on coding and developing the requirements of the application.
4. The community in FOSS plays vital role in its evolution by using metrics, and obtaining an improved understanding of and providing explanations for issues related to the development and evolution of FOSS.

5. The modelled system and understanding of software evolution will:
   - Offer more explanation and improvements for any system; for example Project Code Metrics has system metrics tools to measure the evolution of software.
   - Improve the ability to generate processes for efficient and reliable system development.

5 Conclusions and Future Work

From time to time, it is very important to measure systems; this particularly applies to FOSS, which has large groups of developers working on the evolution and development of a system. For that reason, this paper aims to confirm that FOSS is one of the most important aspects of the evolution of software development, because it is open source and therefore easier to access the code without restrictions or costs. In addition, it offers the most important tools that may be used to develop FOSS.

The main aim of this study was to measure the evolution of FOSS. It was found that there has indeed been evolution between V(1.6) to V(3.2) of the Moodle system and also between V(1.1) to V(1.9) of the Magento system. This paper has made a compression between the rate of change of four metrics between Moodle and Magento systems. It showed that all metrics (SLD, QTMs and PCMT) indicate that evolution occurred between all versions in Moodle and Magento, except for QLMs, in which has some standards showed evolution and others did not. In addition, this study examined the evolution of Moodle by applying the eight Lehman laws of software evolution.

This study found that a number of different concepts in software engineering drive the FOSS industry, such as security, quality, and the reliability of reusability. Therefore, the future of software engineering should consist of industrial rules for FOSS. In addition, some FOSS is still being challenged by closed software, the evolution and development of which are often faster than for OSS.

ACKNOWLEDGEMENT

The authors wish to acknowledge contributions from many people, including Martin Dougiamas who is the author of Moodle. Also, author is indebted to the QU for its encouragement and financial support.

REFERENCES

1. Silberman, G. (2014) A Practical Approach to Working with Open Source Software, Silberman, G. (2014) A Practical Approach to Working with Open Source Software, Intellectual Property & Technology Law Journal, Vol. 26(6).

2. Alenezi, M. and Zarour, M. (2015) Modularity Measurement and Evolution in Object-Oriented Open-Source Projects. In Proceedings of the The International Conference on Engineering & MIS. ACM, Istanbul, pp: 1-7.
3. Alenezi, M. and Khellah. F. (2015) Architectural Stability Evolution in Open-Source Systems. In Proc. of the The International Conference on Engineering & MIS, ACM, USA, Vol. 7(1), pp. 35-39.
4. Gamalielsson, J. and Lundell, B. (2014) Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved?, Journal of Systems and Software, Vol. 89(1), pp. 128–145.
5. Koponen,. T. (2006) Evaluation Framework for Open Source Software Maintenance. In Proc. of the international Conference on Software Engineering Advances. IEEE Computer Society, Washington, pp. 52.
6. Dagienė, V. and Grigas, G. (2006) Quantitative evaluation of the process of open source software localization. Informatica, vol. 17(1), pp.3-12.
7. Li, Y. et al., (2011) Open source software adoption: motivations of adopters and a motivations of non-adopters. SIGMIS Database journal, Vol. 42(2) pp. 76-94.
8. Saeed, F. (2013) Comparing and Evaluating Open Source E-learning Platforms, International Journal of Soft Computing and Engineering, Vol. 3(3), pp. 244-249.
9. Al-Ajlan, A. and Zedan, H. (2008) Why Moodle, in Proc. 12IEEE International Workshop on Future Trends of Distributed Computing Systems, IEEE Press, China, pp. 58–64.
10. Sabine, G. and L. Beate, (2005) An evaluation of open source e-learning platforms stressing adaptation issues, in Procs of Fifth IEEE International Conference on Learning Technologies, IEEE: Ischia, Italy
11. Sclater, N. (2006) Moodle: Transforming Learning Transforming Institutions, in Moodle Regional User Group Conference. London: Packt Publishing.
12. Sauer R. (2007) Why Develop Open Source Software? The Role of Non-Pecuniary Benefits, Monetary Rewards and Open Source Licence Type, University of Southampton, Discussion Paper No. 3197.
13. Nakagawa, E. et al. Software Architecture Relevance in Open Source Software Evolution: A Case Study. In Proc. of the 32nd Annual IEEE international Computer Software and Applications Conference, IEEE Computer Society, Washington, vol. 00, pp. 1234-1239, 2008.
14. Wang, Y. et al. (2007) Measuring the evolution of open source software systems with their communities. SIGSOFT Softw. Eng. Notes, ACM, USA, Vol. 32(6), pp. 1-7,
15. Scacchi. W. (2010) The future of research in free/open source software development. InProcs of the FSE/SDP workshop on Future of software engineering research, ACM, USA, pp. 315-320.
16. Karus, S. and Gall, H. (2011) A study of language usage evolution in open source software. In Procs of the 8th Working Conference on Mining Software Repositories. ACM, USA, pp. 13-22.
17. Herraiz, I. et al., (2013) The evolution of the laws of software evolution. A discussion based on a systematic literature review, ACM Computing Surveys, Vol. 1(1).

