Metrics and Software Quality Evolution: A Case Study on Open Source Software

Nicholas Drouin, Mourad Badri, and Fadel Touré

***Abstract*—This paper aims at analyzing empirically the quality evolution of an open source software using metrics. We used a control flow based metric (*Quality Assurance Indicator* - Qi) which we proposed in a previous work. We wanted to investigate if the Qi metric can be used to observe how quality evolves along the evolution of the successive released versions of the subject software system. We addressed software quality from an internal perspective. We performed an empirical analysis using historical data on the subject system (Apache Tomcat). The collected data cover, in fact, a period of more than seven years (thirty-one versions in total). Empirical results provide evidence that the Qi metric reflects properly the quality evolution of the subject system.**

***Index Terms*—Empirical analysis, metrics, software attributes, software evolution, software quality.**

I. Introduction

Software evolution is the dynamic behavior of programming systems as they are maintained and enhanced over their lifetimes [1]. Software systems need to continually evolve during their life cycle for various reasons: adding new features to satisfy user requirements, changing business needs, introducing novel technologies, correcting faults, improving quality, etc. [2], [3]. The accumulation of changes, along the evolution of a software system, can lead to a degradation of its quality [4] – [8]. It is, therefore, important to monitor how software quality evolves so that quality assurance (QA) activities can be properly planned [8]. Quality plays, indeed, an important role in a software project's success.

Software metrics can be used to analyze the evolution of software systems quality [9]. Metrics have, in fact, a number of interesting characteristics for providing evolution support [10]. A large number of metrics have been proposed for measuring various properties of object-oriented (OO) software systems [11]. Empirical evidence exist showing that there exist a relationship between (many of) these metrics and software quality [9], [12] – [21]. However, with the growing complexity and size of OO software systems, the ability to reason about such a major issue using synthetic metrics would be more appropriate in practice.

We proposed in [22] a new metric, called Quality Assurance Indicator (Qi), capturing in an integrated way different attributes of OO software systems such as coupling (*interactions between classes*) and complexity (*distribution of the control flow in a system*). The Quality Assurance Indicator of a class is based on intrinsic characteristics of the class, as well as on the Quality Assurance Indicator of its collaborating classes. The metric has, however, no ambition to capture the overall quality of OO software systems. Moreover, the objective is not to evaluate a design by giving absolute values, but more relative values that may be used for identifying critical classes on which more QA effort is needed to ensure software quality. We explored in [12] the relationship between the Qi metric and testability of classes. Testability was measured (inversely) by the number of lines of test code (JUnit test suites) and the number of *assert* statements in the test code. More recently, we investigated the capacity of the Qi metric in predicting the unit testing effort of classes using regression analysis techniques [23].

In this paper, we wanted to investigate if the Qi metric reflects (captures) properly the evolution of software quality when software is actively maintained and updated. We focus on retrospective analysis of software quality. We consider software quality from an internal (structural) perspective. We used two well-known OO design metrics, CBO (*Coupling Between Objects*) and WMC (*Weighted Methods per Class*) [24], for measuring the internal quality of a release (particularly coupling and complexity, two important attributes). The CBO metric counts for a class the number of other classes to which it is coupled (and vice versa). The WMC metric gives the sum of complexities of the methods of a given class, where each method is weighted by its cyclomatic complexity. These metrics have received considerable attention from researchers and are also being increasingly adopted by practitioners. Furthermore, these metrics have been incorporated into several development tools. We performed an empirical analysis using historical data on an open source software system (Apache Tomcat). The collected data cover a period of more than seven years. Empirical results provide evidence that the Qi metric reflects properly the quality (as captured by the selected OO metrics) evolution of the subject software system.

The rest of this paper is organized as follows: Section 2 gives a survey on related work. The Qi metric is introduced in Section 3. Section 4 presents the empirical study we performed. Finally, Section 5 concludes the paper.

II. Related Work

Mens et al. [10] provide an overview of the ways software metrics have been (and can be) used to analyze software evolution. A distinction is made between using software metrics before the evolution has occurred (*predictive*) and

Manuscript received September 20, 2012; revised December 17, 2012. This project was financially supported by NSERC (National Sciences and Engineering Research Council of Canada) and FRQNT (Fonds de Recherche du Québec – Nature et Technologies) grants.

