Measuring Developers' Contribution in Source Code using Quality Metrics

Patricia Rücker de Bassi
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba – Brazil
patricia.bassi@ppgia.pucpr.br

Gregory Moro Puppi Wanderley
Heudiasyc UMR CNRS 7253
Université de Technologie de Compiègne
Compiègne – France
gregory.wanderley@hds.utc.fr

Pedro Henrique Banali
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba – Brazil
pedro.banali@ppgia.pucpr.br

Emerson Cabrera Paraiso
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba – Brazil
paraiso@ppgia.pucpr.br

***Abstract*—People involved in software development seek better ways of developing quality software. As software development is a collaborative activity dependent on technology and performed by group of people, the quality can be directly linked to the degree of collaboration and commitment of the development team. This research aims to identify and analyze software quality metrics that can measure software development team members' participation regarding their contribution in the project source code. Analyzing such information can contribute with software engineering project managers to conduct and better organize project development teams. To achieve this goal, we defined a set of quality metrics, applied them to open source code and analyzed them in other to evaluate members' contribution. Results show that it is possible to determine whether the team member contribution increase, decrease or had no influence on source code quality**.

Keywords—Collaborative Software Development, Quality Metrics, Open Source Projects, Developer Contribution

I. INTRODUCTION

Software development is a collaborative activity dependent on technology and performed by group of people such as managers, software architects, developers, testers, and requirements engineers ([27], [38]). As a teamwork, quality in software can be directly linked to development team`s degree of collaboration and commitment ([38], [20]).

Seeking for better quality, measurement process has played an increasingly important role in Software Engineering. In this sense, software metrics allow measurement, evaluation, control, and improvement of software products and processes ([6] [18]). Dozens of metrics have been proposed since the mid-60s. Despite the quantity of metrics, their adoption and application by practitioners has been limited [37]. A challenge to use them is to interpret them for supporting decision-making.

Although software metrics were originally defined to measure software artifact characteristics, such as the number of code lines, the number of methods, and so on, metrics may be used to measure developer's activity [28]. The impact of these metrics, also called Process Metrics, is that developers' habits have a deeper impact on software quality than the intrinsic characteristics of software artifacts, as suggested by [22] and [31].

People involved in software development seek better ways of developing quality software. Even the effects of software programming languages are being studied, aiming at finding out if the idiosyncrasies of different programming languages can increase software quality [33]. Thus, the motivation of this research is to find out ways for measuring individual contributions to source code. In other words, this research aims to identify and to analyze software quality metrics that can measure software development team members' participation regarding their contribution in the project source-code. Analyzing such information can contribute with software engineering project managers to set project teams considering developer's profile. Therefore, incrementing project team qualification and collaboration may increase team development productivity and code quality. To achieve this goal, we defined a set of quality metrics, applied them to open source code and analyzed them to evaluate members' contribution.

According to a Forrester Consulting survey which compared large companies in Europe and North America [10], the adoption of free and open source software has increased significantly in the last decades to the point of becoming influential to the global economy. This survey also states that 92% of the senior business and IT executives say that free software products have met and, in some cases, exceeded their quality expectations. Such a satisfaction and quality are usually achieved thanks to the collaboration of many users and developers who report failures, fix bugs, or add features.

This paper is organized as follows: first we present the quality metrics. Then, in Section 3 we present a method to evaluate developers' contribution using quality metrics. The experiments that we conducted, and the results obtained are detailed in Section 4. We survey related works in the area in Section 5. Finally, in Section 6 we offer our conclusions as well as directions for future work.

II. QUALITY SOFTWARE METRICS

Software metrics provide measurement of the software product and the process of software production. Good metrics

should enable the development of models that are efficient of predicting process or product spectrum. According to Rawat and colleagues [32] optimal metrics should be simple, precisely definable, so that it is clear how the metric can be evaluated; objective, to the greatest extent possible; easily obtainable or at reasonable cost; valid, so the metric should measure what it is intended to measure; and robust that is relatively insensitive to insignificant changes in the process or product.

