Journal of Administrative and Economic Sciences • Vol 18, Issue 1 (2025) • p 585-606

Measuring Time, Cost, Complexity, Quality, and Security of Ten Free E-Learning Platforms and Their Compliance with Lehman's Eight Laws

Ajlan S. Al-Ajlan

Departement of Management Information System, College of Business and Economics, Qassim University, Buraydah 52571, Saudi Arabia

aajlan@qu.edu.sa

Abstract:

E-learning programs in all fields have become necessary, especially after the Covid-19 pandemic, so the use of free e-learning systems has spread and been widely used in many governmental and private institutions, and their provision and management has become an urgent contemporary issue. This paper has selected the top 10 free e-learning programs, and the full code for each program was downloaded and then the code was analyzed using Project Code Metrics, a professional software tool for analyzing and measuring the time, cost, complexity and quality of different parts of the selected programs through source code analysis. After downloading and analyzing 10 programs, the results were analyzed and matched with Lehman's eight laws.

Keywords: Free e-learning Systems, Open-Source Software, Project Code Metrics, and Lehman’s Laws

1. Introduction

Technology has developed rapidly since the invention of the computer, across all fields. Computer programs can be divided into two types: commercial software and open-source software (OSS). Commercial programs are encrypted, and researchers and developers cannot view the files they contain without the permission of the owner of the program, whether that is an individual or a company. By contrast, open-source programs are not encrypted, and researchers and developers, even users, can view, download, modify, and develop their files under license terms (Xuetao, et al.,2024, Silberman 2014 and Alenezi, et al., 2015).

The rapid progress in all fields, especially in technology, means that large software packages, whether open source or commercial programs, need continuous development in order to keep pace with this progress and maintain a competitive market position. The development of these programs is highly complex and takes a long time, with a high cost to add new and modern features and tools. The development of OSS is critical for it to compete with commercial software, which is distinguished by the financial support it receives from companies (Alenezi, et al., 2015 Arghavan, 2024).

The success of OSS is a powerful incentive for researchers, developers, and both for-profit and non-profit companies to work on developing and benefiting from this kind of software. Protecting the security of OSS is task, and represents a challenge for developers and researchers because being open source means that anyone can browse and modify the files (Silberman 2014). Despite this, this is a strong motivation and incentive to work to protect it, especially since working on it does not require permission and it is easy to access files and code without restriction or cost. In addition, OSS has forums within which a group of experts answer questions and inquiries in order to strive to develop these programs (Gamalielsson, et al., 2014 and Koponen 2006).

This paper aims to explore the evolution of open source software by studying ten open source software and comparing them using the analysis program Project Code Meter. It is a professional software tool that will be used to analyze and measure the time, cost, complexity, quality, and security of different parts of the selected software through source code analysis. Through this study, the results are analyzed and matched with Lehman's eight laws to determine which one is the most advanced and widespread. We conclude this study with some recommendations, observations, and suggestions for possible future work for these programs.

This paper will be organized as follows: The second section will briefly summarize the concept of e-learning. A description of the development of OSS will be presented in Section 3. The main section of this paper is section 4, which discusses the development of OSS using the Project Code Meter analytical program as a case study. Section 5 presents an analysis and discussion of the work done in this study. Finally, the conclusion and suggested future works are given in Section 6.

2. Literature Review

2.1. E-Learning Systems

E-Learning programs have been widely applied by for-profit, non-profit, and academic organizations. Researchers and developers have used open-source code as a

programming language to create and develop this software, which is free and available to anyone. The ease of accessing the code and downloading it, for free and without restrictions, has led to a huge revolution in the programming world, with the development of open-source programs including Internet servers (for example, Apache), operating systems (including Linux), email, and e-learning programs (such as Sakai, Moodle and ATutor) (Dagienė, et al., 2006 and Li, et al., 2011).

In the early 1970s, American scientist Richard Stallman developed OSS, creating a free version of the UNIX operating system. GNU is a system to ensure OSS code is open to all users. (Silberman 2014, Koponen 2006 and Dagienė, et al., 2006). In 2000, OSS began as a public organization project, and the first version was released in 2002. A German company then started to develop Star Division, which was acquired by Sun. OpenSource.org provides all the information needed to help users learn how to develop OSS (Li, et al., 2011).

Currently, there are more than 300 e-learning programs, and at least 75 of these are open-source and used as free e-learning programs (Saeed 2013). A study by (Al-Ajlan, et al., 2008) proved that some open-source programs, such as Moodle, are better than commercial e-learning programs. The present study will present a comparison of 10 open-source programs, listed below in Table 1 (Sabine, et al., 2005 and Sauer 2007).

Table 1: The most popular free E-Learning Systems

No	Software	No	Software
1.	Moodle	2.	ATutor
3.	AnaXagora	4.	Opigno
5.	ILIAS	6.	LON-CAPA
7.	OpenOLAT	8.	Fle3
9.	Sakai	10.	OpenACS

Since the COVID-19 pandemic, e-learning programs have been used in public education and higher education in all countries of the world; they have also been used in government organisations and the private sector. E-Learning provides an opportunity for users to communicate with each other electronically by holding online sessions for virtual classes for students and academics, as well as discussions among organisations and councils for private and government companies. Moreover, e-learning programs contain forums, chat functions, and email. Their use fosters a culture of education and self-training to develop and improve the capacity of employees and students with minimal effort and at low cost (Henneke et al., 2012, Postner, et al., 2014 and Patil, 2012).

There are many advantages that can be gained through the use of e-learning programs and the work being done to develop these to suit the needs of all users. The COVID-19 situation demonstrated to the world the advantages of e-learning, and opened the door for educational and other organisations to benefit from reducing costs, increasing profits, and the possibility of employees and students performing work and tasks from home. Among the many advantages of e-learning for educational organisations are that it is possible to conduct training courses, complete activities and tasks, participate in forums, track student progress and schedules, prepare for exams, and so on. Moreover, e-learning provides important benefits to employees, from the ability to complete the

tasks assigned to them easily and conveniently while they are at home, to enabling them to download files, attach documents, and send emails, etc. (Yadav, et al., 2014, Pires 2010, Llanos 2012). The employee enjoys, through the use of e-learning, complete privacy with follow-up by the manager. The use of the Internet provides easy and fast access at low cost to important information that helps the user to accomplish tasks with efficiency and accuracy (Burov, et al., 2014 and Carlos, et al., 2011).

2.2. Why use Open-Source Programs

The reasons for working with free e-learning systems, can be grouped into seven main reasons, as follows) Silberman 2014, Wang, et al., 2007, Scacchi 2010, Saini, et al., 2022 and Yiqiao 2023):

1) Free: Most open-source programs are free of charge, as the code can be obtained, modified, developed, and new things added according to the GNU General Public License. The lack of cost is the main reason for using OSS.
2) Auditing capability: The audit capability helps to facilitate the review process for OSS and enables users to audit these programs by displaying notes and problems on forums. This does not happen with closed-source commercial programs.
3) Openness: The ability to obtain the code for OSS and the possibility to modify and develop it and add new features according to a licence is what makes these programs 'open'. However, they are not safe, due to the fact that anyone can access the files and understand their nature, and thus penetrate and obtain important information when used by official bodies.
4) Flexibility: OSS allows organisations of all kinds to integrate with researchers and developers from around the world, which contributes to understanding the requirements of those programs and speeding up their implementation.
5) Speed: Open-source programs have no restrictions, so their implementation and development is fast, because their files and code are open and available.
6) High quality: The reason for the high quality of OSS is that thousands of researchers and developers from all over the world are working to improve, develop, and innovate new features, and work to solve problems related to securing these programs.
7) Technical support: Because OSS is free, wide-ranging support is available through the Internet. All OSS has online communities where documents, correspondence, forums, and wiki news are stored, as well as live chat.