The authors are with the Software Engineering Research Laboratory, Department of Mathematics and Computer Science, University of Quebec, Trois-Rivières, Québec, Canada (e-mails: nicholas.drouin@uqtr.ca, mourad.badri@uqtr.ca, fadel.toure@uqtr.ca).

DOI: 10.7763/IJCTE.2013.V5.742

after the evolution has occurred (*retrospective*). To support retrospective analysis, metrics can be used to understand the quality evolution of a software system by considering its successive releases. In particular, metrics can be used to measure whether the quality of a software has improved or degraded between two releases. Dagpinar et al. [15] investigate the significance of different OO metrics for the purpose of predicting maintainability of software. Nagappan et al. [25] focus on mining metrics to predict component failures. The authors noted that there is no a single set of complexity metrics that could be used as a universally best defect predictor. Ambu et al. [26] address the evolution of quality metrics in an agile/distributed project and investigate how the distribution of the development team has impacted the code quality.

Lee et al. [9] provide an overview of open source software evolution with software metrics. The authors explored the evolution of an open source software system in terms of size, coupling and cohesion, and discuss its quality change based on the Lehman's laws of evolution [4], [5], [27]. Jermakovics et al. [28] propose an approach to visually identify software evolution patterns related to requirements. Mens et al. [29] present a metrics-based study of the evolution of Eclipse. The authors consider seven major releases and investigate whether three of the laws of software evolution (*continuing change*, *increasing complexity* and *continuing growth*) were supported by the data collected. Xie et al. [3] conduct an empirical analysis on the evolution of seven open source programs and investigate also Lehman's evolution laws. Murgia et al. [18] address software quality evolution in open source projects using agile practices. The authors used a set of OO metrics to study software evolution and its relationship with bug distribution. According to the achieved results, Murgia et al. concluded that there is no a single metric that is able to explain the bug distribution during the evolution of the analyzed systems. Zhang et al. [8] use c-charts and patterns to monitor quality evolution over a long period of time. The number of defects was used as a quality indicator. Eski et al. [16] present an empirical study on the relationship between OO metrics and changes in software. The authors analyze modifications in software across the historical sequence of open source projects and propose a metrics-based approach to predict change-prone classes. Yu et al. [30] study the possibility of using the number of bug reports as a software quality measure. Using statistical methods, the authors analyze the correlation between the number of bug reports and software changes.

III. Quality Assurance Indicator

We give, in this section, a summary of the definition of the Quality Assurance Indicator (Qi) metric. For more details see [12], [22], [23]. The Qi metric is based on *Control Call Graphs* (CCG), which are a reduced form of traditional *Control Flow Graphs* (CFG). A CCG is a CFG from which the nodes representing instructions (or basic blocs of sequential instructions) not containing a call to a method are removed. Compared to traditional *Call Graphs*, CCG are much more precise models. They capture the structure of calls and related control. The Qi metric is normalized and gives values in the interval [0, 1]. A low value of the Qi of a class means that the class is a high-risk class and needs a (relative) high QA effort to ensure its quality. A high value of the Qi of a class indicates that the class is a low-risk class (having a relatively low complexity and/or the QA effort applied actually on the class is relatively high - proportional to its complexity).

A. Quality Assurance Indicator

The Qi of a method M_i is defined as a kind of estimation of the probability that the control flow will go through the method without any failure. The Qi of a method M_i is based on intrinsic characteristics of the method (*cyclomatic complexity*, *unit testing coverage*), as well as on the Qi of the methods invoked by the method M_i. There is a kind of propagation, depending on the distribution of the control flow in a system, which needs to be captured. The Qi of a method M_i is given by:

$$Qi_{M_i} = Qi^{*}_{M_i} \cdot \sum_{j=1}^{n_i} \left[P(C_j^i) \cdot \prod_{M \in \sigma_j} Qi_M \right] \quad (1)$$

with :

Qi_{M_i}: QA indicator of method M_i,

$Qi^{*}_{M_i}$: intrinsic QA indicator of method M_i,

C_j^i: j^{th} path of method M_i,

$P(C_j^i)$: probability of execution of path C_j^i of method M_i,

Qi_M: QA indicator of method M included in the path C_j^i,

n_i: number of linear paths of the CCG of method M_i, and

σ_j: set of the methods invoked in the path C_j^i.