One can group software metrics in three different groups, named: Process Metrics, Project Metrics, and Product Metrics [15]. The Process Metrics highlight the process of software development. Project Metrics are used to monitor project situation and status. Product Metrics describe the attributes of the software product at any phase of its development. Product Metrics may measure program size, software design complexity, performance, portability, maintainability, and product scale. They are used to presume, invent product´s quality and measure final product´s medium.

Source code´s quality can be measured by Software Metrics regard to complexity and maintainability. These quality software metrics seek to manage and reduce structures complexity, improve maintainability, and source code development and consequently the software itself 0.

The Object-Oriented (OO) software development approach provides many advantages such as reusability, decomposition of problem into easily understandable objects and the aiding of future modifications. To ensure quality, OO programming metrics is an aspect to be considered. Metrics should be a set of standards against which one can measure the effectiveness of OO analysis techniques in the design of a system [34].

Studies such as [1] [29] [15] [18] [21] and [8] report that source code metrics may indicate situations of increasing or decreasing of code quality related to maintainability, testability, reusability and complexity.

Table 1 shows an excerpt of these studies, grouping those metrics based on its influence on code quality. The first category groups complexity metrics, considering that more complexes OO artefacts like code, class and methods may turn code more difficult to understand, maintain and test ([23], [4], [15] and [21]). The second category form the inheritance category, since they can indicate problems of abstraction and software maintenance ([4], [35] and [15]). Some size metrics can indicate understanding, maintenance, and reuse code problems ([15] and [14]). They form the Size Metrics category. Finally, strong coupling in software turn more difficult to maintain, modified and reuse code, also can compromise code abstractness and stability [23]. They form the Coupling Metrics category.

Each metric in Table 1 has its own range that can be used to evaluate the real state of the source code. For instance, the MCC measures code complexity. It has the following range [1]: l*ow complexity*: $1 \leq MCC \leq 10$; *moderate complexity*: $10 < MCC \leq 20$; *high complexity*: $20 < MCC \leq 50$; and *very high complexity*: $MCC > 50$. In this case, the ideal situation should place this metric in the *low* level.

Considering that quality software metrics can indicate how

Table 1. Quality Metrics Excerpt.

Metric influence on code quality	Metric
Complexity Metrics More complexes code/class/method more difficult to understand, maintain and test.	McCabe's Cyclomatic complexity metric - MCC
	Weighted Methods per Class metric -WMC
	Lack of Cohesion in Method metric-LCOM*
	Nested Block Depth – NBD
Inheritance Metrics Can indicate problems of abstraction and software maintenance.	Depth of Inheritance Tree - DIT
	Number of Children – NOC
	Number of Overridden Methods – NORM
	Specialization Index - SIX
Size Metrics Can indicate problems of understanding, maintenance, and reuse.	Method Lines of Code -MLOC
	Number of Attributes per Class - NOA
	Number of Static Attribute - NSF
	Number of Static Methods - NSM
	Number of Parameter - NOP
	Number of Interfaces - NOI
	Number of Package – NOP
Coupling Metrics Strong coupling in software is more difficult to maintain, modified, reused. Can compromise code abstractness and stability.	Afferent Coupling - Ca
	Efferent Coupling – Ce
	Instability metric - I, named here as RMI
	Abstractness - A, named here as RMI
	Normalize Distance from Main Sequence – D

a developer contribution impact on project code quality regarding maintainability, testability, reusability, and understandability (Table 1), in this research we are using this information to identify whether a developer contribution in a project source code increases, decreases or maintains the project quality metric.

To reach this goal, we propose to apply 20 quality metrics in open source software projects code as described in the next section.

III. MEASURING DEVELOPER´S CONTRIBUTION

As described in the introduction, given a collection of software quality metrics, our goal is to show that they can express the degree of contribution of each participant in software development. The idea is to evaluate each contribution as developers commit their code.

