applied sciences

Article

Internal Quality Evolution of Open-Source Software Systems

Mamdouh Alenezi

College of Computer and Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia; malenezi@psu.edu.sa

Abstract: The evolution of software is necessary for the success of software systems. Studying the evolution of software and understanding it is a vocal topic of study in software engineering. One of the primary concepts of software evolution is that the internal quality of a software system declines when it evolves. In this paper, the method of evolution of the internal quality of object-oriented open-source software systems has been examined by applying a software metric approach. More specifically, we analyze how software systems evolve over versions regarding size and the relationship between size and different internal quality metrics. The results and observations of this research include: (i) there is a significant difference between different systems concerning the LOC variable (ii) there is a significant correlation between all pairwise comparisons of internal quality metrics, and (iii) the effect of complexity and inheritance on the LOC was positive and significant, while the effect of Coupling and Cohesion was not significant.

Keywords: software engineering; software evolution; software metrics; internal quality; open-source software systems

check for updates

Citation: Alenezi, M. Internal Quality Evolution of Open-Source Software Systems. *Appl. Sci.* **2021**, *11*, 5690. https://doi.org/10.3390/app11125690

Academic Editor: Vito Conforti and Federico Divina

Received: 20 May 2021
Accepted: 17 June 2021
Published: 19 June 2021

1. Introduction

Evolution is a normal phenomenon in the life-cycle of software systems. Software evolution happens in incremental steps as a reaction to changes in the environment, purpose, or use of the considered software system [1]. Changes of a software system may have an impact on its quality, referring to aspects such as correctness, consistency, usability, and maintainability [2]. Evolving software should preserve or even improve its quality throughout software changes. The evolution of software is the process of building, maintaining, and modernizing software systems [3]. This process is key to the success of the software systems since it allows the addition or improvement of different aspects of the system. This process brings different changes in the software design and architecture, which requires the continuous adaptation of both the internal and external quality attributes.

Software evolution can be seen as an ongoing process of change [1]. Software systems can easily adapt to changes over time. If the software does not support the change, it gradually becomes unusable [4]. The software evolution enables new requirements to be incorporated into software systems to meet stakeholders' business objectives. According to all software development processes, the software should be developed in response to the need to adapt to the environment or maintain user satisfaction [4]. To reduce software production costs, both managers and developers must understand the factors that drive software evolution and take proactive steps that facilitate changes and ensure that software does not decay [5,6]. Continuing for an extended period after inception is one successful characteristic of software systems, including open-source ones. For software systems to survive in this high-paced world, they must evolve continuously. The dynamic behavior of software systems is studied by software evolution while it is maintained and enhanced over its lifetime [7].

Engineering practice involves the measurement of the internal quality of the source code during software development. For every software product, quality is an essential characteristic. We can look at software quality from two perspectives, one is from the customer perspective, and another one is from the developer perspective [8]. Easy to use, accurate, and meet the requirements are characteristics that the customer is usually looking for. At the same time, developers look at cost, ease of maintenance, and reuse. The evolution of software has a significant relationship with the quality of software.

The study of software evolution is primarily performed through two main approaches, process improvements and exploratory [9]. The experimental approach addresses the scientific point of view of why software evolves. On the other hand, the process improvement approach addresses the engineering perspective, considering the cause-effect relation between changes and trends.

Typically, software systems grow in size after long-term evolution [10]. As software systems evolve, both their size and complexity will also grow unless specific actions are made to address these changes regarding restructuring (refactoring) the code [11]. Architecture and design changes are investigated at different granularity levels [12–14]. Software design, code, and evolution patterns can be studied well by gaining a deep insight into the process of software evolution. This work emphasizes the evolution patterns of open-source software systems.

The remainder of this paper is organized as follows. Section 2 presents some related work to our study. Section 3 presents more details about the internal quality of software systems. Section 4 presents the used research methodology. The results and analysis are given in Section 5. Discussion is presented in Section 6. Conclusions of the research are presented in Section 7.

2. Related Work

In this section, we will be discussing some related work to our study. Paulson et al. [15] proposed the use of McCabe cyclomatic complexity to measure software complexity. Israeli and Feitelson [16] followed the complexity of 810 releases of the Linux kernel over 14 years, with the Mc-Cabe complexity metric to characterize the system evolution. They observed a linear growth trend in the evolution of the Linux kernel with regards to size and coupling.

One of the earliest studies on Open Source Systems (OSS) was undertaken by Godfrey and Tu [17]. The authors examined the evolution of the Linux kernel between 1994 and 1999 and observed that the size of the Linux kernel increased at a linear rate. One study [18] investigated 12 open-source projects based on size, the number of modules, and developers and it was observed that the growth of the projects became more stable over time, with the growth of the projects at some point at the top. Antoniol et al. [19] have studied the stability of Mozilla, Alice, and Eclipse projects. They observed that these systems evolve towards a higher degree of stability, but occasionally instability is needed to adapt to changes and trends. Gonzalez et al. [20] studied the Debian GNU/Linux and found that the size has doubled every two years. Herraiz et al. [21] analyzed the evolution of the 3821 Libre project from SourceForge from Line of Code (LOC), the number of changes, and the number of files. Capiluppi et al. [22] analyzed growth-related metrics on 64 releases of OSS and found that the LOC, the number of functions, and the number of file metrics increased over time. The authors also observed that the number of developers contributing to the project also increased over time.

