Assessing Developer Contribution with Repository Mining-Based Metrics

Jalerson Lima
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte and
Federal Institute of Education at Rio Grande do Norte
Natal, Rio Grande do Norte, Brazil
jalerson.lima@ifrn.edu.br

Christoph Treude, Fernando Figueira Filho, Uirá Kulesza
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Natal, Rio Grande do Norte, Brazil
{ctreude,fernando,uira}@dimap.ufrn.br

Abstract— **Productivity as a result of individual developers' contributions is an important aspect for software companies to maintain their competitiveness in the market. However, there is no consensus in the literature on how to measure productivity or developer contribution. While some repository mining-based metrics have been proposed, they lack validation in terms of their applicability and usefulness from the individuals who will use them to assess developer contribution: team and project leaders. In this paper, we propose the design of a suite of metrics for the assessment of developer contribution, based on empirical evidence obtained from project and team leaders. In a preliminary evaluation with four software development teams, we found that code contribution and code complexity metrics received the most positive feedback, while participants pointed out several threats of using bug-related metrics for contribution assessment. None of the metrics can be used in isolation, and project leaders and developers need to be aware of the benefits, limitations, and threats of each one. These findings present a first step towards the design of a larger suite of metrics as well as an investigation into the impact of using metrics to assess contribution.**

Index Terms—**Project management, software contribution metrics, mining software repositories.**

I. Introduction and Motivation

Assessing the contribution of a developer in a software project is a challenging task, yet project leaders and managers are constantly faced with having to make decisions based on the performance of their developers. Many potential sources of developer contribution have to be considered, ranging from contributions in source code repositories and issue tracking systems to contributions to documentation and coordination. There is no consensus among researchers on how to assess developer contribution, which is often conceptualized as productivity [1] [2].

While productivity is formally defined as input divided by output [3], output in software development is difficult to measure and researchers do not agree on one definition [4] [5] [6]. Number of lines of code (LOC) has been proposed as a developer contribution metric [7] [8], but its value is limited because it does not necessarily reflect the aggregated value of what is being produced and neither the quality [9] [10] [11]. Other studies have used different definitions of developer contribution to measure performance and participation, for example by including contribution to the issue tracking system [14] [15] [16].

However, while several developer contribution metrics have been proposed, few have been evaluated at software companies with the individuals who will base their decision on them – project and team leaders. In this work, we propose the design of a suite of developer contribution metrics based on empirical evidence obtained from project and team leaders. So far, we have collected feedback from project and team leaders on five repository mining-based metrics, covering code contribution, code complexity, and bug-related metrics. For each team, we calculated the evaluated metrics over a period of 12 weeks, resulting in metrics for 48 developers in total. We then showed the results to seven project and team leaders and asked them to evaluate the usefulness and applicability of each metric for their decision making. Unlike our previous work [12], the answers of participants were based on concrete data from their own developers, and unlike the work of Meyer et al. [13], we asked project and team leaders rather than developers for their input, considering that it is usually individuals in leading roles who need to assess the contribution of developers.

Code contribution and code complexity metrics received the most positive feedback from our participants. On the other hand, several threats were pointed out regarding the use of bug-related metrics for the assessment of developer contribution. While none of the metrics can be used in isolation, these findings provide a first step towards the design of a larger suite of metrics for the assessment of developer contribution, based on empirical evidence from project and team leaders. After reporting our preliminary findings, we propose a set of metrics that cover aspects missed by the metrics that we have evaluated so far, including communication, collaboration and task distribution metrics. In addition and in light of recent research on gamification in software development, we propose to study the Hawthorne effect of assessing developer contribution with repository mining-based metrics. In other words, based on the proposed suite of developer contribution metrics, we plan to study to what extent developers' behavior will change based on their awareness of being measured using certain metrics.