Measurement execution involves gathering and storing data related to defined metrics, according to their measurement procedures. Once data is gathered, it should be analyzed according to an analysis procedure [24].

Fig. 1 shows the main steps of the method. The algorithm is quite simple. Given a project under development, each commit is produced by a contributor, so a project can have k contributors committing in the same code, throughout the timeline. For each contributor's commit, 20 quality metrics are computed. These metrics values were analyzed based on positive, negative and no influence on code maintainability, testability, reusability, and understandability (Table 1).

To understand the approach we have developed, let us consider an open source project where a small team (around 10 developers) working in a collaborative environment contributes by building commits to develop required features. One may commit a feature that increase, for instance, the Complexity Metric MCC. Because the MCC metric is related

```
procedure applyMetricsToCode;

artifact = {class | method | package};
vet = ∅;
commit = 1;
while committing do
    Recover the commit source code, the commit
sequence and the commit contributor;
    Rebuilt the commit source code;
    for metric 1 to 20 do
        Compute the commit source code artifact
metric value;
        diff = abs (commit metric value - (commit -
1) metric value);
        if diff <> 0 then
           vet[metric, artifact, commit sequence,
              commit contributor] = metric value;
        end-if
    end-for
    commit = commit + 1;
 end-while
```

Fig. 1. The main steps of the method.

Table 2. Dataset used on experimentation.

Project	# of commits	# of classes	# of lines	# of contributors	Duration (months)
CSSEmbed	32	8	1516	6	37
Elastic	18	5	277	5	25

to more logical paths to be tested, this may affect in a negative way the source code understanding, testability, and maintainability.

After computing each metric for a given code (or part of it), we store all the values of the metrics to be used in the analysis process. The evolution of a given metric is obtained by comparing it in each commit with its antecessor. Note that, it is important to register who (contributor) is responsible for that commit. This allows us to indicate which contributor is responsible for a change in each metric.

IV. Results and Discussions

Some experiments were performed using a dataset consisting of source code available at GitHub. GitHub contains several real-world software projects most of them developed applying Collaborative Software Development methods, thus it becomes a suitable environment to apply our research. We chose randomly two projects developed in Java and full available (since we need to rebuild them commit by commit, we succeeded rebuilding these two projects). With that in mind, we applied the approach presented in Section 3 to these two small and finished projects. Table 2 shows the main dataset features.

Fig. 2 shows the steps we performed during our experiments. However, first we selected the projects (P) containing *m* commits each one. Note that each commit is developed by a single and only a single contributor. Moreover, a project can have k contributors committing in the same code (1), following a timeline.

$$\forall m \in \mathrm{P}: \exists k \text{ that developed } m \qquad (1)$$

Then, for each contributor's commit we computed 20 quality metrics.

After running the algorithm presented in Fig. 2, the output is a database containing the values of the metrics for each

```
procedure applyMetricsToProjects;

artifact = {class | method | package};
for project 1 to n
   commit = 1
   while commit <= m do
       Recover the commit source code
       Rebuilt the commit source code
       Compute the commit source code metrics
       Save the commit source code metrics
   end-while
   Combine all n projects metrics keeping the m
commits time line by metrics
   for metric 1 to 20
       for each artifact do
            for commit 1 to m
                diff = abs (commit metric value -
(commit -1) metric value)
                if diff <> 0 then
                    for the whole artifact metric do
                        Save the metric, the
artifact, the commit sequence, the commit contributor
and the metric value
                    end-for
                end-if
            end-for
       end-for
   end-for
end-for
```

Fig. 2. The main steps of the experiments.