2.3. Software Evolution

Nowadays, the managers and developers of OSS projects face significant challenges in controlling large-scale e-learning programs. These programs contain tools that are difficult to control in terms of their expansion, distribution, maintenance, and modification (Neil, 2023). Therefore, free e-learning faces a great challenge in regard to developing and updating e-learning programs, especially in improving their quality and protecting them from external interference. The development of e-learning programs consists of two main points, namely, how to develop tools for these programs, and the maintenance and improvement of the code (Wang, I. et al., 2007, Karus, et al., 2011 and Bruno et al., 2019).

This study, which focuses on developing e-learning programs, examines e-learning programs and monitors their development using metric technology, using 10 programs as a case study to monitor and examine the extent of their development.

Good management and care in developing and updating software has become crucial for the success of companies in achieving competitiveness in order to achieve the highest profit margin. Therefore, most companies have relied heavily on developing, updating, and innovating new ideas in open software, especially in the field of e-learning (Yiqiao, 2023, Bruno et al., 2019 and Franco, et al., 2023).

Recent studies have focused on evaluating software scripts, algorithms, and tools, alongside the impact of open science and the associated legal and ethical considerations necessary to ensure the quality of these programs. In contrast, the practices and challenges related to the sharing of software evolution datasets have garnered insufficient attention. To address this gap, a comprehensive study was conducted to analyze software evolution datasets published in the International Conference on Mining Software Repositories from 2017 to 2021. This investigation examined 200 research papers to identify the types of software evolution datasets that were shared, as well as the practices and challenges researchers encountered in the process of sharing or utilizing these datasets. The results demonstrated that this study expands and enriches existing research, offering valuable insights to assist researchers in sharing software evolution datasets in a modern, accurate, ethical, and trustworthy manner (David et al., 2024).

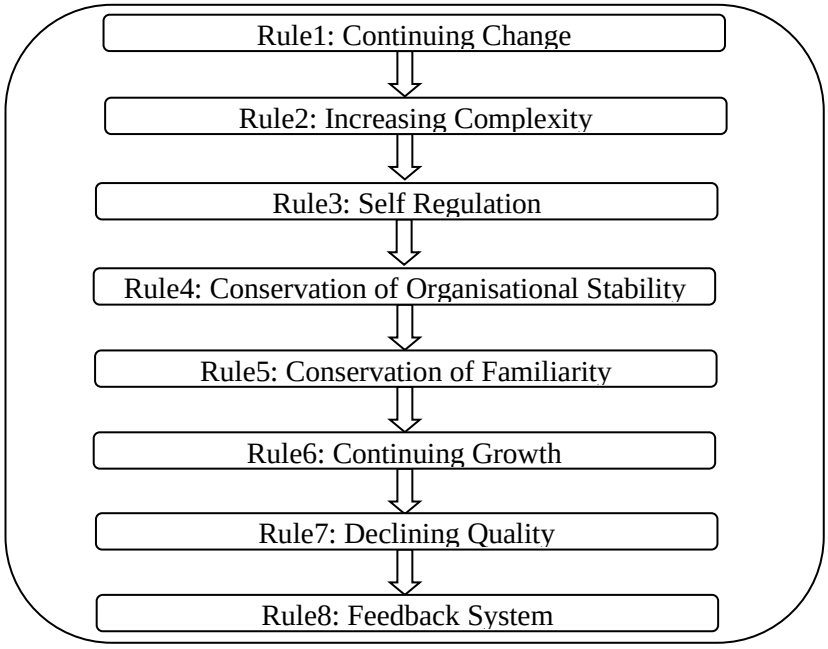
System failure prompts an organization to analyze the system to learn from the failure and correct errors to ensure the system is operating to its intended purpose. Systems development is critical to ensuring that these systems continue to operate, and scheduled analysis keeps them operating properly and as intended. The study collected publicly published incident reports, extracting and analyzing 104 action items. The initial findings of this analysis are reported in four points: (1) the objectives of the changes made, (2) the changes made to the systems, (3) the parts of the systems that are changed, and (4) what motivates those actions (Matt 2024).

Software development is the process of continuously updating, improving, and maintaining these systems. The more software systems are developed, the more complex and large they become in order to satisfy their users. As a result, many studies have been conducted on software evolution to understand the evolution pattern of systems and to propose techniques to overcome the problems inherent in software evolution. This study proposes a comprehensive software evolution dataset with time information on open source Java systems. To do this, this study proposes a four-step methodology: (1) selecting systems using a benchmark, (2) extracting (3) measuring their releases, and (4) creating their time series. Our dataset contains time series of 46 software metrics extracted from 46 open source Java systems, and we make it publicly available (Bruno et al., 2022).

In 1969, Meir Lehmann conducted an experimental study in cooperation with IBM, aiming to improve and develop the effectiveness of the company's programs, yet the study had no impact on the company's development practices. In 1974, Lehmann identified some constants, or laws for software development. After several years of

intense activity, the current version of these laws was published in 1996 (Herraiz, et al., 2013 and Lehman, et al., 1997).

Figure 1: The Lehman Laws to Understanding the Dynamics of Software Evolution



Lehmann introduced eight laws of software development and developed a theory to test software development engineering. These eight laws are important for understanding software development and suggesting the best solutions to the problems facing programs (Liguo et al., 2013 and Herraiz, et al., 2013). This study explores how development problems with software can be solved and also proposes some solutions to these problems, as shown in Figure 1.

As systems continue to evolve with rapid technological advancement, the importance of Lehmann's eight laws becomes clear. This study aims to establish indicators of the validity of Lehmann's laws by using and analyzing three systems. This study uses a General Systems Theory (GST) perspective when analyzing the laws of system evolution. An exploratory empirical approach consisting of four stages was used: 1) applying the GQM framework to validate Lehmann's laws; 2) collecting data using metrics applied to the target information system; 3) preparing data and processing metrics; and 4) quantitative analysis, examining metrics across different versions of systems. This study demonstrated that Lehmann's eight laws have been examined and validated, in part, in the context of the systems studied in this study. It also contributes to the understanding of system evolution with a focus on the practical application of Lehmann's laws (Augusto. et al., 2024).

3. Evolution of E-Learning Systems

This section will present an analysis of CPVTFe-LS (Current and Previous Versions in Ten Free e-learning Systems) using PCM (Project Code Metrics) as a case study. The study utilises FELS to uncover the differences between these ten systems, and will examine four areas of the CPVTFe-LS code: 1) statistical labour distribution, 2) quality measurements, 3) Project Code Meter time, and 4) quantitative metrics.

PCM is a professional tool designed to measure the degree of complexity in the special code and special projects of large programs, and to simplify this complexity to aid understanding, development and maintaining of the program. The main goal of PCM is to identify the nature of the code, regardless of its complexity and analysis, through factors that help reduce the time required for understanding software projects and, thus, reduce costs (Project Code Meter 2024).

PCM is designed to calculate cost, time, objectivity, repeatability, and also compare multiple versions of source code. This program analyzes Java, C, C++, J, PHP, JavaScript, MetaTrader, UnrealEngine, and more source files. It also generates detailed reports compatible with Microsoft Excel, HTML, MS Project, CSV, and cost and effort estimates using WMFP, COCOMO 81, COCOMO II 2000, and Revic 9.2 for comparison. It also calculates several code metrics including flow complexity, logical LOC, comments, constants, and strings. In addition, it warns of some code quality issues such as complex code structure and insufficient comments (Project Code Meter 2024). PCM has four standard metrics, which this study will use to measure the evolution of e-learning systems, as shown in Tables 2, 3, 4, and 5 below.