II. Repository Mining-based Metrics

In this section, we present the metrics from previous work that we have evaluated so far: code contribution, average complexity per method, introduced bugs, and bug fixing contribution. Code contribution is measured in terms of lines of code (LOC). This metric is the sum of added, changed and deleted LOC of each developer, ignoring comments and blank lines.

The average complexity per method is represented by two metrics that indicate the extent of the McCabe cyclomatic complexity [17] added or modified by a developer. The first complexity metric is the average complexity per added method, i.e., the sum of the cyclomatic complexities of all methods added by the developer divided by the number of added methods. The second complexity metric is the average complexity per changed method, i.e., the sum of cyclomatic complexity deltas (calculated based on the complexity before and after the change) of all methods changed by the developer divided by the number of changed methods. Our current implementation ignores negative deltas which occur when the complexity of a method was reduced as part of a change.

The introduced bugs metric quantifies the number of bugs introduced by each developer. To find bugs introduced by developers, our miner, which is based on Eyolfson et al.'s approach [20], identifies code fragments that had to be changed to fix a bug reported in the issue tracker system. After that, the algorithm identifies the last developer who changed or created that code fragment. We chose Eyolfson et al.'s approach because it is a well known algorithm to identify who introduced bugs. The last metric is bug fixing contribution, which quantifies, in percentage, the commits made by the developer in bug tasks, among the commits made by the rest of the team. We chose these metrics based on discussions with the development teams that participated in our preliminary study.

III. Research Method

The preliminary study was conducted at the Informatics Department (aka. SINFO[1]) of the Federal University of Rio Grande do Norte (UFRN) in Brazil, which is responsible for the development of large scale web information systems that provide support for the academic, human resources and asset management at the university. These systems are developed in Java and are currently used by and customized to twenty Brazilian federal universities, five federal education institutes and nine federal government agencies. Our study collected data of 12 weeks from the source code repository (SVN) and the issue tracker system (SINFO's own implementation). The time window was chosen since the developers are being evaluated by the software managers on a monthly basis.

We interviewed seven employees from SINFO: four software project leaders, one requirements team leader, one support team leader and one quality assurance team leader. We refer to the four project leaders as PL1 to PL4, and to the other three team leaders as TL1 to TL3. Every month, the project leaders (PL#) have to define a financial bonus for their software developers based on individual performance and participation in the project. When we asked them about difficulties in defining a developer's bonus, they reported that it is time-consuming, suffers from a lack of established evaluation criteria and a lack of justification, and that it currently does not consider social or psychological aspects. The process is very subjective, and project leaders are looking for more objective criteria.

The data collection was divided into two parts. The first one was the extraction of metrics from a twelve-week period related to the developers' contribution, which were shown to the leaders during the interviews. The second part of the data collection consisted of semi-structured interviews that were performed face-to-face with the leaders. The interviewer used a script with 20 main questions and 10 supporting questions[2]. The average time of the interviews was about one hour, and all of them were recorded for future transcription and analysis.

To analyze the interview data, we used Grounded Theory [18], a methodology to perform qualitative analysis in order to support data-based theories. In the data coding phase, we derived 38 codes divided into four categories: (i) participant background; (ii) current method of bonus definition; (iii) improvements to bonus definition approach; and (iv) usefulness and applicability of the metrics. In the writing memos phase, we used a bottom-up approach, in which we first examined the child-code segments followed by the parent-code fragments and so on.

IV. Preliminary results

In this section, we report our preliminary results regarding the metric-based contribution assessment.

A. Code Contribution

All leaders agreed that the code contribution metric is useful to evaluate developers' individual contribution. However, PL1, PL2 and TL3 pointed out that it must be used together with other information, such as the tasks finished by the developer and/or complexity-related metrics: "*Why does one developer produce one thousand LOC in one week, and in another week he produces just sixty-four? It's not that simple*" [TL3].