By applying the previous formula (1) to each method we obtain a system of N (number of methods in the program) equations. The obtained system is not linear and is composed of several multivariate polynomials. We use an iterative method (method of successive approximations) to solve it. The system is, in fact, reduced to a fixed point problem. Furthermore, we define the Qi of a class C (noted Qi_C) as the product of the Qi of its methods:

$$Qi_C = \prod_{M \in \delta} Qi_M \quad (2)$$

where δ is the set of methods of the class C. The calculation of the Qi metric is entirely automated by a tool (prototype) that we developed for Java software systems.

B. Assigning Probabilities

The CCG of a method can be seen as a set of paths that the control flow can pass through (depending on the states of the conditions in the control structures). To capture this probabilistic characteristic of the control flow, we assign a probability to each path C of a control call graph as follows:

$$P(C) = \prod_{A \in \theta} P(A) \quad (3)$$

where θ is the set of directed arcs composing the path C and $P(A)$ the probability of an arc to be crossed when exiting a control structure.

To facilitate our experiments (simplify analysis and calculations), we assigned probabilities to the different control structures of a (Java) program according to the rules given in Table I. These values are assigned automatically during the static analysis of the source code of a program when generating the Qi models. These values can be adapted according to the nature of the applications (for example). As an alternative way, the probability values may also be assigned by programmers during the development (knowing

the code) or obtained by dynamic analysis. Dynamic analysis is out of the scope of this paper.

TABLE I: ASSIGNMENT RULES OF THE PROBABILITIES.

Nodes	Probability Assignment Rule
(if, else)	0.5 for the exiting arc «condition = true » 0.5 for the exiting arc «condition=false »
while	0.75 for the exiting arc «condition = true » 0.25 for the exiting arc «condition = false »
(do, while)	1 for the arc: (the internal instructions are executed at least once)
(switch,case)	1/n for each arc of the n cases.
(?, :)	0.5 for the exiting arc «condition = true » 0.5 for the exiting arc «condition = false »
for	0.75 for entering the loop 0.25 for skipping the loop
(try, catch)	0.75 for the arc of the «try »bloc 0.25 for the arc of the «catch »bloc
Polymorphism	1/n for each of the eventual n calls.

C. Intrinsic Quality Assurance Indicator

The *Intrinsic Quality Assurance Indicator* of a method M_i, noted Qi^*_{Mi}, is given by:

$$Qi^*_{M_i} = (1 - F_i) \quad (4)$$

with: $$F_i = \frac{(1-tc_i)cc_i}{cc_{max}}$$

where:

CC_i : cyclomatic complexity of method M_i,

$cc_{max} = \max_{1 \le i \le N}(cc_i)$,

tc_i: unit testing coverage of the method M_i, $tc_i \in [0,1]$.

Many studies provided empirical evidence that there is a significant relationship between cyclomatic complexity and fault proneness (e.g., [13], [21], [31]). Testing (as one of the most important QA) activities will reduce the risk of a complex program and achieve its quality. Moreover, testing coverage provide objective measures on the effectiveness of a testing process. The testing coverage measures are (currently in our approach) affected by programmers based on the test suites they performed on the classes of the system. The testing coverage measures can also be obtained automatically (using tools such as Together (www.borland.com) or CodePro (developers.google.com)) by analyzing the code of the test suites (JUnit suites for example) to determine which parts of the classes that are covered by the test suites and those that are not (this issue will be considered in our future work).

IV. EMPIRICAL STUDY

We present, in this section, the empirical study we conducted in order to investigate if the Qi metric captures the evolution of software quality when software is actively maintained and updated. We address software quality from an internal perspective. We used CBO and WMC metrics. We used historical data collected from successive released versions of an open source software system. We selected Tomcat, which is an open source web server developed by Apache Software Foundation. We analysed its 5.5 branch, launched in august 2004. The version 5.5.35, the latest to date, was launched on November 2011. We used the official releases as time captures of this system. The collected data cover a period of more than seven years (thirty-one versions). Table II gives some characteristics of the used system. Table III gives the (average) values of the selected metrics for the first and last versions of the subject system.

TABLE II: SOME CHARACTERISTICS OF THE USED SYSTEM.

Time frame (years)	Releases	First release computed		Last release computed	
		Version	Size (SLOC)	Version	Size (SLOC)
7.2	31	5.5.0	126 927	5.5.35	170 998

TABLE III: VALUES OF THE SELECTED METRICS.