Thomas et al. [23] assessed the increase of the size of software and coupling for Linux and observed that Linux kernel a linear growth is exhibited by it in size and coupling. Moreover, Gonzalez-Barahona et al. [24] studied one large open-source system (glib) and found linear growth with regards to the size. Chatzimparmpas et al. [25] studied the evolution of 7500 releases of JavaScript applications and found constant change and growth.

Most of the conducted research focuses on internal quality and few studies in the literature focus on open-source software, as well as the relationship between internal and external quality. However, empirical studies on this relationship may provide a way for developers or application stores to improve overall software quality and offer changes in the software development process. Stamelos et al. [26] found a positive relationship between testability and simplicity as internal quality attributes and user satisfaction as external quality. Meirelles [27] analyzed the source code of a large set of open-source projects to study the relationship between source code metrics and the attractiveness of the project. Goemnene and Mens [28] analyzed the relationship between the internal quality and the external quality dimensions such as user satisfaction, popularity, developer activity using Netbeans, Eclipse, and ArgoUML open-source projects. This study focuses on the relationship between software evolution in open-source software systems and its relationship with internal qualities.

3. Internal Quality of Software Systems

Software quality is entangled with software evolution. The ISO/IEC 25010 Quality Model (2008) sets standards to guide the development of software products by identifying and evaluating quality requirements. Software quality is seen from three different perspectives: internal and external quality and the quality in use. Internal software quality is the degree to which a set of static attributes of a software product satisfies stated and implied needs for that software product. External software quality is the degree to which a software product enables the behavior of a system to meet expressed and implied requirements for that system. At the same time, quality in use is the degree to which specific users can use a product or system to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk, and satisfaction in specific contexts of use. Xie et al. [29] stated in their empirical study that the quality of the software system must be evaluated from the internal and external quality perspectives.

Attributes of software quality have been divided into two major categories by the experts, internal and external [30]. External quality attributes are defined as those factors that represent the quality, which cannot be gauged through the knowledge of the software artifacts [31]. On the other hand, internal software quality attributes are defined as those attributes that can be gauged through the knowledge of software artifacts. Another critical difference between these two categories is that it is straightforward to measure internal attributes as compared to external attributes. Examples of internal software quality attributes include inheritance, complexity, coupling, size, and cohesion, which can be quantified after the development of the system. Internal quality attributes are also related to the external attributes before the release of software [32]. In addition to this, the internal software quality has a significant relationship with the software structure, which is not the case with external software quality. It must also be noted that the external quality of the software is more concerned with software behavior when it is used. End-user is not able to see the software structure but still has a major association with it as internal quality attributes have a major influence on the external quality attributes [33].

Internal quality attributes are a critical measure of code structural quality [31]. Various other indicators can be gauged through internal quality attributes, which include the size that quantifies the length of a software system. In addition, complexity is another measure that can be gauged through internal attributes, and it defines the overload of decisions and responsibilities of software. Moreover, coupling and cohesion can be measured using internal attributes. Coupling that defines the degree of interdependence between modules and classes can be measured using inheritance which is an internal quality attribute. On the contrary, cohesion is the extent to which internal module elements are interrelated.

Since we are using software metrics in our study, we will be discussing them in more detail. There are four main categories of software metrics [34]. This classification is based on what they measure and what area of software development they focus on. At a very high level, software metrics can be classified as process metrics, project metrics, product metrics, and personnel metrics.

Process metrics coincide with the software development process, encompassing the standards, activities, and methods used. They usually leverage historical project knowledge and assess the capability of the software engineering process. Examples of process metrics are prior defects, prior commits, and metrics for measuring project progress. For example, the efficiency of fault detection.

Project metrics indicate how the project is planned and executed. The intent of these metrics is two-fold: minimize development schedule and assess product quality. Examples include the number of developers, the effort allocated per phase, and the amount of design reuse achieved by the project.

Product metrics measure characteristics of the result of a software development process. Product metrics are typically calculated from the source code. Product metrics refer to different features of the product such as design features, size, complexity, and performance. They provide software engineers with a guide to analyze, design, code, and test software more objectively. Examples include the complexity of the design, the size of the source code, and the usability of the produced documentation.

Personnel metrics indicate the productivity and quality for each of the project team members. Examples of personnel metrics are programming experience, communication level, and workload. For example, ranking developers based on their programming experience (low, moderate, high).