3.1. Statistical Labour Distribution

Statistical labour distribution (SLD) shows the number of hours. In large software projects, this method allows measuring of programming time in order to properly employ it according to the Weighted Micro Function Points Algorithm (WMFPA). By identifying business hours, SLD is useful for maintaining working hours and reducing the burden on programmers or business teams. SLD comprises of five standard measures, which are as follows (Project Code Meter 2024):

1. ***Timing:*** The time is displayed to the programmer in minutes and hours format independently, in order to show the time spent on coding, testing, and correction in the program file.
2. ***Encoding:*** The encoding time is calculated and displayed in the form of minutes and a percentage of the total time of the developer in the program file; this criterion is used only for coding in the program file.
3. ***Debugging:*** Here, the calculated time spent on the process of correcting errors in the program is displayed; the time is shown in the form of minutes and percentage of total time in the program file.
4. ***Testing:*** Only the calculated time spent by the programmer on the test phase is displayed in the program file. The calculated time is shown in minutes and as a percentage of the total file development test.
5. Object vocabulary, flow complexity, data transfer, embedded data, object configuration, code structure, computation, and comments: Scales the WMFPA source code measured for all files within the program. This parameter is again shown in minutes and as a percentage of the total file development time.

The results of the comparison between ten e-learning systems are divided based on the two versions (current version and previous version) in regard to the standard measures of SLD, as shown in Table 2. To summarise, the results show that Moodle and ILIAS appear to show the highest number of hours in their current versions. By contrast, Fle3, OpenOLAT, LON-CAPA, and ILIAS appear to show the lowest number of hours for the same versions. In the previous versions, ILIAS, Moodle, Dokeos, and Opigno show the highest number of hours. By contrast, AnaXagora, Fle3, LON-CAP, AOpenOLAT, and AnaXagora have the lowest number of hours for the same versions.

Table 2: The highest and lowest number of hours in the current and previous versions of ten e-learning systems, according to SLD

No	SLD	Current Version				Previous Version				%
		ELS	HHs	ELS	LHs	ELS	HHs	ELS	LHs	
1.	Object Conjuration	ILIAS	59459	Fle3	1916	ILIAS	43923	AnaXagora	1485	15.0
2.	Code Structure	Moodle	28287	Fle3	608	Moodle	23582	Fle3	598	9.1
3.	Testing	Moodle	88724	Fle3	2965	Moodle	73971	Fle3	2245	9.1
4.	Inline Data	ILIAS	7574	OpenOLAT	183	ILIAS	6842	LON-CAPA	187	5.1
5.	Comments	Moodle	3808	LON-CAPA	85	Moodle	3789	LON-CAPA	82	0.3
6.	Object Vocabulary	Moodle	88565	LON-CAPA	1740	dokeos	84824	Fle3	2043	2.2
7.	Arithmetic Intricacy	Moodle	10257	OpenOLAT	197	Moodle	9456	OpenOLAT	123	4.1
8.	Data Transfer	Moodle	47749	OpenOLAT	2385	Opigno	36763	OpenOLAT	3135	13
9.	Coding	Moodle	174677	Fle3	5351	Moodle	143579	AnaXagora	4365	9.8
10.	Debugging	Moodle	137877	Fle3	4797	Moodle	114589	LON-CAPA	4136	7.3
11.	Flow Complexity	Moodle	140428	OpenOLAT	1432	Moodle	124676	OpenOLAT	1134	5.9

As can be seen from Table 2, Moodle maintained the highest number of hours for both the previous and current version, in the SLD standards for coding, flow complexity, debugging, testing, code structure, arithmetic intricacy, and comments, with an increase in the current version of no more than 10%. However, for data transfer, the current version of Moodle measured higher than Opigno, with an increase of 13% compared to the previous version. On the other hand, ILIAS maintained the highest number of hours in the previous and current versions for the SLD standard for object conjuration (15%) and inline data (5.1).

3.2. Quality Measurements

All of the quality metrics (QLMs) in SCPVTFe-LS are shown in Table 3. These metrics provide an indication of some of the basic source code characteristics that affect maintainability, reuse, and peer review. The QLMs consist of eight standard metrics, as follows (Project Code Meter, 2024):

1. Number of Code Quality Observations: The number of warnings indicating problems affecting code quality is displayed. If the value is zero, this means that the quality of the code is high.
2. Code to Comment Ratio: This criterion indicates the balance between code words and the comment line. A value of 100 means that each symbol in the code has a comment. If the value is greater than 100, each code line contains more than one comment, but if the value is less than 100 this means that only some lines of code have comments.

3. Essential Comment Factor: This criterion expresses a balance between important code words and high-quality comment lines. If the value is 100, this means that each task code statement has a high-quality comment. But if the value is higher than 100, this means that the line contains more than one comment. If the value is less than 100, it means that only some lines of code have comments.
4. Code Structure Modularity: This criterion is concerned with code structure, which is divided into functions and classes. If the values are close to 100, this indicates that there is a good balance of code for each unit. If the values are above 100, this indicates hashed code. Conversely, if the values are less than 100, this indicates low typicality.
5. Logic Density: This standard focuses on how extensively logic is used within program code.
6. Source Divergence Entropy: In this standard, objects are processed by logic. If the values are high, this indicates manipulation.
7. Information Diversity Factor: This criterion attempts to reuse objects again. If the values are high, it means more reuse.
8. Object Convolution Factor: This parameter helps objects to interact with each other. If the values are higher, this means more handling, and therefore a more complex data flow.

The result of the comparison between ten e-learning systems, divided into two versions (current version and previous version), according to standard QLMs, as shown in Table 3. For the current versions, the results can be summarized as showing that the e-learning systems with the highest number of hours according to QLMs are AnaXagora (in LD, CCR, SDE, ECF), Fle3 (LC, SDE), ILIAS (CSM), LON-CAPA (IDF), and Dokeos (CQNC). By contrast, the e-learning systems with the lowest number of hours in standard QLMs are Dokeos (IDF, OCF, CSM), OpenOLAT (LD, CCR), ILIAS (CCR, ECF), ATutor (CCR), LON-CAPA (SDE), and Fle3 (CQNC).

Table 3: The highest and lowest number of hours in ten e-learning systems (current and previous versions) according to QLMs

No	QLMs	Current Version				Previous Version				%
		ELS	HHs	ELS	LHs	ELS	HHs	ELS	LHs	
1.	Information Diversity Factor	LON-CAPA	1137	Dokeos	224	LON-CAPA	980	Opigno	211	7.4
2.	Logic Density	AnaXagora **&** Fle3	125	OpenOLAT	43	ATutor	123	OpenACS	45	0.8
3.	Object Convolution Factor	OpenOLAT	39	Dokeos	15	Moodle	48	Fle3	16	-10.3
4.	Code Structure Modularity	ILIAS	177	Dokeos	81	ILIAS	167	LON-CAPA	68	2.9
5.	Code to Comment Ratio	AnaXagora	36	ATutor **&** ILIAS **&**OpenOLAT	11	AnaXagora	41	OpenOLAT	8	-6.5
6.	Source Divergence Entropy	AnaXagora **&** Fle3	72	LON- OpenACS	43	Dokeos	76	LON-CAPA	41	-2.7
7.	Code Quality Notes Count	Dokeos	4603	Fle3	67	Dokeos	3934	AnaXagora	57	7.8
8.	Essential Comment Factor	AnaXagora	125	ILIAS	28	AnaXagora	143	ATutor	22	-6.7

In previous software versions, as shown in above in Table 3, the results can be summarised as showing that the e-learning systems with the highest number of hours in standard QLMs are AnaXagora (in CCR, ECF), Dokeos (SDE, CQNC), Moodle (OCF), ILIAS (CSM), LON-CAPA (IDF), and ATutor (LC). By contrast, the e-learning systems with the lowest number of hours in standard QLMs are LON-CAPA (CSM, SDE), OpenOLAT (CCR), Opigno (IDF), ATutor (ECF), AnaXagora (CQNC), OpenACS (LD), and Fle3 (OCF).