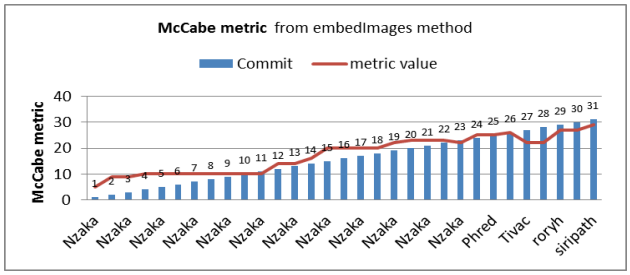

Fig. 3. Applying McCabe Metric (MCC) to *embedImages* method from CSSEmbed project.

artifact and contributor of a given project.

We raise the hypothesis that the values of the metrics obtained for all commits can, in some cases, indicate changes in software quality. We plotted many charts to study whether a commit increases, decreases or does not influence the code maintainability, testability, reusability, and understandability.

Fig. 3 shows the evolution of the MCC metric applied to the *embedImages* method of the CSSEmbed project. The chart shows the commits made by all the project contributors along with the value of the MCC metric for each commit. As mentioned by [1], the MCC metric is related to the degree of difficulty of understanding he code, affecting testability and maintainability. A method with high MCC value means that one has more logical paths to test. It also indicates that maintenance is more difficult. As one may see in the chart, the metric value varies as commits are performed. From commit 1 to 15 the MCC values are considered moderate; from commit 16 to the end, the values go high which may indicate higher risk for method testability. The method loose testability, maintainability and understanding quality as the commits are performed.

At the same time, the MCC from commit 27 to 28, carried by contributor Tivac, go modestly lower indicating a slight improvement in quality.

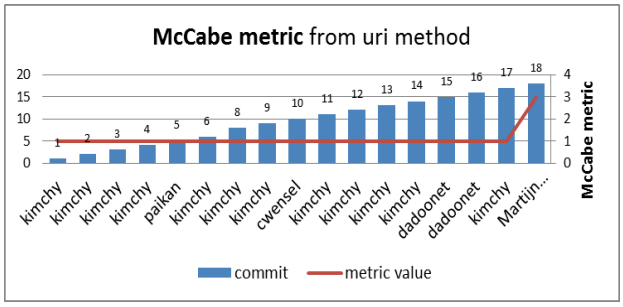

Fig. 4. McCabe Metric (MCC) of *uri* method from Elastic project.

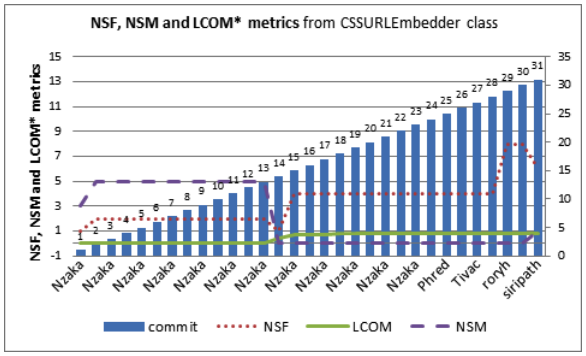

Fig. 5. NSF, NSM and LCOM* metrics of CSSURLEmbedder class from CSSEmbed project.

The same happens when applying this metric in the *uri* method from the Elastic project (Fig. 4). The MCC increases after the commit 17 from contributor Kimchy, degrading the method quality regarding maintainability and testability.

Fig. 5 shows the LCOM* metric evolution of the *CSSURLEmbedder* class from CSSEmbed project. The LCOM* considers that a class is cohesive when all methods access all attributes, resulting in LCOM* equal to zero. A LCOM* close to 1 indicates a lack of cohesion, and a value greater than 1 indicates serious problems of cohesion, evidencing that in this case the class must be subdivided into other classes [14]. In Fig. 5 one may see that the LCOM* starts close to zero and reaching the end of the project it comes closer to 1, indicating that the quality with respect to cohesion is decreasing as the commits are implemented. Clearly, the commit 15 from the contributor Nzaka degraded the quality of cohesion in this class, and this situation remained until the end of the project.