Our focus here in this study is product metrics. Product metrics can be categorized as internal and external attributes. Internal product attributes measure the product itself. Internal attributes are concerned with size, complexity, coupling, and cohesion. External attributes are concerned with product quality, such as usability, testability, reusability, and portability [35]. As external attributes are directly observable only after the system has already been deployed and operational for some time, the focus has been on relating internal attributes (software metrics) to their external qualities. Product metrics have come to be important in a subset of software engineering disciplines, software quality, and evolution. They are used to estimate the effort and cost of software projects and to measure software quality [35]. In software evolution, product metrics are used for recognizing the stability of software system entities, along with recognizing where refactorings can be or have been practiced and identifying the variation of quality in the evolving software systems structure. In reverse engineering, product metrics are used for evaluating the complexity and quality of software products.

Several attempts were made to measure software complexity, such as [36,37] and measures of size, such as [38], which were expected to be programming language independent. Representative Examples of complexity-based code-level metrics are Halstead Volume [36], a metric based on operator and operand counts (data flow), and McCabe Complexity [37], a metric based on the number of possible paths in the program control graph. McCabe's cyclomatic complexity and its variations capture different flavors of code complexity.

The DeLone and McLean model of information systems offers an opportunity to measure the success of open-source software systems, specifically by explaining the relationship between all six (6) dimensions of information systems [39]. The updated version of the DeLone and McLean model demonstrates that these dimensions apply to open-source software systems. One of the prominent dimensions of the DeLone and McLean model is 'code quality' that incorporate various attributes such as Maintainability, Efficiency, and Understandability [40]. It has been established that the dimension of the DeLone and McLean model named 'System Quality' can be used for measuring several metrics on software's 'code quality' such as Maintainability, Efficiency, Effectiveness, and Understandability [41]. The present research has carried out the analysis of different quality attributes, including cohesion, coupling, inheritance, complexity, and size.

4. Research Methodology

Software quality can be easily analyzed through in-depth software evaluation. The current research evaluates the open-source software system for assessing the software quality and to answer following research questions:

- RQ1: How does open-source software evolution affect the Line of Code (LOC)?
- RQ2: Is there a statistically significant relationship between size and other source code attributes (e.g., complexity, coupling, inheritance, and cohesion)?

To answer the first research question, a statistical test named 'Analysis of Variance' (ANOVA) is conducted. ANOVA signifies the set of statistical models and their related estimation procedures that are used for analyzing the differences among means. In particular, it is the statistical method of finding out if the results of the experiment are significant. Moreover, it also assists in identifying if two or more means are equal; hence, generalizing the t-test beyond two means. It indicates that ANOVA is used for determining the equivalence of sample means, for three or more populations. It has been established that the variances, occurred in data, are mainly due to two reasons, i.e., 'just by chance' or 'specific'. In this regard, ANOVA assists in analyzing if the cause of variance is 'just by chance' or 'specific'. Likewise, the present study has used ANOVA analysis to examine if the difference in LOC between the systems is significant.

Moreover, post-hoc analysis is also conducted using the Least Significant Difference (LSD) method to carry out multiple comparisons and defining where exactly the difference occurred. In particular, the LSD method—proposed by Fisher—is adopted for creating confidence intervals for all pairwise differences between factor level means, drawn from ANOVA. During this process, the individual error rate is controlled to the specified significance level. Afterwards, the LSD method of Fisher used the number of comparisons and individual error rate for calculating the simultaneous confidence level. It is important to note that the simultaneous confidence level indicates that all confidence intervals possess the true difference. Moreover, it is also worth noting that the family error rate must be taken into consideration, specifically while conducting multiple comparisons. It is mainly due to the fact that the possibility of occurrence of Type I error is higher during a series of comparisons than the rate of error that is associated with single comparisons.

Another way of analyzing the means is to statistically model them rather than simply describing them as they appear in the data. Marginal means are means that are extracted from a statistical model and represent the average response variable for each level of the predictor variable.

In a multiple regression model, multicollinearity can be understood as the phenomenon in which one predictor can be linearly predicted from the others with higher accuracy levels. In other words, multicollinearity is the extent to which any variable effect can be predicted by other variables. It has been established that multicollinearity takes place when the model includes multiple factors, with a strong correlation. Afterward, changes in the other variables are also shown as a function of the version and Java systems and it is usually represented in the form of a graph.

To answer the second research questions, correlation analysis and regression analysis are conducted. The rationale of conducting correlation analysis is that it helps in studying the variation between two or more independent variables that eventually assists in identifying the extent of correlation between them. On the other hand, regression analysis allows the analysis of the relationship between independent and dependent variables. In particular, it allows the assessment of the strength of the relationship between the variables that is later used for the modelling of the relationship that exists between the variables.

