

## Code and Commit Metrics of Developer Productivity: A Study on Team Leaders Perceptions

Edson Oliveira · Eduardo Fernandes ·  
Igor Steinmacher · Marco Cristo · Tayana  
Conte · Alessandro Garcia

Received: date / Accepted: date

**Abstract** *Context* – Developer productivity is essential to the success of software development organizations. Team leaders use developer productivity information for managing tasks in a software project. Developer productivity metrics can be computed from software repositories data to support leaders’ decisions. We can classify these metrics in code-based metrics, which rely on the amount of produced code, and commit-based metrics, which rely on commit activity. Although metrics can assist a leader, organizations usually neglect their usage and end up sticking to the leaders’ subjective perceptions only. *Objective* – We aim to understand whether productivity metrics can complement the leaders’ perceptions. We also aim to capture leaders’ impressions about relevance and adoption of productivity metrics in practice. *Method* – This paper presents a multi-case empirical study performed in two organizations active for more than 18 years. Eight leaders of nine projects have ranked the developers of their teams by productivity. We quantitatively assessed the correlation of leaders’ rankings versus metric-based rankings. As a complement, we interviewed leaders for qualitatively understanding the leaders’ impressions about relevance and adoption of productivity metrics given the computed correlations. *Results* – Our quantitative data suggest a greater correlation of

---

We thank the financial support from SEFAZ/AM, UFAM, CNPq via grants 430642/2016-4, 423149/2016-4, 311494/2017-0, 204081/2018-1/PDE, 465614/2014-0, 308380/2016-9 and 434969/2018-4, CAPES via grants 175956/2013, 175956, 117875 and 153363/2018-5, FAPERJ via grants E-26/200.773/2019, 102166/2013, 225207/2016, 211033/2019, 202621/2019, National Science Foundation #1815503. Finally, we also thank the participating organizations and their employees, and the support of USES Research Group members.

E. Oliveira, M. Cristo, T. Conte  
Federal University of Amazonas (UFAM), Brazil  
E-mail: {edson.cesar, marco.cristo, tayana}@icomp.ufam.edu.br

E. Fernandes, A. Garcia  
Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil  
E-mail: {emfernandes, afgarcia}@inf.puc-rio.br

I. Steinmacher  
Federal University of Technology – Paraná (UTFPR), Brazil, and Northern Arizona University,  
USA  
E-mail: igorfs@utfpr.edu.br

the leaders' perceptions with code-based metrics when compared to commit-based metrics. Our qualitative data reveal that leaders have positive impressions of code-based metrics and potentially would adopt them. *Conclusions* – Data triangulation of productivity metrics and leaders' perceptions can strengthen the organization conviction about productive developers and can reveal productive developers not yet perceived by team leaders and probably underestimated in the organization.

**Keywords** Developer productivity · Software metrics · Repository mining · Team leaders perceptions · Mixed method

## 1 Introduction

Developer productivity is essential to the success of development projects [2]. Highly productive developers are desired, especially for performing critical development tasks [7]. Therefore, the continuous productivity assessment of the development team is recommended. We recently observed that software organizations often rely on team leaders' perceptions to assess developer productivity [40]. However, perceptions are subjective and biased. Thus, a systematic productivity assessment could help team leaders in their work.

Team leaders' perceptions are usually the primary source of information when project managers make decisions about development teams [51,53]. Thus, team leaders perceptions have a direct and non-negligible effect on organization finance [28]. Team leaders participate in different activities in software projects, especially in monitoring and managing the development tasks within a team [51]. We recently also investigated how software project managers perceive developer productivity [40]. Our results indicate that project managers strongly depend on the team leaders perceptions to assess the productivity of their teams, which, in turn, build up their developers' productivity perceptions only through observations. Therefore, organizations usually end up overlooking opportunities for using productivity metrics.

Several current studies investigate productivity metrics focus on software productivity at the organization level [44], neglecting the analysis on the developer level [41]. Thus, the findings do not support tracking unproductive developers and managing development tasks in a particular software project. However, some recent studies [45,37,48] measured developers' productivity by relying on information stored in the software project repositories, exploring many productivity metrics in practical settings. That is the case of metrics that (1) are based on characteristics of the source code produced by developers during development time (e.g., [31]) — which here are called *code-based metrics* — and that (2) are based on the developers' commit characteristics in a software project (e.g., [37,48]) — which are called *commit-based metrics* in this paper.