The other two metrics presented in Fig. 5 are NSF and NSM. NSF calculates de number of static attributes and NSM shows the number of static methods of a class. A high value of NSF can turn difficult the class understandability and maintainability. A high value of NSM may indicate procedural programming and not OO programming [15]. As one may see in Fig. 5, the NSM starts in 3, then it goes to 5, and then it decreases to 0, finally ending in 1.

This variation shows that the recommended use of OO programming paradigm increases as the commits occur. The chart also shows that the NSF metric starts in 1, then it increases to 4 and to 8, finally ending in 6. Although it is possible to notice a reduction in the metric value, it ends in 6,

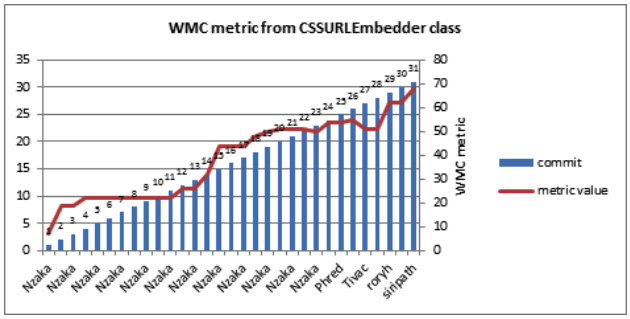

Fig. 6. WMC metric of CSSURLEmbedder class from CSSEmbed project.

showing that the code understandability and maintainability become a little bit more difficult, decreasing the code quality.

The WMC metric measures the complexity of a class considering the complexity of its methods, meaning that a class with several methods is more complex to be maintained [29]. The chart on Fig. 6 shows the WMC variation as the commits are performed. One may see that from commit 4 performed by Nzaka, the WMC evolves from *low risk* to *moderate risk*, indicating quality degradation related to maintainability. In commits 19 (made by Nzaka), 27 and 28 (made by Tivac) the WMC decreases, indicating a slight improvement in quality. However, the project closes with a WMC of 68 (see commit 31 performed by Siripath), ending with a *moderately high risk*.

Fig. 7 shows the WMC metric from the *NodeServlet* class of the Elastic project. Throughout the project the metric stayed in a *low risk* level. In the commits 8 and 9, performed by Kimchy, it decreased slightly (increasing quality), and it rose again with Cwensel's next commit (commit 10).

Fig. 8 shows the evolution of the metrics Ce, Ca and RMI. The RMI metric refers to package instability and concerns the relationship between the Efferent Coupling (Ce) relative to the total coupling. The Afferent Coupling (Ca) concerns the number of classes from other packages that depend on classes of the considered package. The Efferent Coupling (Ce) is the number of classes in a package that depend on classes from other packages.

According to Martin [21], the Ce and Ca metrics help to understand the probability of a defect in a class replicate through a system. Fig. 8 shows that the metric RMI is at the

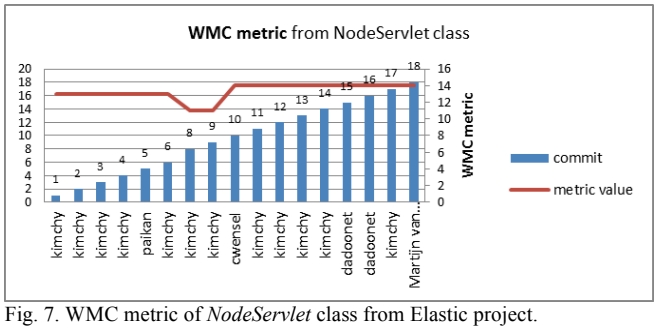

Fig. 7. WMC metric of *NodeServlet* class from Elastic project.