4.1. Data Sets

Five open-source software systems were selected based on particular selection criteria. In particular, the criteria required that the system must be written in Java, well-maintained, under maintenance for at least 5 years, large enough, and widely used. A total of ten releases of these systems were collected with two releases per year (around 6 months). Moreover, the code of all the 5 systems was retrieved from their code repository. For extracting metrics, the "o3smeasures" tool was used to collect different metrics about the selected systems. This tool has been used previously in different empirical studies. The selected systems are:

- Lucene is an Apache search engine software library. The collected versions are from 5.3.2 to 8.6.0. The archives can be downloaded from [42]

- JabRef is a cross-platform citation and reference management software. The collected versions are from 3.0 to 5.2. The archives can be downloaded from [43]
- PMD is a cross-language static code analyzer. The collected versions are from 5.3.7 to 6.26.0. The archives can be downloaded from [44]
- Struts is a web-based application framework that is used for the designing of Java EE web applications. The collected versions are from 2.3.25 to 2.5.26. The archives can be downloaded from [45]
- Commons Lang is a Java utility package for java.lang API. The collected versions are from 3.3.1 to 3.12. The archives can be downloaded from [46]

The tool used in this study is an Eclipse plugin that can be downloaded from [47]. The tool can calculate different internal quality attributes. Table 1 shows the details of the metrics that are used in this study. The selected metrics are found to be frequently used in the literature [48]. They have been extensively validated and used in previous research. Moreover, they also capture three important aspects of software quality, including, size, complexity, inheritance, coupling, and cohesion.

Table 1. Software Internal Quality Metrics and Attributes.

Attribute	Metric	Source
Size	Number of Classes	[49]
	Line of code (LOC)	[49]
Complexity	Cyclomatic Complexity (CC) demonstrates different number of paths that are present in a method plus one.	[37]
Inheritance	Depth of Inheritance Tree (DIT) denotes the measure of a total number of ancestors present in a class.	[50]
Coupling	Coupling Between Objects (CBO) signifies the number of classes that are coupled to a particular class.	[50]
Cohesion	Lack of Cohesion of Methods (LCOM) can be understood as the total number of pairs, present in the member functions, without any shared instance variables, minus the number of member functions' pairs with shared instance variables.	[50]

5. Results and Analysis

This section presents the results and analysis of the experimental evaluation. To answer RQ1, we will assess the pattern of evolution in size (LOC). In this account, change can be plotted in each variable, across the versions, to observe how software systems evolve for each Java system. Figure 1 depicts the change in the LOC as a function of the version and Java system. It seems that Lucene and Jabref outperform other systems with respect to LOC. Moreover, ANOVA analysis is also conducted to examine if the difference in LOC between the systems is significant.

It is important to note that one-way ANOVA is used when there are three or more independent samples which is also the case in the current research work. Table 2 shows the ANOVA analysis results. dependent variable is LOC, R Squared is 0.975, and the adjusted R Squared is 0.973. The results of one-way ANOVA analysis shows that there is a significant difference between Java systems with respect to the LOC variable. Therefore, post-hoc analysis is conducted with the LSD method for multiple comparisons, to define where exactly the difference occurred. The post-hoc analysis showed based on observed means that the error term is Mean Square(Error) = 31,274,123.587 and the mean difference is significant at the .05 level. The results indicate that only the mean difference between the Strtus and PMD; Strtus and Common language; and PMD and common language are not significant. All other pairwise comparisons appear to be significant. Figure 2a shows the estimated marginal means of LOC in different systems and Figure 2b shows the estimated marginal means of LOC in different systems over versions.

Figure 3 shows the change of different internal quality attributes over versions. It is very clear from these figures that internal quality attributes are growing worse overtime. This is in line with different theories and laws of software evolution. Unless there is a serious investment in improving the internal qualities, they will grow worse over different versions.

Table 2. ANOVA Analysis Tests of Between-Subjects Effects.

Source	DF	Mean Square	F	Partial Eta Squared
Corrected Model	4	13,923,329,927.650	445.203	0.975
Intercept	1	136,464,462,738.000	4363.494	0.990
Systems	4	13,923,329,927.650	445.203	0.975

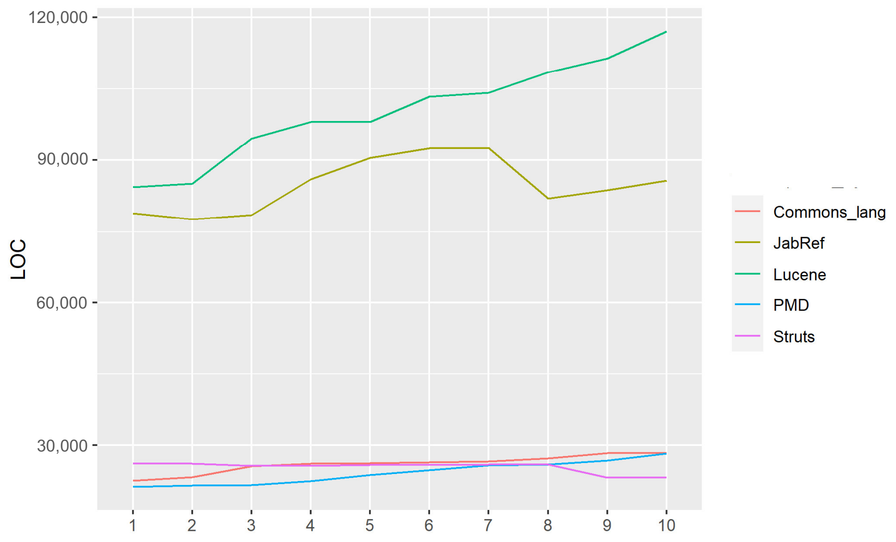
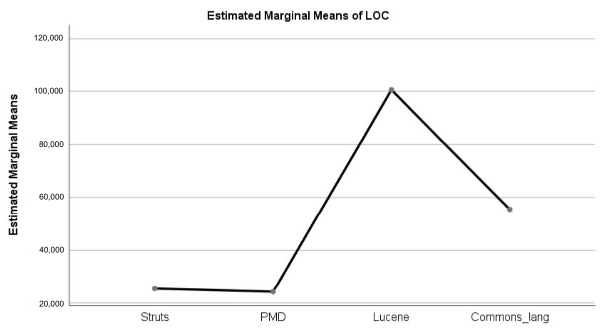
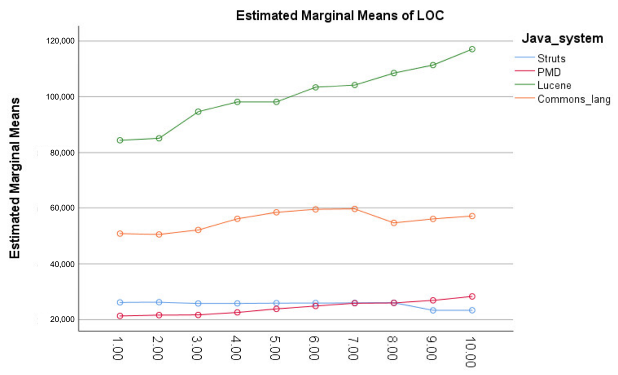