The developer productivity metrics studied in this paper that focus on the characteristics of the source code are: Source Lines of Code by Time (*SLOC/Time*) [31], Halstead Effort by Time (*HalsteadEff/Time*) [26] and Code Owned by Time (*Code-Owned/Time*). The first metric (*SLOC/Time*) is based on the size of source code, meaning that the larger the source code created by the developer, the higher his/her productivity. The second metric (*HalsteadEff/Time*) is computed in terms of source code complexity, meaning the more complex the source code created

by the developer, the higher his/her productivity. Finally, the last metric (*Code-Owned/Time*) relies on the number of source code files owned by a developer in the software project, meaning that the more source code the developer owns, the higher his/her productivity. A developer owns a source code file when the developer is the one who most contributed to that source code file.

The developer productivity metrics studied in this paper that focus on the characteristics of the commits are: Commits Performed by Time (*Commits/Time*) [37], Committed Source Lines of Code by Time (*CommittedSLOC/Time*) [24] and Committed Characters by Time (*CommittedChars/Time*) [48]. The first metric (*Commits/Time*) is based on the number of commits performed by a developer, meaning that the higher the number of commits, the higher is his/her productivity. The second metric (*CommittedSLOC/Time*)—also known as Code Churn [39]—is computed considering the number of lines of code in each developer commit, meaning that the larger the commits created by the developer, the higher his/her productivity. Finally, (*CommittedChars/Time*) is calculated also considering the size of commits made by a developer, but counting the number of characters instead of the number of lines, also meaning the larger the commits, the higher his/her productivity. This last metric can reduce false positives in developer productivity, for example, by ignoring substantial number of source code lines that were only commented out by the developer.

In this paper, we present and discuss a multi-case empirical study aimed to investigate the relations between team leaders perceptions and developer productivity metrics. We hypothesize that existing metrics are sufficiently aligned with the team leaders perceptions of productivity. Although correlation does not imply causation, the differences between productivity metrics and leaders' perceptions would provide significant and complementary information about developers' productivity. This work relies on a mixed-method empirical study, combining quantitative and qualitative data. We quantitatively investigated whether six productivity metrics, selected from a previous research [41], correlate with the perceptions of eight team leaders from nine software projects. We computed Kendall's tau ( $\tau$ ) correlation coefficients for three code-based and three commit-based metrics. Also, we also interviewed team leaders regarding the relevance and potential of adoption of those six metrics. By using thematic analysis [12], we sought to capture team leaders impressions about the selected productivity metrics, investigating whether and why team leaders would adopt or not productivity metrics.

We relied on data collected from projects and experienced team leaders of two software development organizations. Our results revealed that team leaders perceptions are more correlated with code-based metrics than with commit-based metrics. Complementarily, our interviews revealed a major concern of leaders with commit-based metrics once the commit practices vary depending on the developer and project. Finally, team leaders have positive impressions of code-based metrics and potentially would adopt them.

The remainder of this paper is structured as follows. Section 2 provides background information. Section 3 describes our research method. Section 4 presents our study findings by organization and for all organizations together. Section 5 discusses our actionable study findings. Section 6 compares our study with previous work. Section 7 discusses threats to the study validity. Finally, Section 8 concludes the paper and suggests future work.

## 2 Background

This section discusses developer productivity metrics (Section 2.1) and, specifically, two categories of metrics that we use in this paper: code-based (Section 2.2) and commit-based (Section 2.3).

### 2.1 Developer Productivity Metrics

Productivity, in general, is the relationship between what is produced (output) and the resources applied (input) to produce it. This sense of productivity is widely used in many areas, from agriculture to economics [2]. However, for the service industries, such as Software Engineering, where products are often intangible, this concept does not apply adequately [27]. One very complex problem is to quantify the results produced in a software project in terms of size, complexity, or customer value [21].