When asked about unexpected metric values of their developers, PL3 and PL4 were surprised because they did not know how to explain why two of their developers had high values for code contribution. This anecdote reinforces the idea of incompleteness of this metric and indicates that it has to be complemented with other information. In addition, TL3 said that the code contribution metric could be useful to instigate the project leader to investigate why there was a drop in the code production of a particular developer: "*Why did a developer have such high coding production level and now it's so low?*"

PL3 argued that the current implementation of the code contribution metric does not quantify contributions to non-Java artifacts. Thus, it may penalize front-end developers, which mostly contribute to HTML/JSP pages, CSS and JavaScript scripts. Another reported threat (PL4) is that developers that use modern technologies or techniques, such as generics or regular expressions which solve a problem with less coding, may be penalized. This threat drives us to question whether a developer with high code contribution indeed provides the best coding performance.

[1] http://www.info.ufrn.br

[2] Interview script available at http://jalerson.me/docs/dissertation-interviews-script.pdf

PL2 and TL3 argued that the metric should not be used isolated from other information, because of potential misinterpretations.

B. Average Complexity per Method

Complexity metrics were well received by our participants. PL2 argued that these metrics can help project leaders evaluate how complexity was distributed and identify developers who write unnecessarily complex code or need training: *"So we'll be able to see both sides: who is developing well and who needs technical support*". PL3 agreed: "*Higher complexity makes it harder to make some sort of adjustment, changes, a possible evolution of the system, so I think this is a good metric*".

To better understand the values of these metrics, PL3 suggested that the values should be accompanied by base values, such as the number of added and changed methods per developer as well as the cyclomatic complexity of the added and changed code: "*This metric value is X, but how many methods did he change?*" Similarly, PL1 suggested that these metrics should be accompanied by a list of tasks closed by the developer, to allow the project leader to evaluate the metric values and the complexity of the tasks.

C. Introduced Bugs

The feedback about the "introduced bugs" metric was mixed. Some of our participants argued that it is useful, whereas other leaders said that it could not be used isolated from other information or that too many threats hinder its interpretation. PL1 found that one of his developers, who he believed to have low bug rates (based on the number of errors associated with the developer) indeed had low values for this metric. "*So she developed entire new modules of the system. So it's really a confirmation of what I said, that she produces a lot of code but with a low error rate compared to other developers*".

According to TL1, TL2 and TL3, this metric may be useful to identify situations in which developers do not properly test their code, possibly caused by lack of time, as pointed out by TL1. TL1 also suggested that project leaders have to be aware of the number of tasks performed by the developer. TL3 argued that the project leader needs to take additional aspects into consideration, for example, what pressures the developer was faced with due to deadlines.

For TL3 and PL3, this metric should not be used in isolation, because this could lead to misinterpretation: "*[Specific developer] is a senior developer, and he has high values of introduced bugs, but he works on complex tasks, handling complex code [...] So this isolated metric can't help me, I need to see it associated with what he is producing*" [TL3].

The main threat of this metric, reported by six of the seven interviewed leaders after observing the metric values for the developers in their teams, was that the developers who had greater values for this metric were those who had more experience and had been on the project for longer. This metric tends to penalize the most experienced developers of the team, since they have produced more code compared to newer developers. Because of the way the algorithm identifies who introduced bugs, the experienced developers have a higher chance of being identified as "guilty" by the algorithm.

D. Bug Fixing Contribution

SINFO's development teams have bug fixing cells, composed of one or two developers who focus on fixing bugs. The project leaders confirmed that the bug fixing contribution metric indeed showed higher values for the developers who belong to these cells. According to PL1 and PL4, at certain times there is an increase in the occurrence of bugs, and project leaders tend to distribute these bugs among team members, instead of concentrating them in the bug fixing cells. PL1 commented that, in these situations, he can ask the members of these cells to contribute to other kind of tasks, for example, developing new features: "*That week I distributed the error tasks better, so this reinforces that he could have produced more*".