Figure 1. The change in the LOC as a function of the version.

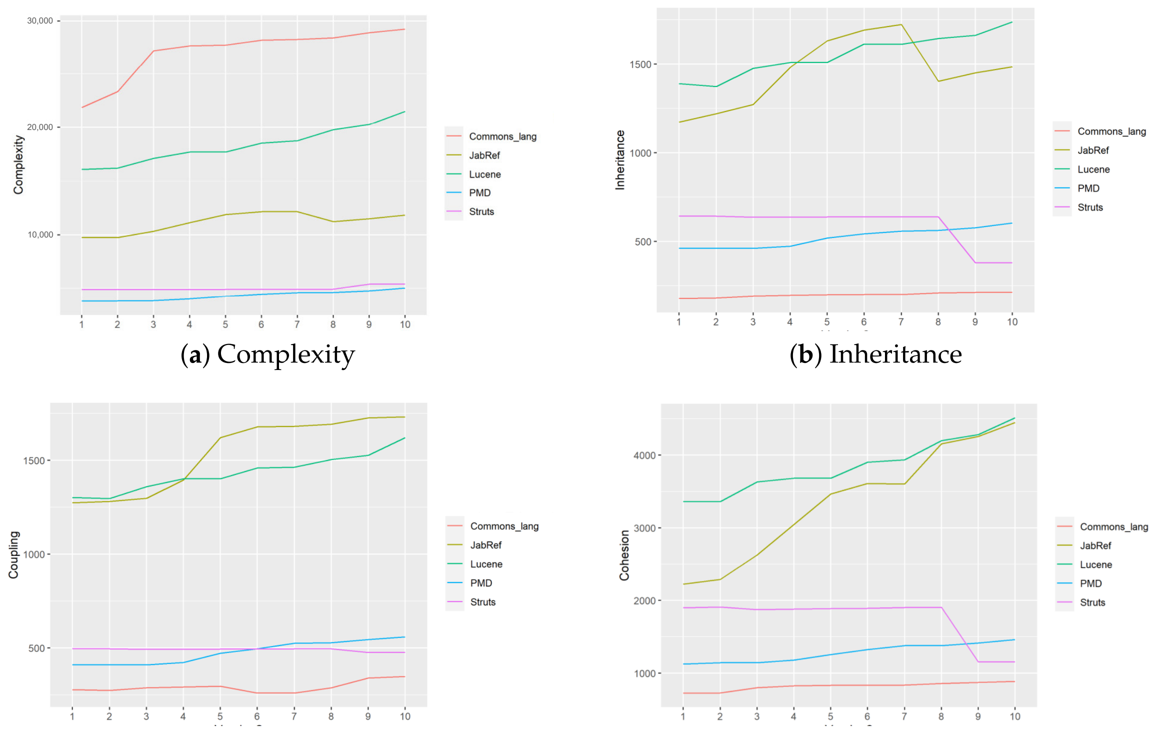


(**a**)



(**b**)

Figure 2. Estimated Marginal Means of LOC.



(**a**) Complexity (**b**) Inheritance

(**c**) Coupling (**d**) Cohesion

Figure 3. Change of Internal Quality Attributes over Versions.

5.1. Correlation Analysis

The results, drawn from correlation analysis are shown in Table 3. According to the literature [51], correlations can be considered high if it is larger than 0.8, moderate if it is in between 0.5 and 0.8, and low if it is below 0.5. The correlation between all pairwise comparisons appears to be significant. The correlation of the LOC variable, with other variables, indicate that the LOC variable has a significant and positive relationship with inheritance coupling, cohesion, and the number of classes ranging from 0.934 to 0.978. However, the correlation between LOC and complexity is significant but negligibly small. The correlation statistics above .90 indicates that there is multicollinearity between some of these variables. Therefore, multicollinearity analysis is required to be conducted to detect variables that have multicollinearity with others.