Versions	Total no. of classes	Total KLOC	Qi	CBO	WMC
First vers.	837	126	0.743	8.97	29.8
Last vers.	1108	171	0.735	9.41	30.3

We retrieved the official releases on the 5.5.*x* branch from the official website of Apache. We used the Borland Together tool (http://www.borland.com/) to collect data on CBO and WMC metrics. For each released version of the subject system, we computed the metrics values at the micro level (classes) as well as at the macro level (system). We used the average as an aggregation method. We collected the Qi data using the tool we developed. We computed the Qi value for each class of each released version of the subject system. Here also, we computed the Qi values at the two levels (micro and macro). For our experiments, since we did not have any data on the test suites used for testing the subject system and knowing that the main purpose of this study is to investigate if the Qi metric can be used to observe the quality evolution along the evolution of the subject system, the testing coverage is set to 0.75 for all methods. As mentioned previously, the objective of the Qi metric is not to evaluate a design by giving absolute values, but more relative values that may be used for identifying the critical classes (in a relative way) on which more QA effort is required to ensure software quality.

A. Software Quality Evolution

In this section, we investigate the evolution of the Qi metric (in parallel with the selected OO metrics) along the evolution of the subject system. The objective is to observe how the Qi metric behaves relatively to these metrics. We also analyze the correlations between the Qi metric and each of the OO metrics at the two levels: micro and macro. We tested in our study the following hypothesis.

Hypothesis: The evolution of the OO metrics along the evolution of the subject system will be reflected (captured) in the Qi metric values on both micro and macro levels.

The analysis of the collected data allowed us to observe the values of the Qi (and OO metrics) along the period of evolution (overall trend) of the studied system. Fig. 1 shows the results (in terms of evolution) for the Qi and selected OO metrics (shown with the min-max normalization).

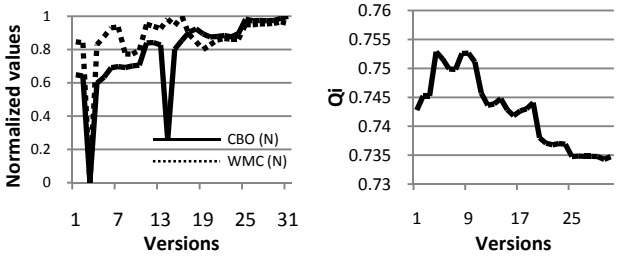

Fig. 1. Evolution of Qi and OO metrics.

From Fig. 1, it can be seen various significant variations (increase and decrease) along the evolution of Apache. The global trend for all the metrics is characterized by a steady increase and some fluctuations (peaks). The Qi curve shows a slight growth in the first iterations, which becomes however a decrease around the iteration 8. This negative growth appears relatively continuous. Results, overall, suggest that the Qi metric captures (in general) the evolution of OO metrics. In order to validate our hypothesis, we analyzed the correlations between Qi and selected OO metrics (as indicators of software quality from an internal perspective) along the evolution of the subject system. We performed statistical tests using correlation. We used both Spearman's and Pearson's correlation coefficients in our study. These techniques are widely used for measuring the degree of relationship between two variables. Correlation coefficients will take a value between -1 and +1. A positive correlation is one in which the variables increase together. A negative correlation is one in which one variable increases as the other variable decreases. A correlation of +1 or -1 will arise if the relationship between the variables is exactly linear. A correlation close to zero means that there is no linear relationship between the variables. We used the XLSTAT (http: //www.xlstat.com/) tool to perform the analysis. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations where significant. A significant correlation between 0.7 and 1.0 (or -0.7 and -1.0) is considered as a strong correlation [9].

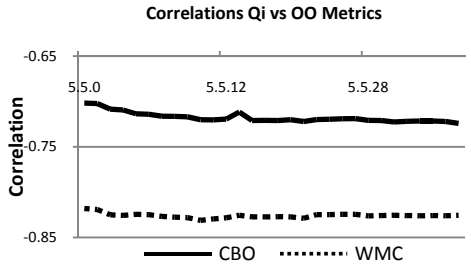

Fig. 2. Evolution of correlations between Qi and OO metrics (micro level).

TABLE IV: CORRELATION MEAN VALUES BETWEEN QI AND OO METRICS (MICRO LEVEL).