However, PL4 argued that this metric is not useful, since it only indicates who contributed most to bug fixing tasks, which is something that project leaders already know. PL1 argued that some tasks do not require coding, and since this metric only quantifies commits, these contributions would not be considered: "*Some tasks don't require coding [... for example] the developer produces a database fixing patch*". In addition, PL1 said that it is necessary to consider the complexity of the bug fixing tasks, and not only the number of tasks or commits.

For PL1 and TL3, the values of this metric should not be shown as percentage, because this promotes a comparison among team members which may not be fair, because it may favor developers in the previously mentioned bug fixing cells. For these leaders, the comparison should be made using absolute values and among bug fixing cells of different development teams: "*The development teams have bug fixing cells, so I have to compare among them*" [TL3].

TL2 presented two interesting situations for this metric. First, a bug task that was closed with many commits may be too complex and require a lot of effort, or it may result from a developer who has the habit to make many small commits. Second, a task that was closed with a single commit may be either simple or the result of a developer with the habit to only commit once when the task is finished. These two situations show that the number of commits is not a reliable attribute to measure the effort needed to finish the task, which is why TL2 asked: "*What is better? Is it better if the metric has high or low values?*"

E. Overall Benefits and Limitations

According to PL2, the metrics may help project leaders to analyze the software contribution of their developers in a technical way, since their analysis is currently done subjectively: "*We can see the productivity of the developer in terms of code, technique. These metrics could help so much*". In addition, PL3 argued that these metrics can help project leaders identify developers who need training: "*The manager will be able to define strategies to technically improve his team with training, aiming that these metric values become more homogeneous*".

PL3 also argued that SINFO's current approach to define the monthly bonus of their developers takes a significant amount of time, and after seeing our developer contribution metrics which can be generated periodically and automatically, he said that they will help reduce the evaluation time. For PL1, the metrics could be used as technical and objective criteria to justify why developers are receiving a particular bonus: "*Maybe just the number of tasks is not enough [to justify], but showing these numbers [of the metrics], maybe they could support me to justify*". TL1 also reported that the metrics could be useful to formalize the criteria to evaluate developers.

TL3 suggested that these metrics may complement the subjective evaluation of developers' contribution. By reading the historical values of the metrics, project leaders will be able to detect considerable variations, which may indicate that developers deal with personal or professional problems. For PL2 and TL3, the metrics should not be used in isolation, but combined with other metrics or information about the tasks performed by the developer: "*It's limited because it's an isolated number, but when you have several metrics, several indicators, then you start to complement [...] but even having several metrics, the personal side is lacking.*" [TL3].

V. Discussion and Future Work

These preliminary results confirm the need for a suite of metrics rather than one isolated metric. Aiming to build a better suite of software contribution metrics and based on the insights gained from our preliminary work, we plan to evaluate further metrics, especially considering communication, collaboration and task distribution. We intend to evaluate communication through e-mail messages and team communication platforms, for example, Slack, HipChat and Gitter. Collaboration is hard to measure because it is often not explicitly stored in an artifact. However, some software collaboration platforms such as GitHub, GitLab and BitBucket are useful to assess collaboration among developers, for example, by tracking comments and pull requests. Metrics are also valuable to keep project leaders aware of what is happening in their teams and what their developers are working on. For example, a useful awareness metric for project leaders is a set of percentage indicators that provide an overview of which kind of tasks their developers are working on during a certain time period.

According to our preliminary results, the metrics of code contribution and average complexity per method are best suited to support project leaders' evaluation of the contribution of their software developers, while bug-related metrics may not be applicable due to the previously reported limitations and threats. We plan to expand on this work by also asking developers about their impressions, and then comparing the feedback obtained from project and team leaders to that of the developers.