3.3. Quantitative Metrics

Quantitative Metrics (QTMs) is a traditional metric that uses the Legacy Sizing Algorithms (LSA) approach and is oriented to obtain specific data. This approach is presented for the entire project based on context. QTMs consist of seven standard metrics, as follows (Project Code Meter 2024):

1. Files: The files within any project are measured and their number is determined.
2. Boolean Lines of Code: This indicates the number of lines in a code file. This standard can be used with LSA and price models as an advanced change to input precision for the SLOC (Somatic Source Lines to Token) parameter.
3. Multi-line Comments: This criterion indicates the number of comments that exceed more than one line of text.
4. Single-line Comments: This parameter displays comments in a single line of text.
5. High-quality Comments: This criterion indicates the number of comments, regardless of the number of lines in the code.
6. Strings: This parameter indicates the number of text strings specified in the source code. The external source code for the text is not calculated. For example, a PHP page contains an HTML body.
7. Numeric constants: This standard indicates the number of encoded digits the code contains.

The result of the comparison between ten e-learning systems (current version and previous version) according to the standard measures of QTMs are shown in Table 4. For the current software versions, the results can be summarized as showing that the e-learning systems with the highest number of hours according to standard QLMs are Moodle (in LLC, SLC, MLC, HQC, NC), Opigno (F), and ILIAS (S). By contrast, the e-learning systems with the lowest number of hours in standard QLMs are LON-CAPA (F, LLC, SLC, S), OpenOLAT (HQC, NC), and AnaXagora (MLC).

For the previous software versions, as shown below in Table 4, the results can be summarised as showing that the e-learning systems with the highest number of hours in standard QLMs are Moodle (in LLC, SLC, HQC, NC), Opigno (F, MLC), and ILIAS (S). By contrast, the e-learning systems with the lowest number of hours in standard QLMs are LON-CAPA (SLC, S), OpenOLAT (MLC, NC), ATutor (HQC), and Fle3 (F).

Table 4: The highest and lowest number of hours in ten e-learning systems (current and previous versions), based on QTMs

No	QTMs	Current Version				Previous Version				%
		ELS	HHs	ELS	LHs	ELS	HHs	ELS	LHs	
1.	Files	Opigno	12241	LON-CAPA	192	Opigno	10764	Fle3	296	6.4
2.	Logical Lines of Code	Moodle	1161993	LON-CAPA	37323	Moodle	1064746	AnaXagora	4353	4.4
3.	Single Line Comments	Moodle	224271	LON-CAPA	486	Moodle	197653	LON-CAPA	563	-6.3
4.	Multi Line Comments	Moodle	97077	AnaXagora	1726	Opigno	84310	OpenOLAT	1763	7
5.	High Quality Comments	Moodle	294972	OpenOLAT	8525	Moodle	283421	ATutor	8081	2.0
6.	Strings	ILIAS	1076577	LON-CAPA	15037	ILIAS	953016	LON-CAPA	13948	6
7.	Numeric Constants	Moodle	469180	OpenOLAT	10453	Moodle	383105	OpenOLAT	8264	10

3.4. Project Code Meter Time

For the current software versions, the results for Project Code Meter Time (PCMT) can be summarized as showing that Moodle has the highest total time (in C, D, T, FC, OV, OC, AI, DT, C, CS) and ILIAS in (ID). On contrast, Fle3, OpenOLAT and LON-CAPA appear to be the low number of hours in the same version as in Table5. The previous version, Moodle and ILIAS appear to be the high number of hours. On contrast, LON-CAP, AnaXagora, Fle3 and AOpenOLAT appear to be the low number of hours in the same version.

Table 5: The highest and lowest number of hours in ten e-learning systems (current and previous versions) according to PCMT

No	PCMT	Current Version				Previous Version				%
		ELS	HHs	ELS	LHs	ELS	HHs	ELS	LHs	
1.	Coding	Moodle	10480668	Fle3	321078	Moodle	98524847	Fle3	296420	-81
2.	Debugging	Moodle	8272651	Fle3	287648	Moodle	8139482	AnaXagora	275935	0.8
3.	Testing	Moodle	5323445	Fle3	177905	Moodle	4903639	LON-CAPA	148492	4.1
4.	Flow Complexity	Moodle	8425682	OpenOLAT	85927	Moodle	7963591	OpenOLAT	82864	2.8
5.	Object Vocabulary	Moodle	5313902	LON-CAPA	104456	Moodle	5186369	LON-CAPA	98451	1.2
6.	Object Conjuration	Moodle	4507044	Fle3	114988	Moodle	4385296	AnaXagora	116025	1.4
7.	Arithmetic Intricacy	Moodle	615435	OpenOLAT	11872	ILIAS	598573	OpenOLAT	10539	-1.4
8.	Data Transfer	Moodle	2864977	LON-CAPA	81725	Moodle	2795734	LON-CAPA	77837	1.2
9.	Comments	Moodle	228513	LON-CAPA	5118	Moodle	217459	LON-CAPA	4745	2.5
10.	Code Structure	Moodle	1697238	LON-CAPA	34441	Moodle	1558730	Fle3	30764	4.3
11.	Inline Data	ILIAS	454440	OpenOLAT	10995	ILIAS	429574	OpenOLAT	9073	-2.8

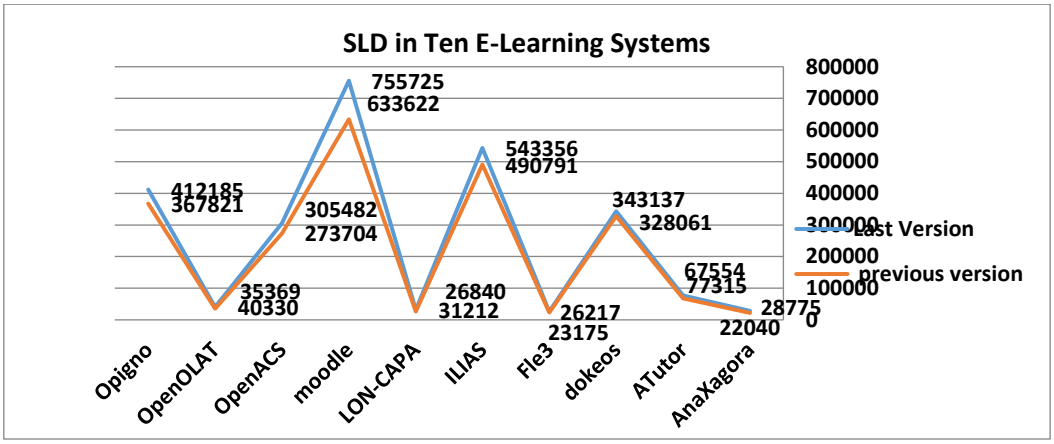
4. DISCUSSION AND RESULTS

In Section 3 above, we used PCM to measure the ten systems of SCPVTFe-LS as a case study. The study examined two versions of all SCPVTFe-LS systems and discovered differences between these versions across the four metrics chosen for the present study, which are described in depth in the above section. This discussion will summarise the results of the present study, focusing on each of the four areas in turn, which are: 1) SLD, 2) QLMs, 3) PCMT, 4) and QTMs.