18. Postner, K. and Jackson, S. (2014) Teaching open source (software), In Procs of the 45th ACM technical symposium on Computer science education, ACM, USA, pp. 734-734.
19. Scacchi, W. (2006) Software Evolution and Feedback, chapter 9 (Understanding Open Source Software Evolution (p 181-205), John Wiley and Sons Inc, New York.
20. Lehman, M. et al., (1997) Metrics and laws of software evolution - the nineties view. In Proceedings of the 4th International Software Metrics Symposium, pp: 20–32, Albuquerque.
21. Wu, J. (2006) Open Source Software Evolution and Its Dynamics, Thesis, Computer Science, University of Waterloo, Waterloo, Ontario, Canada,.
22. Patil, A. (2012) Emerging technologies in distance education and their impact on the stakeholders, International Conference 2012 on Sousse (ICEELI), pp. 1-8.
23. Llanos, J. and Castillo S. (2012) Differences between traditional and open source development activities. In Proceedings of the 13th international conference on Product-Focused Software Process Improvement, Springer-Verlag, Berlin, pp: 131-144
24. Burov, E. and Parfenov, R. (2014) Learning Analytics for Mixed E-Governance-E-Learning Projects. In Proces of the 2014 Conference on Electronic Governance and Open Society: Challenges in Eurasia. ACM, USA, pp: 34-37.
25. Henneke M. and Matthee M. (2012) The adoption of e-Learning in corporate training environments: an activity theory based overview. In Proc.s of the South African Institute for Computer Scientists and Information Technologists Conference, ACM, USA, pp. 178-187.
26. Yadav, N., et al., (2014) Developing an Intelligent Cloud for Higher Education, SIGSOFT Softw. Eng. Notes, Vol:39(1), pp. 1-5.
27. Pires J. and Cota M. P., (2010) Evolutive mechanism for E-Learning platforms: A new approach for old methods, IEEE EDUCON Conference, Madrid, 2010, pp. 891-894.
28. Carlos J. Costa and Manuela A. (2011) Analysis of e-learning processes. In Proceedings of the 2011 Workshop on Open Source and Design of Communication. ACM, USA, pp: 37-40.
29. Project Code Meter, User Manual, retrieved on November 2016 from http://www.projectcodemeter.com/
30. All official releases of Moodle https://docs.moodle.org/dev/Releases#Moodle_1.0
31. Daniel A., María R., and Slinger J. (2015) Relating Health to Platform Success: Exploring Three E-commerce Ecosystems. In Procs of the 2015 European Conference on Software Architecture Workshops. Croatia,pp: 1-6.

APPENDIX

Appendix A

Statistical Labor Distribution in all versions of Moodle by Time in Hours

	Versions of Moodle									
Package	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**% Change 9V**
Coding	26352	40385	99970	112184	133423	174659	162646	**175662**	174677	**562.86**

ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145
ORIGINAL ARTICLE

Debugging	22902	34174	81822	90858	107305	**139724**	129115	139371	137877	**502.03**
Testing	14415	21647	52221	58148	68739	88625	82838	**89499**	88724	**515.50**
Flow Complexity	27127	39580	90617	99340	116778	135393	134256	**144614**	140428	**417.67**
Object Vocabulary	12356	18925	49021	54252	63748	84861	81575	88361	**88565**	**616.78**
Object Conjuration	8791	14823	40568	46046	54812	67014	68342	74173	75117	**754.48**
Arithmetic Intricacy	1149	1747	4685	5415	6621	**12026**	9959	10356	10257	**792.69**
Data Transfer	8898	12576	27157	30991	36261	**52263**	44044	47666	47749	**436.63**
Comments	363	695	2195	2469	3017	3344	3666	**3937**	3808	**949.04**
Code Structure	3687	5978	16085	17978	21189	**36852**	26018	28296	28287	**667.21**
Inline Data	1295	1882	3681	4696	7039	**11251**	6737	7128	7066	**445.64**
Total Time	127335	192412	468022	522377	618932	806012	749196	**809063**	802555	**530.27**