In addition, several factors influence software productivity, making its measurement even more complicated. Although many of these factors are known since Boehm’s work with COCOMO [6] and COCOMO II [5], it is possible that more factors exist. Considering all of them for analysis would not be feasible [51]. In addition, when dealing with developer’s productivity, it is important to consider the human factors involved, which play a central role in software development [1]. The complexity of the topic makes it very hard to reach a consensus about how to measure software productivity—and more specifically developer’s productivity.

Despite all these difficulties, productivity is a critical success factor for software projects. To go further and improve our understanding about productivity, it is necessary to find ways measure it. Therefore, while there is still no consensus about how to measure productivity accurately, investigating it leveraging existing productivity metrics still provides information to advance our understanding and improve productivity measurement.

Existing literature reviews aggregate already proposed productivity metrics to measure productivity at various levels, such as organization, process, task, developer. Petersen [44], for example, conducted a systematic mapping exploring how to measure and predict software productivity. They considered only those studies that effectively evaluated the metrics, either by experiment, case study, or proof of concept. Of the 38 articles selected, only five presented developer productivity metrics, and none of them have been evaluated in the industry to assess productivity. The systematic literature review by Hernández-López et al. [27] focused on metrics for developer-level productivity. Of the six studies selected, the productivity metrics found were source lines of code (SLOC) per time and tasks per time. Finally, we also conducted a systematic mapping [41] of productivity metrics aiming to identify how researchers have measured productivity. We found that, to measure developer productivity, researchers mainly used time and effort as input measures and source lines of code as the output measure. From these three reviews, we observed that the number of productivity metrics used for assessing developer productivity is small, especially to investigate developer productivity in industry.

In the next subsections, we detail the developer productivity metrics chosen for this study. We explain the rationale for our choice in Section 3.2.

## 2.2 Code-based Metrics

Metrics extracted from the source code have been largely employed for assessing many aspects of software quality [38,43]. These metrics have been used in many automated tools aimed to guide organizations and team leaders in enhancing software projects [13,17]. Particularly, some metrics extracted from the source code can be a proxy for developer productivity [26,31]. Thus, in this study, these metrics are referred to as *code-based metrics*.

In a software project developed by a team of developers, each developer contributes by inserting their developed source codes into the project code repository. Thus, in order to calculate the individual productivity of developers based on the code-based metrics, it is first necessary to identify which source code belongs to which developer. Therefore, in this study, we identified the author of each source code file automatically, based on the work of Bird et al. [4]. As explained by Bird et al. [4], “ownership is a general term used to describe whether one person has responsibility for a software component” and, in our case, responsibility for a specific source code file. With Code Ownership, it is possible to identify the source code author (developer) automatically from the information of the project source code repository. Finally, we calculated the productivity of each developer by applying each code-based metrics to their source code set.

**Source Lines of Code by Time (SLOC/Time) [31]:** This metric is computed based on the number of Source Lines of Code (SLOC) [34] produced by a developer over time. The larger the source code files created by a developer along the project timeline, the higher is the developer productivity. There is a significant threat regarding coding habits when computing productivity with SLOC/Time. Let us suppose that two different developers (D1 and D2) produced equally critical source code files (F1 and F2, respectively). Although the developers are expected to have equal (or similar) productivity, in the case when the SLOC for F1 is higher than for F2, this productivity metric would point out D1 as more productive.

Conversely, SLOC and code maintainability seem correlated such that the smaller is SLOC, the easier is to read and modify the code [16]. Thus, if developers spend too much time to reduce SLOC towards a maintainable code, the developer productivity decays according to SLOC/Time. Nevertheless, SLOC/Time can alleviate that threat, and provide a more useful hint of developer productivity in cases when larger source code files tend to implement more critical features.

**Halstead Effort by Time (HalsteadEff/Time):** This metric is computed in terms of source code file complexity measures known as Halstead Effort [26]. Such measures rely on the number of operands and operators used in the source code, which supposedly reflects the difficulty level to read and understand the code. Thus, the higher the Halstead Effort of code produced by a particular developer along time, the higher is the developer productivity. Differently, from SLOC/Time, this metric considers the internal complexity of source code files instead of considering only their size, which alleviates the threat related to coding habits mentioned above.