4.1. Statistical Labour Distribution

Regarding the SLD criteria, the best evolution in SCPVTFe-LS is Moodle, which had the highest total number of hours (755,725) in all SLD criteria, as shown in Figure 2. The second-best evolution is ILIAS, which has 543,356 hours, and the third is Opigno, which has 41,285 hours. By this benchmark, Fle3 is the program with the lowest number of hours (26,217) across all SLD criteria.

Figure 2: SLD total for all ten e-learning systems

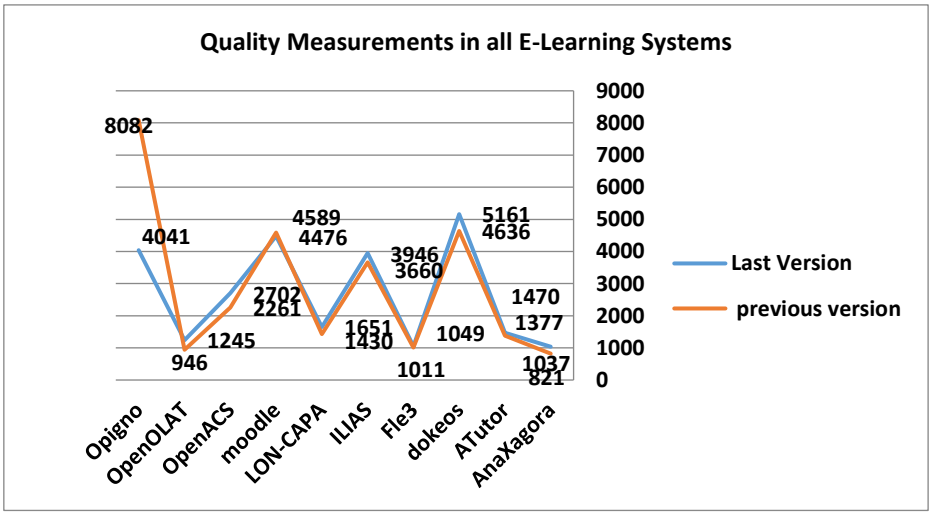


For the previous software versions, Moodle has the highest total number of hours (633,622) in all SLD criteria as in Figure 2. The second-best evolution is ILIAS, which has 490,791 hours, and the third is Opigno, which has 367,821 hours. By this benchmark, Fle3 is the program with the lowest number of hours (23,175) across all SLD criteria.

4.2. Quality Measurements

Figure 3 shows the QLMs for all SCPVTFe-LS systems. The best evolution among the current software versions is Dokeos, which has the highest total number of hours (5,161) across all SLD criteria. The second-best evolution is Moodle, which has (4,589) hours, and the third is Opigno, which has 4,041 hours. By this benchmark, AnaXagora is the program with the fewest hours (1,037) across all SLD criteria.

Figure 3: Total QTMs for all ten e-learning systems

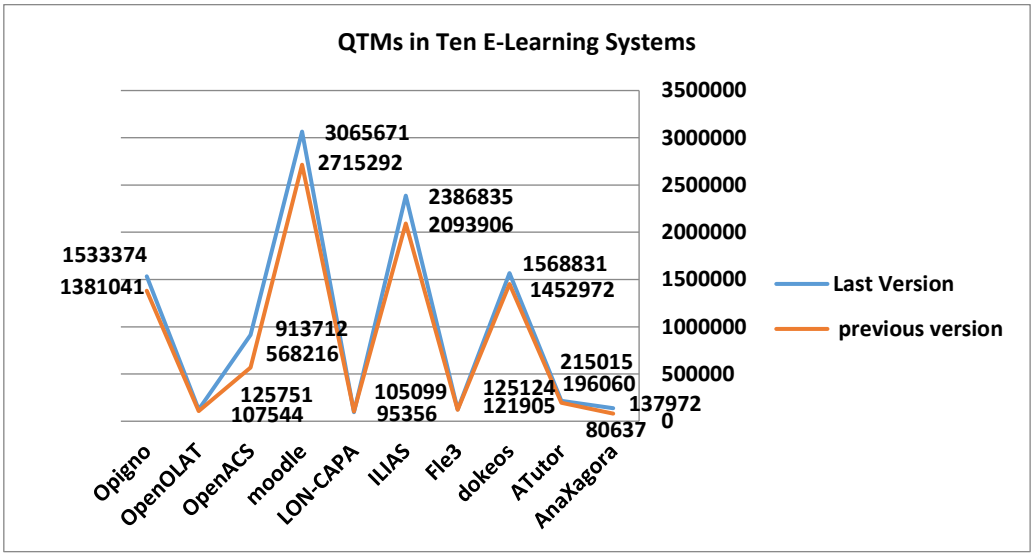


For the previous versions of the software, Dokeos has the highest total number of hours (4,636) across all SLD criteria, as shown in Figure 2. The second-best evolution is Moodle, which has 4,476 hours, and the third is ILIAS, which has 3,660 hours. By this benchmark, AnaXagora is the program with the lowest number of hours (821) in all SLD criteria.

4.3. Quantitative Metrics

In terms of the QTM criteria, Moodle is still a high-scoring evolution, with the highest total number of hours (306,571) of all QTM criteria, as shown in Figure 4. The second highest scoring evolution is ILIAS, which has 2,386,835 hours, and the third is Dokeos, which has 1,568,831 hours. By this benchmark, LON-CAPA is the program with the lowest number of hours (95,356) across all QTM criteria.

Figure 4: Total QTMs for all ten e-learning systems



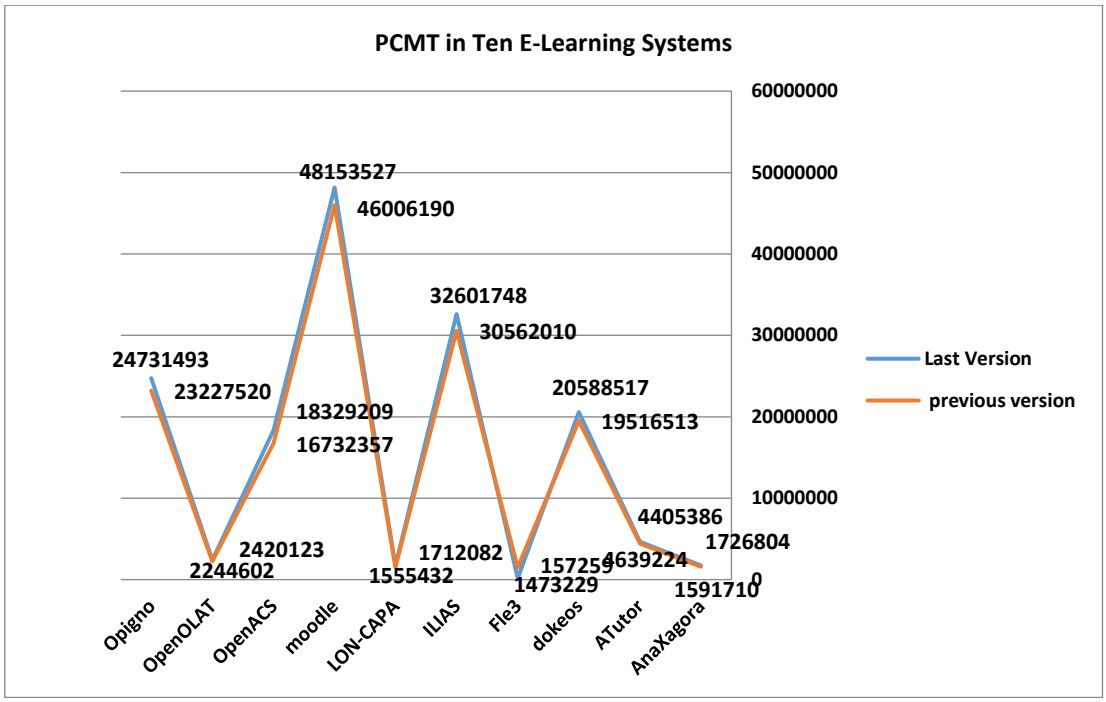
For the previous software versions, Moodle is still a high-scoring evolution, with the highest total number of hours (2,715,292) in all QTMs criteria, as shown in Figure 4. The second-best evolution is ILIAS, which has 2,093,906 hours, and the third evolution is Dokeos, which has 1,452,972 hours. By this benchmark, AnaXagora is the program that gets the fewest hours (80,637) across all QTM criteria.