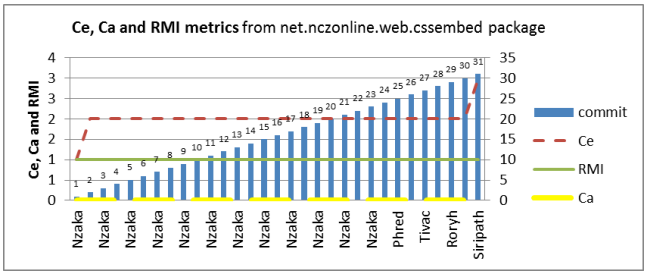

Fig. 8. Ce, Ca and RMI metrics from *net.nczonline.web.cssembed* package of CSSEmbed project.

maximum instability (value 1). Also, according to Martin, a package is unstable when it depends on many other packages. In Ce metric (Fig. 8), one may see that even changing the Ce value of the package, there was no modification in RMI. In this case the Ca value of the package, also shown in Fig. 8, remained zero throughout the project, explaining the RMI with maximum instability.

After analyzing the results of all the 20 metrics, we learned some important lessons. First, we confirmed the hypothesis that one can evaluate individual contributions using quality metrics. Moreover, it is also possible to evaluate the evolution of the metrics in real time (even though in our experiments we used finished projects). The evolution of each metric may influence the code maintainability, testability, reusability and understanding. A significant point is that, we can identify, for each metric in every artifact, if it contributes positively, negatively or have no influence on the source code. Furthermore, we noted that changes in the values of metrics in the beginning of the project are expected.

Finally, we proved that it is possible to relate the software quality metrics of each team member commit to maintainability, testability, reusability, and understandability.

V. Related Work

We studied a number of researches that applied quality metrics in source code ([25] [12] [13] [17] [36] and [7]). None of them were interested in evaluating the individual contribution of a member on a project source code. We did not find a work that relates the software quality metrics of each team member commit to maintainability, testability, reusability, and understandability. We found some works that use quality metrics with different goals.

Meirelles and colleagues [26] work contributes with an analysis of source code metrics from thousands of free software projects, causally linking software source code characteristics with attractiveness, as free software projects fail when they lack attractiveness. The results indicate that source code size and structural complexity explain a relevant percentage of the attractiveness of free software projects. The analysis indicates that a project should keep its complexity constant as new code is incorporated, because developers are interested in improving the software, thereby increasing their chances of forming a community of contributors around their software, further enhancing its quality. Thus, projects should grow managing their complexity, keeping the new members willingness to contribute.

Perkushi and colleagues [30] proposed a method to construct Bayesian Networks to assist on the interpretation of software metrics considering the influence of subjective factors, like lack of expertise or attempt to have better metric numbers to present to the manager.

The objective of Foucault and colleague study [11] was to replicate the study of ownership metrics of Bird et al. on FLOSS projects. Such metrics are known to be good quality indicators and can be used to organize software development teams. They observed that classical code metrics are better quality indicators, and no relationship was found between ownership metrics and module faults when controlling for module size.

Finlay and colleagues research [9] presents the outcomes of an initial systematic attempt to predict build success and/or failure for a software product by utilizing source code metrics. To achieve this goal, they used a data stream mining technique, the Hoeffding Tree algorithm.

Drouin, Badri and Touré [5] proposed Qi (quality assurance indicator) a synthetic metric, which captures in an integrated way different OO software attributes to observe how quality evolves along the evolution of software systems.

Beal and colleagues [3] built a developer model based on user modelling methods and source code quality software metrics data applying a machine learning approach. Their goal is to return results to developers, so they can improve their programming skills, managers can better organize their development teams, and companies can better qualify their development employees.

Also, Barroso and colleagues [2] worked on analyzing the influence of human personality on software quality. They performed a psychological test on a set of industry developer's workers and applied OO software metrics on software developed by each one. They evidenced, through statistical analysis some influences.

VI. Conclusions and Future Work

Software development is considered a collaborative activity since it involves various professionals, inter alia, software engineers, system architects, developers, and testers, who develop activities together and contribute to achieve the same goal. These professionals and the organizations where they work are constantly seeking to improve the quality of their projects to deliver a suitable product for their customers, using efficient and appropriate technology and seeking low-cost development.