Table 3. The Results of Correlation Analysis.

	# Classes	**LOC**	**Complexity**	**Inheritance**	**Coupling**	**Cohesion**
# Classes	1	0.978	0.042	0.988	0.98	0.974
LOC		1	0.215	0.957	0.959	0.934
Complexity			1	−0.049	0.022	0.004
Inheritance				1	0.977	0.968
Coupling					1	0.958
Cohesion						1

5.2. Regression Analysis

The regression analysis is carried out for examining the impact of each independent variable on the LOC variables which is defined as the dependent variable. In other words, the regression analysis helps in recognizing the impact of the variable on the dependent variables as well as on the relationship between the dependent and independent variables since regression coefficients can be interpreted as correlation coefficients.

The regression analysis results of the first model that contains 5 independent variables are shown in Table 4. The dependent variable is LOC. The Variance Inflation Factor (VIF) values for the metrics were calculated. If the VIF values are higher than 10, multicollinearity is strongly suggested [52]. All variables appear to make a significant contribution to the model except for the coupling variable. However, VIF (above 10) and Collinearity statistics indicate that there is multicollinearity between some of the independent variables. The results of multicollinearity analyses indicate the existence of multicollinearity between the number of classes variables and other variables. Therefore, this variable was excluded from the model.

Table 4. Regression Analysis.

	Unstandardized Coefficients		**Standardized Coefficients**			**Collinearity Statistics**	
Model	**B**	**Std. Error**	**Beta**	**t**	**Sig.**	**Tolerance**	**VIF**
Constant	−9078.904	1171.597		−7.749	0.000		
# Classes	54.919	8.764	0.709	6.266	0.000	0.013	79.879
Complexity	0.814	0.061	0.210	13.370	0.000	0.652	1.533
Inheritance	30.131	6.433	0.491	4.684	0.000	0.015	68.618
Coupling	1.323	4.219	0.021	0.314	0.755	0.036	27.999
Cohesion	−6.802	1.523	−0.253	−4.466	0.000	0.050	20.000

The regression results of the model that can be used to predict LOC variable is given in Table 5. The results of the regression analysis shown that the effect of complexity and inheritance on the LOC was positive and significant. However, the effect of coupling and cohesion was not significant. Additionally, only the cohesion variable had a negative effect on the LOC, but this effect was not significant. Moreover, the standardized regression coefficients indicate that inheritance had the largest effect on the LOC, followed by complexity.

Table 5. Multicollinearity Analysis.

	Unstandardized Coefficients		**Standardized Coefficients**		
Model	**B**	**Std. Error**	**Beta**	**t**	**Sig.**
Constant	−11560.023	1499.943		−7.707	0.000
Complexity	1.004	0.072	0.259	14.015	0.000
Inheritance	58.155	6.290	0.949	9.245	0.000
Coupling	8.794	5.505	0.140	1.597	0.117
Cohesion	−3.192	1.918	−0.119	−1.664	0.103

6. Discussion

The first research was answered in this study since we have seen that there is a continuous increase in the LOC in all systems. This is expected since more features are usually added to these systems. The second research question looked in more detail about the relationship between these internal qualities (product metrics). The focus is size and how it relates to other attributes. We found that size strongly correlates with coupling, cohesion, and inheritance. However, it does not correlate with complexity. The results of the regression analysis showed complexity and inheritance have a positive and significant impact on size. However, the effect of coupling and cohesion was not significant.

The size and complexity increase revealed in our study show evidence that software engineers should take proactive action to prevent software decay and avoid producing software that is difficult, if not impossible, to repair and evolve. Software engineers can prevent the trend of ever-increasing code complexity by continuously monitoring code complexity and taking proactive steps to reduce evolution costs. Our study can help software managers plan their projects more thoughtfully; since software tends to grow a lot, change a lot, and become more complex by provisioning resources to accommodate growth and by taking aggressive steps to avoid software decay and prevent complexity build-up, managers can stay on time and budget. Finally, we underscore the importance of managers and developers continuously monitoring software quality to keep the software.

7. Conclusions

The current empirical research analyzes open-source software systems' evolution in Java systems. The main goal of the study was to assess how software systems have been evolved in different versions, with regards to size and the relationship between size and different internal quality metrics. In this study, ten versions of 5 open-source Java systems were retrieved to calculate their metrics. Moreover, several statistical tests including ANOVA, correlation analysis, and regression analysis, were used to answer the research questions.

The results and analysis show (i) there is a significant difference between different systems with respect to the LOC variable, (ii) there is a significant correlation between all pairwise comparisons of internal quality metrics, and (iii) the effect of complexity and inheritance on the LOC was positive and significant; however, the effect of coupling and cohesion was not significant. As far as the future direction of the research is concerned, it is intended to define a global model for predicting how the size evolves. In addition to this, future research aspiration also involves the investigation of the evolution of the metrics while considering different systems belonging to other domains.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://malenezi.github.io/malenezi/data/Internal-Quality-Evolution-Java.