4.4. Project Code Meter Time

For the PCMT criteria, Moodle is again a high-scoring evolution, with the highest total number of hours (48,153,527) in all PCMT criteria, as shown in Figure 5. The second-best evolution is ILIAS, which has 32,601,748 hours, and the third is Opigno, which has 24,731,493 hours. By this benchmark, Fle3 is the program that gets the fewest hours (1,573,259) across all PCMT criteria.

For the previous software versions, Moodle is again a high-scoring evolution, with the highest total number of hours (46,006,190) for all PCMT criteria, as shown in Figure 4. The second-best evolution is ILIAS, which has 30,562,010 hours, and the third evolution is Opigno, which has 23,227,520 hours. By this benchmark, Fle3 is the program that gets the fewest hours (1,473,229) across all PCMT criteria.

Figure 5: Totals for PCMT across all ten e-learning systems



4.5. The Rate Change of Four Metrics Between The SCPVTFe-LS Systems

This discussion will focus on the SCPVTFe-LS comparison, particularly the change in rates between the current and previous software versions. The highest rate across all standards in the SLD areas, between current and previous versions, is for Object Conjuration in the ILIAS software, which recorded 15%. The highest rate in QLMs was for Code Quality Notes Count in the Dokeos software, which recorded 7.8%. Moreover, the highest rate in QTM areas, between current and previous software versions, was for Numeric Constants in Moodle, which recorded 10%. Finally, the highest rate in PCMT was for Code Structure in Moodle software, which recorded 4.3%.

By contrast, the lowest rate in the SLD areas was for the Comments standard, which recorded 0.3% in Moodle software, but the lowest rate in QLMs was for the Object Convolution Factor, which recorded -10.3% in OpenOLAT software. Moreover, the lowest rate in QTMs was for Single-Line Comments, which recorded -6.3% in Opigno software, and the lowest rate in the PCMT areas was for Coding, which recorded -81% in Moodle software.

Data analysis based on Table 7 above indicates that the change in rate between the current and previous versions of the ten e-learning systems is high for SLD, where all evolutions in this stage are between 2% and 15%. The second highest evolution is in QTMs, where all evolutions in this stage are between -6.3% and 10%. For QLMs and PCMT, the systems perform well in some standards but in others show a minus rate.

This study proves that there has been no evolution in some of the SCPVTFe-LS systems, namely ATutor and OpenACS. The other systems have seen evolution; these systems are: Moodle, AnaXagora, ILIAS, OpenOLAT, Opigno, LON-CAPA, Dokeos, and Fle3.

Table 7: Changing Rate between ten e-Learning systems in current and previous versions

1. Statistical Labor Distribution		2. Quality Measurements		3. Quantitative Metrics		4. Project Code Meter Time	
Package	**%**	**Package**	**%**	**Package**	**%**	**Package**	**%**
Object Conjuration-ILIAS	15	Information Diversity Factor- LON-CAPA	7.4	Files- Opigno	6.4	Coding-Moodle	-81
Code Structure-Moodle	9.1	Logic Density-AnaXagora **&** Fle3	0.8	Logical Lines of Code -Moodle	4.4	Debugging-Moodle	0.8
Testing-Moodle	9.1	Object Convolution Factor-OpenOLAT	-10.3	Single Line Comments-Moodle	-6.3	Testing-Moodle	4.1
Inline Data-ILIAS	5.1	Code Structure Modularity- ILIAS	2.9	Multi Line Comments -Moodle	7	Flow Complexity -Moodle	2.8
Comments-Moodle	0.3	Code to Comment Ratio- AnaXagora	-6.5	High Quality Comments -Moodle	2.0	Object Vocabulary -Moodle	1.2
Object Vocabulary-Moodle	2.2	Source Divergence Entropy- AnaXagora **&** Fle3	2.7	Strings-ILIAS	6	Object Conjuration-Moodle	1.4
Arithmetic Intricacy-Moodle	4.1	Code Quality Notes Count- Dokeos	7.8	Numeric Constants-Moodle	10	Arithmetic Intricacy-Moodle	-1.4
Data Transfer-Moodle	13	Essential Comment Factor- AnaXagora	-6.7			Data Transfer-Moodle	1.2
Coding-Moodle	9.8					Comments-Moodle	2.5
Debugging-Moodle	7.3					Code Structure-Moodle	4.3
Flow Complexity-Moodle	5.9					Inline Data-ILIAS	-2.8

4.6. Applying Lehman's Laws in SCPVTFe-LS

In this part of the research, Lehman's laws were applied to monitor the development in SCPVTFe-LS. In most of the programs, these laws worked to achieve rapid development to ensure quality and accuracy in obtaining data related to the four standards that were used in this study. This indicates that the SCPVTFe-LS systems are in line with Lehman's laws.

After checking the data shown in Appendix E, there was found to be no evolution in some of the SCPVTFe-LS systems, namely: ATutor and OpenACS. The other systems – Moodle, AnaXagora, Fle3, Dokeos, OpenOLAT, ILIAS, LON-CAPA, and Opigno – all showed evolution.

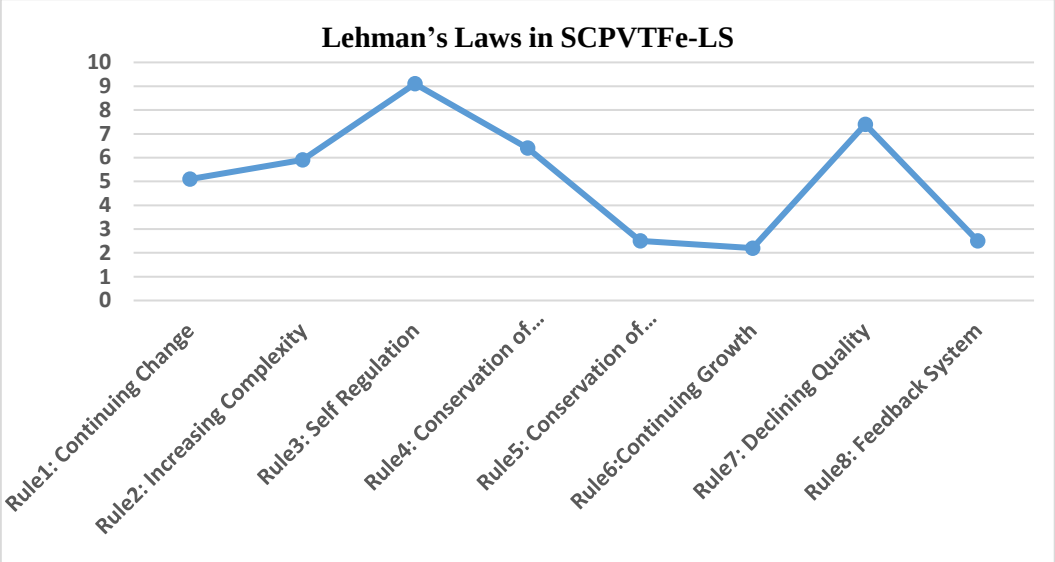
Rules 3 and 7: with regard to the third and seventh laws, a high percentage was achieved for these, ranging between 7 and 9 of the general trends of these systems. This proves that these systems have developed continuously and rapidly, which has led to an increase in the complexity of these systems and, therefore, requires an increase in support and maintenance.