**Code Owned by Time (CodeOwned/Time):** It represents the number of source code files authored (owned) by a developer over time. Recent studies have investigated code ownership as a means to infer how much code-related knowledge and expertise each developer has [4,25,50]. Code ownership is particularly interesting because it reveals key developers in a team, whose participation is essential to

the development of a given software project [19]. Basically, the higher the number of source code files owned by a developer over time, the higher their productivity. With this metric, we want to observe if developers' productivity is related to the number of source code files that belongs (code ownership) to this developer. We did not find previous studies on team leaders' perception of productivity metrics from this nature.

Besides, since the concept of code ownership is also used to compute the other code-based metrics to associate which files belong to which developers, we decided to investigate to what extent this metric reflects the team leaders productivity perceptions.

### 2.3 Commit-based Metrics

Recent studies extensively explored the use of metrics derived from the commit history of software projects (e.g., [8, 14, 55]). These metrics are used to understand software projects from various perspectives, from internal code structure [8] to external program behavior [55]. Because of that, several mechanisms have been proposed to support the collection and analysis of commit data [14]. Notably, researchers have used specific metrics extracted from commit history for assessing developer productivity [37, 48]. In this study, we refer to these metrics as *commit-based metrics*.

Just as the other code-based metrics, to calculate the commit-based metrics, we also needed to identify which commits are owned by which developer. The information related to each commit already provides the developer identification. Based on commit information, we calculated the productivity of each developer by applying each commit-based metrics to their commits in the project. Note that, in this case, we only used commits' information to calculate the developers' productivity, and did not consider the content of the source code files.

**Commits Performed by Time (Commits/Time) [37]:** This metric computes the number of commit operations performed by one particular developer over time. Shortly, the higher the number of performed commit operations, the higher is the developer productivity. Only using this metric as a hint of productive developers can be quite imprecise, once commit policies can vary [56]. Let us suppose that two different developers (D1 and D2) produced the same amount of relevant source code, but D1 performed a single commit operation against two or more commit operations performed by D2. In this case, D2 would be considered the most productive developers. Anyway, this metric can be useful in the case of projects whose developers follow strict commit policies.

**Committed Source Lines of Code by Time (CommittedSLOC/Time) [24]:** Each commit has a set of modified lines associated with it, which are frequently only small parts of source code files. This metric aims to capture the developer productivity using commit information in a more fine-grained than Commits/Time: considering not only the number of commits but also the number of lines associated with each commit. As an example, let us suppose that two different developers (D1 and D2) applied three and four commits operations, respectively in the last four months. D1 applied commits  $c_{1,1}$  after modifying 200 SLOC,  $c_{1,2}$  after modifying 200 SLOC, and  $c_{1,3}$  after modifying 400 SLOC. Thus, CommittedSLOC/Time for D1 equals  $(200 + 200 + 400)/4 = 200$ . D2 applied commits  $c_{2,1}$ ,  $c_{2,2}$ ,  $c_{2,3}$ , and

c<sub>2.4</sub> after modifying 100 SLOC by commit. Thus, CommittedSLOC/Time for D2 equals  $(100 + 100 + 100 + 100)/4 = 100$ . As a conclusion, D1, despite having performed fewer commits in four months, was more productive than D2, which had performed more commits in the same four months. CommittedSLOC/Time is also known as Code Churn [39].

**Committed Characters by Time (CommittedChars/ Time) [48].** Similarly to CommittedSLOC/Time, this metric provides a more fine-grained measurement of productivity by commit. CommittedChars/Time counts the number of characters modified in source code files committed by a developer, as the developer often only changes a small number of chars of a source code line in a commit operation. We computed the modifications at the character level via Levenshtein’s edit distance [32]. This metric can significantly reduce false positives in the number of source code files modified across commits. Here is one practical example: in popular languages such as C++ and Java, developers may comment on existing lines of code by adding only a unique special character per line. Modern code editors facilitate this kind of modifications and, therefore, it does not require too much effort from developers. In this case, the metric will capture only a minimum change, and it will count little to the developer productivity.

### 3 Research Method

This section describes our research method as follows. Section 3.1 introduces our study goal and research questions. Section 3.2 justifies our developer’s productivity metrics selection. Section 3.3 presents our study steps and procedure. Section 3.4 overviews our data set, including the organizations that participated in our study, a team leader characterization, and other collected data.