Acknowledgments: The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Madhavji, N.H.; Fernandez-Ramil, J.; Perry, D. *Software Evolution and Feedback: Theory and Practice*; John Wiley & Sons: Hoboken, NJ, USA, 2006.
2. Reussner, R.; Goedicke, M.; Hasselbring, W.; Vogel-Heuser, B.; Keim, J.; Märtin, L. *Managed Software Evolution*; Springer: Berlin/Heidelberg, Germany, 2019.
3. Sousa, B.L.; Bigonha, M.A.; Ferreira, K.A. Analysis of Coupling Evolution on Open Source Systems. In Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse, Salvador, Brazil, 23–27 September 2019; pp. 23–32.
4. Mens, T.; Demeyer, S. *Software Evolution*, 1st ed.; Springer: Berlin/Heidelberg, Germany, 2008.
5. Neamtiu, I.; Xie, G.; Chen, J. Towards a better understanding of software evolution: An empirical study on open-source software. *J. Softw. Evol. Process.* **2013**, *25*, 193–218.
6. Alenezi, M.; Khellah, F. Architectural stability evolution in open-source systems. In Proceedings of the The International Conference on Engineering & MIS 2015, Istanbul, Turkey, 24–26 September 2015; pp. 1–5.
7. Alenezi, M.; Almustafa, K. Empirical analysis of the complexity evolution in open-source software systems. *Int. J. Hybrid Inf. Technol.* **2015**, *8*, 257–266.
8. Gezici, B.; Tarhan, A.; Chouseinoglou, O. Internal and external quality in the evolution of mobile software: An exploratory study in open-source market. *Inf. Softw. Technol.* **2019**, *112*, 178–200.

9. Tripathy, P.; Naik, K. *A Practitioner's Approach, Software Evolution and Maintenance*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2014.
10. Chatzigeorgiou, A.; Melas, G. Trends in object-oriented software evolution: Investigating network properties. In Proceedings of the 2012 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2–9 June 2012; pp. 1309–1312.
11. Yu, L.; Mishra, A. An Empirical Study of Lehman's Law on Software Quality Evolution. *Int. J. Softw. Inform.* **2013**, *7*, 469–481.
12. Counsell, S.; Hassoun, Y.; Johnson, R.; Mannock, K.; Mendes, E. Trends in Java code changes: The key to identification of refactorings? In Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java, Kilkenny, Ireland, 16–18 June 2003, pp. 45–48.
13. Bagherzadeh, M.; Kahani, N.; Bezemer, C.P.; Hassan, A.E.; Dingel, J.; Cordy, J.R. Analyzing a decade of Linux system calls. *Empir. Softw. Eng.* **2018**, *23*, 1519–1551.
14. Ivers, J.; Ozkaya, I.; Nord, R.L.; Seifried, C. Next generation automated software evolution refactoring at scale. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, NY, USA, 8–13 November 2020; pp. 1521–1524.
15. Paulson, J.W.; Succi, G.; Eberlein, A. An empirical study of open-source and closed-source software products. *IEEE Trans. Softw. Eng.* **2004**, *30*, 246–256.
16. Israeli, A.; Feitelson, D.G. The Linux kernel as a case study in software evolution. *J. Syst. Softw.* **2010**, *83*, 485–501.
17. Tu, Q.; others. Evolution in open source software: A case study. In Proceedings of the 2000 International Conference on Software Maintenance, San Jose, CA, USA, 11–14 October 2000; pp. 131–142.
18. Capiluppi, A. Models for the evolution of OS projects. In Proceedings of the International Conference on Software Maintenance, Amsterdam, The Netherlands, 22–26 September 2003; pp. 65–74.
19. Antoniol, G.; Gueheneuc, Y.G.; Merlo, E.; Tonella, P. Mining the lexicon used by programmers during sofware evolution. In Proceedings of the 2007 IEEE International Conference on Software Maintenance, Paris, France, 2–5 October 2007; pp. 14–23.
20. Gonzalez-Barahona, J.M.; Robles, G.; Michlmayr, M.; Amor, J.J.; German, D.M. Macro-level software evolution: A case study of a large software compilation. *Empir. Softw. Eng.* **2009**, *14*, 262–285.
21. Herraiz, I. A statistical examination of the evolution and properties of libre software. In Proceedings of the 2009 IEEE International Conference on Software Maintenance, Edmonton, AB, Canada, 20–26 September 2009; pp. 439–442.
22. Capiluppi, A.; Ramil, J.F. Studying the evolution of open source systems at different levels of granularity: Two case studies. In Proceedings of the 7th International Workshop on Principles of Software Evolution, Kyoto, Japan, 6–7 September 2004; pp. 113–118.
23. Thomas, L.; Schach, S.R.; Heller, G.Z.; Offutt, J. Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux. *IET Softw.* **2009**, *3*, 58–66.
24. Gonzalez-Barahona, J.M.; Robles, G.; Herraiz, I.; Ortega, F. Studying the laws of software evolution in a long-lived FLOSS project. *J. Softw. Evol. Process* **2014**, *26*, 589–612.
25. Chatzimparmpas, A.; Bibi, S.; Zozas, I.; Kerren, A. Analyzing the Evolution of Javascript Applications. In Proceedings of the ENASE 2019 14th International Conference on Evaluation of Novel Approaches to Software Engineering, Crete, Greece, 4–5 May 2019; pp. 359–366.
26. Stamelos, I.; Angelis, L.; Oikonomou, A.; Bleris, G.L. Code quality analysis in open source software development. *Inf. Syst. J.* **2002**, *12*, 43–60.
27. Meirelles, P.; Santos, C., Jr.; Miranda, J.; Kon, F.; Terceiro, A.; Chavez, C. A study of the relationships between source code metrics and attractiveness in free software projects. In Proceedings of the 2010 Brazilian Symposium on Software Engineering, Salvador, Brazil, 27 September–1 October 2010; pp. 11–20.
28. Goeminne, M.; Mens, T. Towards the analysis of evolution OSS ecosystems. In Proceedings of the 8th BElgian-NEtherlands Software eVOLution Seminar (BENEVOL 2009), Louvain-la-Neuve, Belgium, 17–18 December 2009; pp. 30–35.
29. Xie, G.; Chen, J.; Neamtiu, I. Towards a better understanding of software evolution: An empirical study on open source software. In Proceedings of the 2009 IEEE International Conference on Software Maintenance, Edmonton, AB, Canada, 20–26 September 2009; pp. 51–60.
30. Morasca, S. A probability-based approach for measuring external attributes of software artifacts. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, FL, USA, 15–16 October 2009; pp. 44–55.
31. Fenton, N.; Bieman, J. *Software Metrics: A Rigorous and Practical Approach*; CRC Press: Boca Raton, FL, USA, 2014.
32. Lee, Y.; Chang, K.H. Reusability and maintainability metrics for object-oriented software. In Proceedings of the 38th Annual on Southeast Regional Conference, Clemson, SC, USA, 7–8 April 2000; pp. 88–94.
33. Briand, L.C.; Morasca, S.; Basili, V.R. Property-based software engineering measurement. *IEEE Trans. Softw. Eng.* **1996**, *22*, 68–86.
34. Abran, A. *Software Metrics and Software Metrology*; John Wiley & Sons: Hoboken, NJ, USA, 2010.
35. Fenton, N.E.; Neil, M. Software Metrics: Roadmap. In Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 4–11 June 2000; pp. 357–370.
36. Halstead, M.H. Elements of Software Science. 1977. Available online: https://dl.acm.org/doi/book/10.5555/540137 (accessed on 1 May 2021).
37. McCabe, A. Complexity Measure. *IEEE Trans. Softw. Eng.* **1976**, *4*, 308–320.