Given this context, this research sought to answer the question regarding the contribution of a team member in software development project considering the source code level. A set of software quality metrics have been defined trying to identify its influence in code maintainability, testability, reusability, and understandability requirements. A method was defined and applied to identify whether the commit performed by the contributor increases/decreases/have understandability.

The analysis of metric values was performed using metric influence in code quality for each contributor's commit

recovered from GitHub project repository source code. It was possible to check whether each contributor commit quality metric affected positively, negatively or did not influence the code maintainability, testability, reusability, and understandability.

In the next steps we are collecting data to evaluate larger projects. We are also applying opinion mining on commit's messages changed between project manager and developer, so we can identify weather changes made at the commit was ordered by managers or not. With these approaches we are elaborating another and extended set of metrics, so it will be possible to increase developer's profile contributing with software engineering project managers to set project teams. Finally, on developer's size, we are setting up a real-time evaluation. We will follow a set of metrics during code development, presenting the results in real time. This will allow us to check if, knowing what is happening to their code, contributors will change the way they produce the code.

REFERENCES

[1] Anderson, J L. Using software tools and metrics to produce better quality test software. AUTOTESTCON 2004, 2004.

[2] Barroso, A. S. Silva, J. S. M. Souza, T. D.S. Cezario, B. S. de A.Soares, M. S. Nascimento, R.P.C. Relationship between Personality Traits and Software Quality – Big Five Models vs. Object-oriented Software Metrics. ICEIS 2017

[3] Beal, F. de Bassi, P. R. Paraiso, E. C. Developer Modelling using Software Quality Metrics and Machine Learning. ICEIS 2017.

[4] Chidamber, S R. Kemerer, C. F. A metrics suite for object-oriented design. IEEE Transaction on Software Engineering, 1994.

[5] Drouin, N. Badri, M. Touré, F. Analyzing Software Quality Evolution Using Metrics: an empirical study on open source software. Journal of Software, 2013.

[6] Fenton, N E. Neil, M. Software metrics: Roadmap. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, ACM, New York, NY, USA, 2000

[7] Ferreira, K A. Bigonha, M A. Bigonha, R S. Mendes, L F. Almeida, H C. Identifying thresholds for object-oriented software metrics. Journal of Systems and Software, 2012.

[8] Filó, T. Bigonha, M. Ferreira, K. A catalogue of thresholds for object-oriented software metrics. In *1st International Conference on Advances and Trends in Software Engineering, IARIA*. 2015

[9] Finlay, J. Pears, R. Connor, A M. Data stream mining for predicting software build outcomes using source code metrics. Journal of Information and Software Technology. Elsevier, 2014.

[10] Forrester-consulting, Open Source Paves the Way for the Next Generation of Enterprise IT, Forrester Research, Tech. Rep., 2008.

[11] Foucault, M. Falleri, J. Blanc, X. Code ownership in open-source software. EASE 2014, England, 2014.

[12] Gómez, V U. Ducasse, S. D'Hondt, T. Visually characterizing source code changes. Journal of Science of Computer Programming, Elsevier, 2013.

[13] Goyal, R. Chandra, P. Singh, Y. Why Interaction between Metrics should be considered in the Development of Software Quality Models: A Preliminary Study. ACM SIGSOFT Software Engineering Notes, 2014.

[14] Harrison, R. Counsell, S. Nithi, R. An overview of oriented-object design metrics. 8th International Workshop on Software Technology and Engineering Practice. 1997.

[15] Henderson-Sellers, B. Object-Oriented Metrics: Measures of Complexity. Prentice Hall, 1996.

[16] Honglei, T. Wei, S. Yanan, Z. The Research on Software Metrics and Software Complexity Metrics, International Forum on Computer Science-Technology and Applications, 2009.