Rules 1, 2, and 4: it became clear that the first, second, and fourth Lehman's Laws are compatible with the general laws in the data of SCPVTFe-LS, and they are between 5 and 6, as shown in Figure 6. This proves that these systems are being constantly modified and updated in order to obtain user satisfaction, increase the life-cycle of the system, and keep pace with new developments.

Rules 5, 6, and 8: the study found that laws 5, ,6 and 8 are the laws that least affect the ten systems under study. These laws are compatible with some of the policies and trends of some regimes and achieved a low percentage, ranging between 2 and 3. This

indicates that the feedback system in these systems is not active and interaction with users is low. In addition, it suggests that the continuation of growth and development in these systems is slow and is not keeping pace with the rapid changes of the current era. As for maintaining intimacy within these programs, it is low, unstable, and volatile.

Figure 6: Lehman's Laws in SCPVTFe-LS



4.7. Objectives and Recommendations

It is important to measure systems from time to time in order to develop them. For free e-learning systems, there are large groups of developers who are constantly developing these systems to compete with paid e-learning systems. This paper helps developers and researchers and encourages them to examine and measure these systems in order to uncover errors in previous versions and work to solve these issues in new versions.

The main objective of this study was to measure the development of SCPVTFe-LS. The study has found that there has indeed been an evolution among these ten systems. Based on the results of this study, some implications and recommendations are put forth:

1. It is important to measure the development of any system because this will uncover the strengths and weaknesses of these systems. This is beneficial because:
    a. Discovering strengths will demonstrate that these systems are working properly.
    b. Discovering errors will help developers to understand and work to correct them in order to resolve them in new versions of the software.
2. This study proves that the systems must be continually modified to ensure they are acceptable, or it will become progressively less satisfactory to use. A development process is necessary for any system, but especially free OSS. This development will help in planning and coordination, in both the short and the long term.
3. Any existing system will face many problems that necessitate improvement; among these problems is the continuous updating of the program to be compatible with

new technology and to avoid problems in the latest version. This will only be achieved by continuously measuring the performance of these programs with evaluation tools, such as PCM.

4. Controlling systems, especially free OSS, is an important challenge today, particularly with regard to how to develop the environment of these systems and how to introduce the necessary improvements to secure these systems from external interference.
5. The development of programs leads to an increase in the complexity of operations, and this requires updating and self-maintenance of these systems.
6. The organisational stability of the program is critical so the organisational environment of the system must be preserved.
7. Lack of interest in developing, updating, and maintaining these programs will lead to a decrease in their quality, and thus in users' confidence in them. This means they will be less competitive compared with other programs. Therefore, continuous measurement of these systems is needed to enhance the quality of the programs by making the necessary updates and fixing errors.
8. The results will be valuable in providing a deeper understanding for researchers and professionals interested in information systems evolution.

There are some challenges and problems that this study faced, which are:

1. Dealing with 10 programs and analyzing them requires a long time, high concentration, and double effort to reach the desired results.
2. Difficulty in obtaining previous studies to analyze the quality of the source code.
3. It was difficult to download the entire source code for some programs, so many programs were excluded for this reason.
4. The huge and huge size of the source code for some programs and the difficulty of analyzing it with the Pw program
5. It is often difficult to conduct an effective source code analysis, especially in complex and large-scale programs.

5. Conclusion And Future Work

The quality of any OSS program gives an indication of the efficiency, reliability, and ability of these programs to secure protection and to self-update. Often, developing OSS projects comes with difficulty maintaining the quality of these programs. Therefore, the managers of such projects apply methods and techniques that use algorithms to evaluate the quality of the project, using a variety of metrics. For any high-quality software development, it is recommended that there be low coupling and high cohesion between program modules.

The importance of measuring the development of any system helps in discovering the strengths and weaknesses of these systems, as discovering the strengths will prove that

these systems are working properly, and discovering errors will help developers understand them and work to correct them in order to avoid them in new versions.

This paper has measured the evolution of ten OSS programs in the field of e-learning, looking at two versions of each (the current version and previous version). Four main metric-based tools were used to measure the source code, and Project Code Metrics were also used. The results of the study show that two of the ten programs have not evolved; these programs are ATutor and OpenACS. The others have seen evolution, at a level that differs from one program to another; these programs are Moodle, AnaXagora, Fle3, Dokeos, OpenOLAT, ILIAS, LON-CAPA, and Opigno.

In addition, this study has used Lehman's laws to monitor the development of SCPVTFe-LS. In most programs, these laws have worked to achieve rapid development to ensure quality and accuracy in obtaining data related to the four standards used in this study. Therefore, it can be concluded that the SCPVTFe-LS are in line with Lehman's laws.

Furthermore, this paper has identified a number of studies that have described the evolution of OSS as having age, like people, highlighting that it must adapt to changing requirements and environments. Additionally, it has shown that the ten OSS programs have been a concern for a number of developers and have long been at the core of investigations. Thus, a number of theoretical models for the evolution of OSS have been developed and empirically examined. Measuring commercial software is now relatively easy to do, being within the skill set of most developers.

Acknowledgement

The authors wish to acknowledge contributions to Qassim University for its encouragement and financial support.

References

1. Xuetao, L, et al. (2024) Systematic Literature Review of Commercial Participation in Open Source Software. ACM Trans. Softw. Eng. Methodol.
2. Silberman, G. (2014). A Practical Approach to Working with Open Source Software, Intellectual Property & Technology Law Journal, 26(6), 76–89.
3. Alenezi, M. & Zarour, M. (2015) Modularity Measurement and Evolution in Object-Oriented Open-Source Projects. In Proceedings of the The International Conference on Engineering & MIS. (pp. 7-16) ACM, USA.
4. Alenezi, M. & Khellah. F. (2015) Architectural Stability Evolution in Open-Source Systems. In Proc. of the The International Conference on Engineering & MIS. (pp. 35-39) ACM, USA.
5. Arghavan S. and Jinghui C. (2024) Characterizing Usability Issue Discussions in Open Source Software Projects. Proc. ACM Hum.-Comput. Interact. 8, CSCW1, No: 30 pp.1- 26.
6. Gamalielsson, J. & 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), 128–145.