38. Albrecht, A.J. Measuring Application Development Productivity. In Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, Monterey, CA, USA, 14–17 October 1979.
39. DeLone, W.H.; McLean, E.R. Information systems success: The quest for the dependent variable. *Inf. Syst. Res.* **1992**, *3*, 60–95.
40. Gorton, I.; Liu, A. Software component quality assessment in practice: Successes and practical impediments. In Proceedings of the 24th International Conference on Software Engineering, Orlando, FL, USA, 23–25 May 2002; pp. 555–558.
41. Abreu, R.; Premraj, R. How developer communication frequency relates to bug introducing changes. In Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, Amsterdam, The Netherlands, 24–25 August 2009; pp. 153–158.
42. Lucene. Lucene Repository. Available online: https://archive.apache.org/dist/lucene/java/ (accessed on 1 May 2021).
43. JabRef. JabRef Repository. Available online: https://github.com/JabRef/jabref/releases (accessed on 1 May 2021).
44. PMD. PMD Repository. Available online: https://github.com/pmd/pmd/releases (accessed on 1 May 2021).
45. Struts. Struts Repository. Available online: https://github.com/apache/struts/releases (accessed on 1 May 2021).
46. Lang, C. Common Lang Repository. Available online: https://github.com/apache/commons-lang/releases (accessed on 1 May 2021).
47. Azevedo, M. o3smeasures-tool. Available online: https://github.com/mariazevedo88/o3smeasures-tool/ (accessed on 1 May 2021).
48. Nuñez-Varela, A.S.; Pérez-Gonzalez, H.G.; Martínez-Perez, F.E.; Soubervielle-Montalvo, C. Source code metrics: A systematic mapping study. *J. Syst. Softw.* **2017**, *128*, 164–197.
49. Lanza, M.; Marinescu, R. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*; Springer: Berlin/Heidelberg, Germany, 2007.
50. Chidamber, S.R.; Kemerer, C.F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.* **1994**, *20*, 476–493.
51. Succi, G.; Pedrycz, W.; Djokic, S.; Zuliani, P.; Russo, B. An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. *Empir. Softw. Eng.* **2005**, *10*, 81–104.
52. Fenton, N.E.; Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Eng.* **2000**, *26*, 797–814.