Another important avenue for future work is the evaluation of the impact of measuring contribution on developers' behavior, known as Hawthorne effect[3], i.e., the effect of individuals modifying an aspect of their behavior in response to their awareness of being observed. In terms of a research methodology, we plan to record metric values of developers for a certain period without the developers being aware of the measurement. In a second phase, the same developers will then be notified about what aspects of their contributions are measured, which will allow us to see the extent to which the Hawthorne effect changes developers' behavior. Understanding the effect caused by the specific metrics will enable us to fine-tune the metrics and to interpret them in context.

The idea behind studying the Hawthorne effect is related to gamification. Gamification concepts and techniques have been used in software development to motivate developers and increase productivity [19]. A key concept of gamification is the reward mechanism, which consists of awards given to developers that have accomplished certain goals. As part of this research agenda, we will investigate how the suite of contribution metrics can be used to reward developers as well as how the suite's implementation affects developer behavior and engagement. In addition, future work will include investigating ethical implications of using automated tools to measure employees' work.

VI. Limitations and Threats

Our preliminary work has some limitations and threats, which will partly be mitigated by our future work. Since the qualitative study reported in this paper was performed with seven project and team leaders of SINFO, we cannot claim that we covered all possible perspectives about individual contribution evaluation of software developers, and the results may be biased due to SINFO's bonus system for developers. In addition, we focused on feedback from project and team leaders rather than developers because it is usually individuals in leading positions who will make use of the numbers provided in a metrics suite. Because of this focus of our work, we cannot make claims about the viewpoints of developers at this point. Our evaluation is limited to a set of five metrics, and while we cannot claim anything beyond these metrics, this is the main goal of our future work, in particular focusing on human and social aspects, including communication, collaboration and task distribution.

VII. Related Work

Costa et al. [1] used repository mining-based metrics and process mining to evaluate a developer's contribution. Three metrics were used to assess contribution: commits that introduced bugs, code contribution, and fixed priority bugs. The study tried to identify which groups of software developers (core, peripheral and active) have the best values for these metrics. The authors conclude that the metric values of the developer groups have statistically significant differences and that the core developers have the greatest code contribution and introduce proportionally fewer bugs into the system. In addition, a new group of developers was identified, those with fewer introduced bugs compared to peripheral developers, but more bugs compared to the core developers.

Gousios et al. [14] presented an approach to measure the contribution of an individual software developer using a set of repository mining-based metrics. Some of these contributions may be considered positive actions, for example, fixing a bug or creating a new Wiki page, while others may be considered negative actions, for example, performing a commit without comments or introducing a bug into the system. The evaluation study of the proposed approach mined data from 48 GNOME subprojects. The authors collected data from code repositories and email

[3] http://www.economist.com/node/12510632

messages to produce 17 metrics, out of which 14 were considered positive and 3 negative.

Rastogi et al. [15] proposed a framework, called Samiksha, to evaluate the maintenance contribution of bug reporters, bug triagers, bug owners and collaborators using eleven metrics. Their study points out the importance of the attributes captured by the metrics as well as the lack of precision to capture and assess these attributes. Nagwani and Verma [16] presented a technique to rank developers based on their bug fixing contribution. This ranking is based on four metrics: number of fixed bugs, number of comments, number of reported bugs and number of messages received. An evaluation study was conducted using the Mozilla Bug Repository, with about a thousand bugs that were fixed by 97 developers. The study results presented two rankings with the developers who most contributed to bug fixing.

None of these papers report an evaluation of contribution metrics with the decision makers in a development team.

VIII. Conclusion

Assessing the contribution of developers in software projects is challenging, and while many metrics have been proposed for this purpose, few have been evaluated with the individuals who will use them in their decision making processes: project and team leaders. In this paper, we have proposed a research agenda to build a suite of contribution metrics based on empirical evidence collected from project and team leaders. In our preliminary work, we collected five metrics from 48 developers in four teams over a period of twelve weeks, and we showed the collected data to the project and team leaders to find out which metrics are suitable for contribution assessment. Code contribution and complexity metrics received the most positive feedback while bug-related metrics were seen more critically. Based on this preliminary work, we plan to expand our suite of contribution metrics, in particular by including metrics that assess communication, collaboration and task distribution.