#### 3.1 Study Goal and Research Questions

We systematically defined our study goal based on the Goal Question Metric (GQM) framework [3] as follows: *analyze* the team leaders’ perceptions of developer productivity; *for the purpose of* understanding to what extent the team leaders’ perceptions match the developer productivity computed with the support of existing metrics; *with respect to* (1) the correlation of team leaders’ perceptions and developer productivity metrics, and (2) the team leaders’ impressions about the relevance and adoption of productivity metrics in practice; *from the viewpoint of* software engineering researchers and development team leaders; and *in the context of* team leaders from software organizations active for decades. To achieve our goal, we designed a multi-case study [47] based on two research questions (RQ<sub>s</sub>), as follows.

**Research Question 1 (RQ<sub>1</sub>):** *Do developer productivity metrics correlate with team leaders’ perceptions of developer productivity?* – The investigation of the correlation between productivity metrics and team leader perceptions can quantify to what extent productivity metrics correlate with leaders’ perceptions of their developer’s productivity. If there is any degree of positive correlation, although correlation does not imply causality, one could assume that both leaders’ perceptions and productivity metrics share a common ground. Triangulating these

quantitative results with the qualitative results of the previous question can show us which productivity metrics can be recommended to complement the leader's view of their developers' productivity.

**Research Question 2 (RQ<sub>2</sub>):** *How do team leaders perceive the relevance and adoption of developer productivity metrics in practice?* – We have designed RQ<sub>1</sub> for a better understanding of what team leaders think about existing productivity metrics. We are interested in understanding to what extent developer productivity metrics can help the team leaders. Therefore, if the rankings presented by the developer productivity metrics are very different from the leaders' perceptions, these metrics cannot contribute to them. However, if the metrics' rankings make any sense, then the disagreeing points could mean some factor that the metric does not take into account or something that the leader has not noticed yet about their developers. Thus, with the possibility of complementing the leader's perception, we could recommend the adoption of developer productivity metrics.

### 3.2 Selecting Developer Productivity Metrics

As presented in Section 2.1, the idea of productivity in software development is a historically complex problem [2]. Nevertheless, for a measurement of productivity to be effective in an organization, the organization needs to define its own measurement of productivity [27] that is aligned with its objectives [15], in order to carry out benchmarking of their data. However, participating organizations did not have any formally defined developer productivity metrics. Therefore, we initially sought to investigate how organizations identify the productivity of their developers.

**How do software managers identify the developer's productivity?** We conducted a case study in three software organizations [40]. As one of the study's results, we found that software managers identify which developers are productive through (1) the deliveries resulting from the developer tasks, (2) the developer's team feedback, and (3) the developer's behavior. This team feedback comes from meetings with the development team and, more specifically, from the team leader. Therefore, we decided to investigate how the team leader identifies the productivity of their developers.

**How do team leaders identify the developer's productivity?** To answer this specific research question, we returned to participating organizations to ask the team leaders how they identify the productivity of their developers. The team leaders answered that they do it by observing (1) the characteristics of their deliveries (from their programming tasks), (2) the feedback from the developer themselves and their team members, and (3) their behavior and attitudes. Based on this information, we decided to collect the data to measure productivity.

**Data collection in the participating organizations.** We kindly requested participating organizations to have access to the data from the projects led by the interviewed leaders. We asked permission to interview the developers and use the history of their tasks and code repository data to measure their productivity. Unfortunately, the organizations denied access to developers and did not make data available from their internal task tracking systems. They only granted us a copy of the source code repositories.

**How have Software Engineering researchers measured developer productivity?** Unable to measure developer productivity based on team leaders' point of