OO Metrics	Avg. value
CBO	-0.7178
WMC	-0.8260

For the micro level, we selected the classes present from the first version of the system to its latest version. We considered that these classes represent in some ways the core of the system throughout the period of evolution. We calculated correlations between the values of Qi (for each class) and the values of OO metrics. Fig. 2 shows the evolution of the correlation values (Pearson) between the Qi and selected OO metrics along the different versions of Apache Tomcat. Table IV shows the average value of these correlations.

It can be seen, from Fig. 2 and Table IV, that the obtained correlations between Qi and OO metrics are strong, and this throughout the evolution of the system. Moreover, they remain relatively stable from one iteration to another (Fig. 2), aside the slight peak observed for version 5.5.13. Several observations on the calculated metrics can explain this fluctuation. In comparison with version 5.5.12, the size in terms of lines of code of the version 5.5.13 increases (from 148 900 to 149 500), while the number of classes decreases (from 968 to 964), which has the effect of increasing the average size (in terms of lines of code) of classes (from 153.8 to 155.1), decreasing the coupling (CBO from 9.19 to 8.49) and increasing the number of operations per class (from 12.9 to 13.1). Between these two versions, we can still observe a stability of Qi. We can therefore explain these observations by an amount of added instructions which is concentrated in a (relative) small number of classes.

Moreover, the correlations between Qi and selected OO metrics are negative. A negative correlation indicates that one variable (Qi metric) decreases as the other variable (OO metrics) increases. These results are plausible and not surprising. In fact, the more strongly a class is coupled (with a high complexity and large size) to other classes, the less the quality of the class is likely to be. A low value of the Qi of a class (probably a high value of coupling and complexity) indicates that the class is a high-risk class and needs a relatively high QA effort to ensure its quality. These results (and observations) suggest that the Qi metric, at the micro level, captures not only the evolution of the selected OO metrics but also an important part of the information captured by these metrics (given the high correlation values). These results support therefore our hypothesis at the micro level.

TABLE V: CORRELATIONS BETWEEN QI AND OO METRICS (MACRO LEVEL).

Variables	CBO	WMC
Qi	**-0.854**	**-0.566**

For the macro level, we used the average values of the metrics for each version. Table V shows the correlations values (Spearman) obtained between Qi and OO metrics for the system studied. From Table V, it can be seen that correlations values between Qi and selected OO metrics are significant (in boldface) and relatively high. This suggests that the Qi metric captures the evolution of the selected OO metrics for an evolving system at the macro level. Such observations thus allow us reasonably to validate our hypothesis at the macro level.

B. Threats to Validity

The study performed in this paper should be replicated using many other OO software systems in order to draw more general conclusions about the ability of the Qi metric to reflect the evolution of the quality of software systems. In fact, there are a number of limitations that may affect the results of the study or limit their interpretation and

generalization. The achieved results are based on the data set we collected from one open source software system. The used system is, however, a relatively large project. The collected data cover a period of more than seven years. Even if we believe that the analyzed data set is large enough to allow obtaining significant results, we do not claim that our results can be generalized. It is also possible that facts such as the development style used by the developers for developing and maintaining the code of the subject system (or other related factors) may affect the results or produce different results for specific applications.

V. Conclusions and Future Work

In this paper, we investigated the quality evolution of an open source Java software system using metrics. Software quality was addressed from an internal point of view. We wanted to investigate if the Qi metric, which we proposed in a previous work, can be used to observe (understand) how quality evolves along the evolution of the subject software system. We used OO design metrics for measuring the internal (structural) quality of a released version (in terms of coupling and complexity). We performed an empirical analysis using historical data collected from the successive released versions of the subject software system. Empirical results provide evidence that the Qi metric may be used to observe the evolution of software quality along the evolution of successive released versions of a software system. The achieved results are, however, based on the data set we collected from only one system. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. They show, at least, that the Qi metric, as a synthetic metric, offers a promising potential for capturing (reflecting) the quality evolution of evolving software systems. Further investigations are, however, needed to draw more general conclusions. The performed study should be replicated using many other OO software systems. As future work, we plan to (among others): investigate if the Qi metric may be used to observe the evolution of software quality from an external point of view (using the number of defects as a quality indicator) and replicate the study on other OO software systems to be able to give generalized results.

References

[1] M. M. Lehman and L. A. Belady, *Program Evolution: Processes of Software Change*, Academic Press, 1985.

[2] I. Sommerville, *Software Engineering*, 9th Edition, Addison Wesley, 2010.

[3] G. Xie, J. Chen, and I. Neamtiu, "Towards a better understanding of software evolution: An empirical study on open source software," in *ICSM '09*, pp. 51-60, 2009.