Appendix B

Quality Measurements in all versions of Moodle

	Versions of Moodle									
Package	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**% Change 9V**
Code Quality Notes Count	753	1029	2547	2432	2770	**4275**	3453	3814	3499	**364.67**
Code to Comment Ratio	23	24	24	24	23	17	**25**	**25**	**25**	**8.6957**
Essential Comment Factor	64	65	62	63	63	40	**66**	65	65	**1.5625**
Code Structure Modularity	140	145	162	161	156	**188**	161	162	159	**13.57**
Logic Density	**104**	103	95	102	93	75	99	98	97	**-6.73**
Source Divergence Entropy	**68**	65	61	62	62	61	62	62	62	**-8.82**
Information Diversity Factor	453	499	527	540	**548**	503	533	535	540	**19.21**
Object Convolution Factor	21	23	26	27	27	20	28	28	**29**	**38.10**
Total Time	1626	1953	3504	3411	3742	**5179**	4427	4789	4476	**175.28**

Appendix C

Quantitative Metrics in all versions of Moodle

	Versions of Moodle									
Package	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**% Change 9V**
Files	2180	2801	6218	6615	7434	9197	9849	10594	**11106**	**363.58**
Logical Lines of Code	183563	280801	676863	754308	905887	**1504693**	1068966	1159295	1161993	**533.02**
Single Line Comments	40954	58346	116952	136808	156948	197357	203862	216214	**224271**	**447.62**
Multi Line Comments	5069	15478	62156	66901	79095	88406	96760	**104758**	97077	**1815.11**
High Quality Comments	42636	67619	164540	186369	216919	264300	276851	**296103**	294972	**591.84**
Strings	172099	226687	466908	556058	839564	**1351206**	751506	814433	808072	**369.54**
Numeric Constants	36644	89072	182956	278038	377296	**675927**	462710	470152	469180	**1180.37**
Total Time	483145	740804	1676593	1985097	2583143	**4091086**	2870504	3071549	3065671	**534.52**

Appendix D

Project Code Meter Time in all versions of Moodle

	Versions of Moodle									
Package	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**% Change 9V**
Coding	1581143	2423127	5998213	6731091	8005382	10479595	9758775	**10539736**	10480668	**562.85**
Debugging	1374133	2050487	4909342	5451493	6438339	**8383478**	7746938	8362317	8272651	**502.03**
Testing	864924	1298878	3133314	3488925	4124396	5317508	4970312	**5369991**	5323445	**515.48**

Journal of Engineering Technology (ISSN: 0747-9964)
Volume 6, Issue 1, Jan, 2017, PP.124-145

ORIGINAL ARTICLE

Flow Complexity	1627655	2374841	5437069	5960434	7006711	8123639	8055414	**8676890**	8425682	**417.66**
Object Vocabulary	741370	1135506	2941314	3255177	3824889	5091704	4894504	5301662	**5313902**	**616.77**
Object Conjuration	527513	889388	2434100	2762793	3288777	4020888	4100531	4450405	**4507044**	**754.39**
Arithmetic Intricacy	68961	104822	281135	324947	397264	**721588**	597560	621398	615435	**792.44**
Data Transfer	533904	754563	1629475	1859494	2175697	**3135816**	2642653	2859973	2864977	**436.61**
Comments	21835	41700	131719	148166	181029	200680	220004	**236255**	228513	**667.16**
Code Structure	221237	358698	965154	1078700	1271379	**2211166**	1561128	1697774	1697238	**946.54**
Inline Data	77724	112972	220900	281796	422368	**675097**	404228	427685	423972	**445.48**
Total Time	3820200	5772493	14040870	15671509	18568117	24180581	22476025	**24272044**	24076766	**562.85**

Appendix E

Applying Lehman's Laws on 9 versions of Moodle

Versions of Moodle	
Rule	V(1.6) to V(3.2)
Rule1: Continuing Change	**443.42**
Rule2: Increasing Complexity	**376.85**
Rule3: Self-Regulation	**376.86**
Rule4: Conservation of	**241.01**
Rule5: Conservation of Familiarity	**307.52**
Rule6:Continuing Growth	**415.93**
Rule7: Declining Quality	**11.6**
Rule8: Feedback System	**307.51**