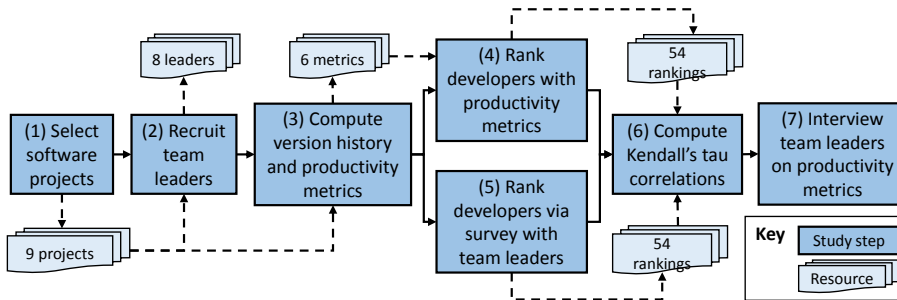


view, we decided to investigate in the literature how researchers had been measuring the software productivity [41]. While identifying primary studies in our systematic mapping, we found two other literature reviews, with different scopes, but also related to developer productivity metrics (Section 2.1). With the results of these literature reviews, containing the point of view of the researchers, we began to select the developer productivity metrics to be used in this work.

**Selecting developer productivity metrics.** Based on these reviews, we chose the most widely used metrics that could extract the developer’s productivity from the data contained in the software repositories. Thus, the first selected metrics, based on the characteristics of the source code files, were: *SLOC/Time* and *HallsteadEff./Time*. Motivated by the strategy used, explained below, to identify the authors of the source code, we introduced the *CodeOwned/Time* metric. Besides, we continued to monitor articles that assessed developer productivity from source code repositories. Thus, we find the work of Scholtes et al. [48], also referencing other works, which presented us developer productivity metrics based on commit characteristics: *Commit/Time*, *CommittedSLOC/Time* and *CommittedChars/Time*.

### 3.3 Study Steps and Procedure

Figure 1 illustrates the seven steps designed for guiding our mixed-method empirical study. We introduce and discuss each study step as follows.



**Fig. 1** Steps of our mixed-method empirical study

**Step 1: Select software projects.** In this step, we selected the software projects for collecting and analyzing data. Because we intended to perform a multi-case study [47], we needed to analyze two or more different projects. For this purpose, we have contacted as many companies as possible to ask for authorization to analyze their projects, contact the team leaders, and compute developer productivity (anonymously). By selecting multiple projects, we expected to achieve comprehensive study findings. As a result, we could analyze data of nine software projects from two different organizations. Each organization holds at least 18 years of activity in the software industry.

**Step 2: Recruit team leaders.** In this step, we recruited team leaders to participate in our interviews and support us in computing the productivity rankings. We kindly asked the managers of the nine projects selected in **Step 1** to indicate

team leaders that could engage with our study. As a result, we were able to recruit eight team leaders for the nine projects selected (*i.e.*, one leader was responsible for two projects at the time of our study execution).

**Step 3: Compute version history and productivity metrics.** We mined the software repository from the nine projects. The first author relied on automated scripts for computing the metrics. We calculated the productivity metrics for each of the 68 developers who participated in any of the nine selected projects. More information about the collected data is presented in Section 3.4.

**Step 4: Rank developers with productivity metrics<sup>1</sup>.** We ranked the developers according to their productivity for each software project. We computed the developer rankings based on each of the six metrics presented in Section 2. For each code-based and commit-based metric, we calculated the ranked list of developers from the most productive to the less productive. As a result, we obtained six developer rankings by software project.

**Step 5: Rank developers via a survey with team leaders<sup>1</sup>.** This step consisted of surveying team leaders regarding the developer productivity within their teams. We asked each team leader to sign a consent form for allowing us to use the data. Thus, through a short survey, we asked the team leaders to rank the developers of their respective teams from the most productive to the less productive. At the survey-filling time, we isolated team leaders so that they could not inter-communicate, or even consult any resources about their projects and development teams.

**Step 6: Compute Kendall’s tau correlations.** This step consisted of computing correlations for the metric-based rankings computed in **Step 4** with the rankings produced by team leaders in **Step 5**. We computed Kendall’s tau ( $\tau$ ) rank correlations [30]. The Kendall rank correlation coefficient evaluates the degree of similarity between two ordered sets (the rankings) on the same objects (the developers). This degree of similarity depends on the number of different pairs between these two ordered sets. A Kendall coefficient of  $-1$  means the largest possible number of different pairs, obtained when one ranking is the exact reverse of the other ranking, and a coefficient of  $+1$  corresponding to the smallest possible number of different pairs (equal to 0), obtained when both rankings are identical.