[17] Karus, S. Dumas, M. Code churn estimation using organizational and code metrics: an experimental comparison. Journal of Information and Software Technology, Elsevier, 2011.

[18] Kitchenham, B. What's up with software metrics? A preliminary mapping study. Journal of Systems and Software, SI: Top Scholars, 2010.

[19] Li, H. A Novel Coupling Metric for Object-Oriented Software Systems. 2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop, 2008.

[20] Magdaleno, A M. Barros, M O. Werner, C M L. Araujo, R M. Batista, C F A. Collaboration in process optimization software composition. The Journal of Systems and Software, 2015.

[21] Martin, R C. Agile Software Development: Principles, Patterns, and Practices. Alant Apt Series. Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[22] Matsumoto, S. Kamei, Y. Monden, A. Matsumoto, K.-i. Nakamura, M. An analysis of developer metrics for fault prediction. 6th International Conference on Predictive Models in Software Engineering, 2010.

[23] McCabe, T. Complexity Measure. IEEE Transactions on Software Engineering, 1976.

[24] McGarry, J. Card, D. Jones, C. Layman, B. Clark, E. Dean, J. Hall, F. Practical Software Measurement: Objective Information for Decision Makers. Addison Wesley, 2002.

[25] Mehdi, A. Urso, P. Charoy, F. Evaluating Software Merge Quality. EASY'14, England, 2014.

[26] Meirelles, P. Jr Santos, C S. Miranda, J. Kon, F. Terceiro, A. Chavez, C. A study of the relationships between source code metrics and attractiveness in free software projects. Brazilian Symposium on Software Engineering, 2010.

[27] Mohtashami, M. Ku, C S. Marlowe, T J. Metrics are needed for collaborative software development, The Journal of Systemics, Cybernetics and Informatics, 2011.

[28] Munson, J. C. Elbaum, S. G. Code churn: A measure for estimating the impact of code change. 1998 International Conference on Software Maintenance, 1998.

[29] Olague, H M. Etzkorn, L H. Cox, G W. An Entropy-Based Approach to Assessing Object-Oriented Software Maintainability and Degradation - The Method and Case Study. Software Engineering Research and Practice, CSREA Press, Nevada, USA, 2006.

[30] Perkusich, M. Medeiros, A. Silva, LC. gorgonio, K C. Almeida, H. O. Perkusich A. A Bayesian network approach to assist on the interpretation of software metrics. SAC 2015, Spain, 2015.

[31] Rahman, F. Devanbu, P. How, and why, process metrics are better. 2013 International Conference on Software Engineering, 2013.

[32] Rawat, MS. Mittal, A. Dubey, S K. Survey on Impact of Software Metrics on Software Quality. Journal of Advanced Computer Science and Application, 2012.

[33] Ray, B. Posnett, D. Devanblu, P. Filkov, V. A Large-Scale Study of Programming Language and Code Quality in GitHub. Communications of the ACM, 2017.

[34] S. NASA. Software Quality Metrics for Object Oriented System Environments. Crosstalk Journal of Defense Software Engineering. 1997.

[35] Schroeder, M. A practical guide to object-oriented metric. IT Professional. 1999

[36] Taibi, F. Reusability of Open-Source Program Code: A Conceptual Model and Empirical Investigation. ACM SIGSOFT Software Engineering Notes, 2013.

[37] Wallace, L. G. Sheetz, S. D.The adoption of software measures: A technology acceptance model (tam) perspective. Information & Management, 2014.

[38] Whitehead, J. Mistrik, I. Grundy, J. Collaborative software engineering: concepts and techniques, in: Mistrík I, Grundy J, van der Hoek A, J Whitehead (eds.) Collaborative Software Engineering, Springer, 2010.

[39] Yu, S. Zhou, S. The survey on metric of software complexity. 2nd IEEE International Conference on Information Management and Engineering (ICIME), 2010.