7. Koponen, T. (2006) Evaluation Framework for Open Source Software Maintenance. In Proc. of the international Conference on Software Engineering Advances. IEEE Computer Society, (pp. 52-59) Washington.
8. Dagienė, V. & Grigas, G. (2006) Quantitative evaluation of the process of open source software localization. Informatica, 17(1), 3-12.
9. Li, Y. et al., (2011) Open source software adoption: motivations of adopters and a motivations of non-adopters. SIGMIS Database journal, 42(2),76-94.
10. Al-Ajlan, A. & Zedan, H. (2008) Why Moodle, in Proc. 12IEEE International Workshop on Future Trends of Distributed Computing Systems, IEEE Press, (pp. 58-64) China.
11. Sabine, G. & 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: (pp. 78-91) Ischia, Italy.
12. Saeed, F. (2013) Comparing and Evaluating Open Source E-learning Platforms, International Journal of Soft Computing and Engineering, 3(3), 244-249.
13. Sauer R. (2007) Why Develop Open Source Software? The Role of Non-Pecuniary Benefits, Monetary Rewards and Open Source Licence Oxford Review of Economic Policy, 23(4), 605-619.
14. Henneke M. & 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, (pp. 178-187), ACM, USA.
15. Postner, K. & Jackson, S. (2014) Teaching open source (software), In Procs of the 45th ACM technical symposium on Computer science education, (pp. 722-734).
16. Patil, A. (2012) Emerging technologies in distance education and their impact on the stakeholders, International Conference 2012 on Sousse (ICEELI), (pp. 1-5).
17. Yadav, N., et al., (2014) Developing an Intelligent Cloud for Higher Education, SIGSOFT Softw. Eng. Notes, 39(1), 1-17.
18. Pires J. and Cota P. (2010), Evolutive mechanism for E-Learning platforms: A new approach for old methods, IEEE EDUCON 2010 Conference, Madrid, (pp. 891-894).
19. 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, (pp. 131-144). Verlag, Berlin: Springer, Heidelberg.
20. Burov, E. & Parfenov, R. (2014) Learning Analytics for Mixed E-Governance-E-Learning Projects. In Proceedings of the 2014 Conference on Electronic Governance and Open Society: Challenges in Eurasia (EGOSE '14), (pp. 34-37).
21. 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 (OSDOC '11), (pp. 37-40).
22. Wang, Y. et al. (2007) Measuring the evolution of open source software systems with their communities. SIGSOFT Softw. Eng, 32(6), 1-7.
23. 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, (pp. 315-320). USA: ACM.

24. Saini, M., Arora, R., & Adebayo, S. O. (2022). In-Depth Analysis and Prediction of Coupling Metrics of Open Source Software Projects. Journal of Information Technology Research (JITR), 15(1), 1-16.
25. Yiqiao J., et al. (2023) Code Recommendation for Open Source Software Developers. In Proc. of the ACM Web Conference 2023 (WWW '23). Association for Computing Machinery, pp. 1324–1333.
26. Neil S. (2023) Locking Down Secure Open Source Software. Commun. ACM, Vol: 66(5), pp:13–14.
27. Karus, S. & Gall, H. (2011) A study of language usage evolution in open source software. In Procs of the 8th Working Conference on Mining Software Repositories, (pp. 13-22).
28. Bruno S., et al., (2022) A time series-based dataset of open-source software evolution. In Proc of the 19th International Conference on Mining Software Repositories (MSR '22), No: 13, pp.702–706.
29. Bruno, L. et al. (2019) Analysis of Coupling Evolution on Open Source Systems. In Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse, (pp. 23–32). Association for Computing Machinery.
30. Franco, F., et al. (2023). A systems interpretation of the software evolution laws and their impact on technical debt management and software maintainability. Software Quality Journal, 31(1), 179-209.
31. David B., et al., (2024) Sharing Software-Evolution Datasets: Practices, Challenges, and Recommendations. Proc. ACM Softw. Eng. 1, FSE, No: 91, pp.1-24.
32. Matt P. & Jonathan S. (2024) Post-Incident Action Items: Crossroads of Requirements Engineering and Software Evolution. In Proc. of the 1st IEEE/ACM Workshop on Multi-disciplinary, Open, and RElevant Requirements Engineering (MO2RE 2024)., No: 21, pp.1-7.
33. Herraiz, I., et al (2013). "The evolution of the laws of software evolution". ACM Computing Surveys. 46 (2), 1–28.
34. Lehman, M. et al., (1997) Metrics and laws of software evolution - the nineties view. In Proceedings of the 4th International Software Metrics Symposium, 1997 (pp 20–32), Albuquerque.
35. Liguo Yu & Alok Mishra (2013) An Empirical Study of Lehman's Law on Software Quality Evolution, in International Journal of Software and Informatics, 7(3), 469-481.
36. Augusto, B. et al., (2024) An Exploratory Study on the Validation of Lehman's Laws. In Proc of the 20th Brazilian Symposium on Information Systems (SBSI '24). No: 47, , pp: 1–10.
37. Project Code Meter (2024). User Manual, Retrieved October 02, 2024 from http://www.projectcodemeter.com

Appendix

Appendix A

Statistical Labor Distribution in high and low number of hours in current and previous versions in ten e-Learning systems

Statistical Labor Distribution				
Software	**Last Version**	**previous version**	**difference**	**Ratio**
AnaXagora	28775	22040	6735	13.3
ATutor	77315	67554	9761	6.7
dokeos	343137	328061	15076	2.2
Fle3	26217	23175	3042	6.2
ILIAS	543356	490791	52565	5.1
LON-CAPA	31212	26840	4372	7.5
moodle	755725	633622	122103	8.8
OpenACS	305482	273704	31778	5.5
OpenOLAT	40330	35369	4961	6.6
Opigno	412185	367821	44364	5.7

Appendix B

Quality Measurements in high and low number of hours in current and previous versions in ten e-Learning systems.

Quality Measurements				
Software	**LV**	**PV**	**difference**	**ratio**
AnaXagora	1037	821	216	11.6
ATutor	1470	1377	93	0.3
dokeos	5161	4636	525	5.4
Fle3	1049	1011	38	1.8
ILIAS	3946	3660	286	3.8
LON-CAPA	1651	1430	221	7.2
moodle	4476	4589	-113	-1.2
OpenACS	2702	2261	441	8.9
OpenOLAT	1245	946	299	13.6
Opigno	4041	3736	305	3.9

Appendix C

Quantitative Metrics in high and low number of hours in current and previous versions in ten e-Learning systems.

Quality Measurements				
Software	**LV**	**PV**	**difference**	**ratio**
AnaXagora	137972	80637	57335	26.2
ATutor	215015	196060	18955	4.6
dokeos	1568831	1452972	115859	3.8
Fle3	125124	121905	3219	1.3
ILIAS	2386835	2093906	292929	6.5
LON-CAPA	95356	105099	-9743	-4.9
moodle	3065671	2715292	350379	6.1
OpenACS	913712	568216	345496	23.3
OpenOLAT	125751	107544	18207	7.8
Opigno	1533374	1381041	152333	5.2

Appendix D

Project Code Meter Time in high and low number of hours in ten e-Learning systems in current and previous versions in PCMT

Project Code Meter Time				
Software	**LV**	**PV**	**difference**	**ratio**
AnaXagora	1726804	1591710	135094	4.1
ATutor	4639224	4405386	233838	2.6
dokeos	20588517	19516513	1072004	2.7
Fle3	1573259	1473229	100030	3.3
ILIAS	32601748	30562010	2039738	3.2
LON-CAPA	1712082	1555432	156650	4.8
moodle	48153527	46006190	2147337	2.3
OpenACS	18329209	16732357	1596852	4.6
OpenOLAT	2420123	2244602	175521	3.8
Opigno	24731493	23227520	1503973	3.1

Appendix E

Applying Lehman's Laws in SCPVTFe-LS

	SCPVTFe-LS	
Rule	**Lehman's Laws in SCPVTFe-LS**	**Software**
Rule1: Continuing Change	5.9	ILIAS
Rule2: Increasing Complexity	9.1	Moodle
Rule3: Self-Regulation	6.4	Moodle
Rule4: Conservation of	2.5	Opigno
Rule5: Conservation of Familiarity	2.2	Moodle
Rule6:Continuing Growth	7.4	Moodle
Rule7: Declining Quality	2.5	LON-CAPA
Rule8: Feedback System	5.9	Moodle