[4] M. M. Lehman, "Laws of software evolution revisited," in *Lecture Notes in Computer Science*, vol. 11, pp. 108-124, 1997.

[5] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution – The nineties view," in *Proceedings of the Fourth International Software Metrics Symposium,* pp. 20-32, 1997.

[6] P. L. Parnas, "Software aging," in *Proceedings of the 16th International Conference on Software Engineering (ICSE '94)*, pp. 279-287, 1994.

[7] J. V. Gurp and J. Bosch, "Design erosion: Problems and causes," in *Journal of Systems and Software*, vol. 61, no. 2, pp. 105-119, 2002.

[8] H. Zhang and S. Kim, "Monitoring software quality evolution for defects," in *IEEE Software*, vol. 27, no. 4, pp. 58-64, 2010.

[9] Y. Lee, J. Yang, and K. H. Chang, "Metrics and evolution in open source software," in *7th International Conference on Quality Software (QSIC '07)*, pp. 191-197, 2007.

[10] T. Mens and S. Demeyer, "Future trends in software evolution metrics," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, pp. 83-86, 2001.

[11] B. H. Sellers, *Object-Oriented Metrics-Measures of Complexity*, Prentice Hall, New Jersey, 1996.

[12] M. Badri and F. Touré, "Empirical analysis for investigating the effect of control flow dependencies on testability of classes," in *23rd International Conference on Software Engineering and Knowledge Engineering*, 2011.

[13] V. Basili, L. Briand, and W. L. Melo, "A validation of object oriented design metrics as quality indicators," in *IEEE Transactions on Software Engineering*, vol. 22, no. 10, 1996.

[14] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," in *Journal of Systems and Software 51*, pp. 245-273, 2000.

[15] M. Dagpinar and J. H. Jahnke, "Predicting maintainability with object-oriented metrics - An empirical comparison," in *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*, pp. 155-164, 2003.

[16] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *2011 IEEE Fourth International Conference on Software Testing, V&V Workshops*, pp. 566-571, 2011.

[17] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd Edition, PWS Publishing Company, 1997.

[18] A. Murgia, G. Concas, S. Pinna, R. Tonelli, and I. Turnu, "Empirical study of software quality evolution in open source projects using agile practices," in *CoRR*, vol. abs/0905.3287, 2009.

[19] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," in *Software Quality Journal*, vol. 18, no. 1, pp. 3-35, 2010.

[20] R. Subramanyan and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," in *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297-310, 2003.

[21] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," in *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771-789, 2006.

[22] M. Badri, L. Badri, and F. Touré, "Empirical analysis of object-oriented design metrics: Towards a new metric using control flow paths and probabilities," in *Journal of Object Technology*, vol. 8, no. 6, pp. 123-142, 2009.

[23] M. Badri and F. Touré, "Evaluating the effect of control flow on the unit testing effort of classes: An empirical analysis," in *Advances in Software Engineering*, vol. 2012.

[24] S. R. Chidamber and C. F. Kemerer, "A metric suite for object-oriented design," in *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[25] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th International Conference on Software Engineering (ICSE '06). ACM*, pp.452-461, 2006.

[26] W. Ambu, G. Concas, M. Marchesi, and S. Pinna, "Studying the evolution of quality metrics in an agile/distributed project," in *Extreme Programming and Agile Processes in Software Engineering*, pp. 85-93, 2006.

[27] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," in *Journal of Systems and Software*, vol. 1, no. 3, pp. 213-221, 1980.

[28] A. Jermakovics, M. Scotto, and G. Succi, "Visual identification of software evolution patterns," in *9th International Workshop on Principles of Software Evolution (IWPSE '07): in Conjunction with the 6th ESEC/FSE Joint Meeting*, pp. 27-30, 2007.

[29] T. Mens, J. F. Ramil, and S. Degrandsart, "The evolution of eclipse," in *IEEE International Conference on Software Maintenance 2008 (ICSM '08)*, pp. 386-395, 2008.

[30] L. Yu, S. Ramaswamy, and A. Nail, "Using bug reports as a software quality measure," in *Proceedings of the 16th International Conference on Information Quality (ICIQ '11)*, pp. 277-286, 2011.

[31] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Lalhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study," in *Software Process: Improvement and Practice*, vol. 16, no. 1, 2009.