Measuring contribution can have side-effects, both beneficial and undesirable. As part of this research project, we will also investigate the impact of measuring developers' contribution on the work of these developers. While positive effects may be enhanced, for example using gamification, there is a risk that developers focus on achieving certain metric values instead of building good software.

References

[1] D. A. da Costa. "Avaliação da contribuição de desenvolvedores para projetos de software usando mineração de repositórios de software e mineração de processos". Master's degree dissertation, 2013.

[2] C. Melo, D. Cruzes, F. Kon, F. Conradi. "Interpretative case studies on agile team productivity and management". Information and Software Technology, v. 55, n. 2, p. 412-427, 2013.

[3] B. Kitchenham, E. Mendes. "Software productivity measurement using multiple size measures". IEEE Transactions on Software Engineering, v. 30, n. 12, p. 1023–1035, 2004.

[4] G. S. de Aquino, S. R. L. Meira. "Towards effective productivity measurement in software projects". Fourth International Conference on Software Engineering Advances, p. 241-249, 2009.

[5] K. Petersen. "Measuring and predicting software productivity: A systematic map and review". Information and Software Technology, v. 53, n. 4, p. 317–343, 2011.

[6] A. Trendowicz, J. Münch. "Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences". Advances in computers, v. 77, p. 185–241, 2009.

[7] A. Maccormack, C. F. Kemerer, M. Cusumano, B. Crandall. "Trade-offs between productivity and quality in selecting software development practices". IEEE Software, v. 20, n. 5, p. 78–85, 2003.

[8] N. Ramasubbu, M. Cataldo, R. K. Balan, J. D. Herbsleb. "Configuring global software teams: a multi-company analysis of project productivity, quality, and profits". Proceedings of the 33rd International Conference on Software Engineering, p. 261-270, 2011.

[9] T. C. Jones. "Measuring programming quality and productivity". IBM Systems Journal, v. 17, n. 1, p. 39–63, 1978.

[10] B. W. Boehm. "Improving software productivity". Computer, 1987.

[11] A. S. Duncan. "Software development productivity tools and metrics". Proceedings of the 10th International Conference on Software Engineering, p. 41-48, 1988.

[12] C. Treude, F. F. Filho, U. Kulesza. "Summarizing and Measuring Development Activity". Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, to appear, 2015.

[13] A. N. Meyer, T. Fritz, G. Murphy, T. Zimmermann. "Software developers' perceptions of productivity". Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, p. 19-29, 2014.

[14] G. Gousios, E. Kalliamvakou, D. Spinellis. "Measuring developer contribution from software repository data". Proceedings of the 2008 International Working Conference on Mining Software Repositories, p. 129-132, 2008.

[15] A. Rastogi, A. Gupta, A. Sureka. "Samiksha: mining issue tracking system for contribution and performance assessment". Proceedings of the 6th India Software Engineering Conference, p. 13-22, 2013.

[16] N. K. Nagwani, S. Verma. "Rank-me: A java tool for ranking team members in software bug repositories". Journal of Software Engineering and Applications, v. 5, p. 255, 2012.

[17] T. J. Mccabe. "A complexity measure". IEEE Transactions on Software Engineering, n. 4, p. 308–320, 1976.

[18] J. Corbin, A. Strauss. "Basics of qualitative research: Techniques and procedures for developing grounded theory". Sage publications, 2014.

[19] D. J. Dubois, G. Tamburrelli. "Understanding gamification mechanisms for software development". Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, p. 659-662, 2013.

[20] J. Eyolfson, L. Tan, P. Lam. "Do time of day and developer experience affect commit bugginess?". Proceedings of the 8th Working Conference on Mining Software Repositories, p. 153-162, 2011.