Our choice for this particular non-parametric test relied on three points: (1) both of our variables are of ordinal type (rankings), ruling out the Pearson’s correlation coefficient parametric test as it requires interval data for both variables; (2) Kendall tau works better with small samples (our case) than Spearman’s rank correlation coefficient, a more popular non-parametric correlation [20], as it “approaches a normal distribution more rapidly than Spearman, as  $N$ , the sample size, increases” [23]; and finally (3) “the Kendall correlation measure is more robust and slightly more efficient than Spearman’s rank correlation, making it the preferable estimator from both perspectives” [11]. To understand the strength of correlation results, we used the interpretation guidelines provided by Cohen [10].

**Step 7: Interview team leaders on productivity metrics.** Our last step consisted of interviewing the same team leaders that computed rankings in Section 5. We first explained the six productivity metrics described in Section 2 to the team leaders. After that, we presented the ranking that he/she had provided

---

<sup>1</sup> Plots of the joint distributions between calculated metric values and rankings (metric-based and leader-informed) are available online at <https://doi.org/10.5281/zenodo.3534258>

and the rankings that we computed using the six metrics. Next, we conducted a semi-structured interview following the Runeson’s guidelines [46]. This interview consisted of two high-level questions. **Question 1:** *Do these metrics make sense to you?* – We aimed to explore the extent in which team leaders with the relevance of the six productivity metrics. **Question 2:** *Would you adopt any of these metrics?* – Our goal was capturing the potential adoption of productivity metrics by team leaders in practice. We analyzed the data by performing a thematic analysis [12].

### 3.4 Data Set Overview

Table 1 describes the two organizations that participated in our empirical study. **Organization 1** provides non-profit and private support to hardware and software development. **Organization 2** is a governmental organization that provides systems to support public education, healthcare, and other domains. Both organizations kindly agreed to participate in the study anonymously.

**Table 1** Organizations that participated in this study

Characteristic	Organization 1	Organization 2
Years of activity	18	46
Total of employees	280	392
Social nature	Non-profit, private	Government, public
Software project type	Desktop, web, mobile*	Desktop, web, mobile*
Software project domain	Healthcare, industry automation	Public assistance (various)
Development process	Agile (prescriptive)	Agile (prescriptive)
Process certification	Mps.Br level F	ISO-9001
Software technologies	Java, JavaScript, Python	Java, NATURAL, PHP

\*Projects for both Google Android and Apple iOS platforms.

These organizations have their development processes certified by external certifying entities. Organization 1 has its development process certified as an Mps.Br [54] level F. This certification is equivalent to CMMI level 2 [9,18]. Organization 2 has received an ISO 9001:2015 [29] certification for all its internal processes, including their software development process.

The development teams of these organizations use variations of an agile SCRUM-based model to manage their development tasks. In these models, development sprints ran between 1 and 2 weeks, starting with the team leader’s allocation of development tasks to each team developer member. Developers are responsible for record in the internal task tracking system (developed internally by each organization) the tasks they will be developing. At the end of task execution, developers upload their commits to the central code repository server and record in the task tracking system the end of their implementation. Each organization has a quality team that monitors and evaluates process execution using data from these task tracking systems. As already mentioned, data from these systems were not available for this research.

In these organizations, all developers work in the same physical location, organized by projects, in which the developers from the same project are very close to each other (practically side by side). When a new software project starts, there is always a reorganization of the developers’ position, to privilege this proximity.

This positioning strategy is intended to facilitate communication and collaboration among developers in the same project. Regarding software testing, the two organizations differ in the way they work. Organization 2 has a dedicated testing team for each project, which is responsible for the preparation and execution of test plans. Project developers in Organization 1 are in charge of planning and execute the software testing themselves.

As far as team communication and developers' capability and experience are concerned, we could observe that these two organizations are very similar. All the developers of these organizations have a degree in Computer Science. The development teams in both organizations are designed to combine more experienced with less experienced developers. This allocation and physical proximity allow the exchange of knowledge and experiences through intensive team communication. For example, although these organizations do not have formal code review mechanisms, the physical proximity between the developers enables more experienced developers to review the code by pairing with their less experienced peers before submitting it to the version control system. This collaborative practice is expected to improve the quality of reviews [42]. Besides, the development teams have also other communication mechanisms (e.g., Slack<sup>2</sup>), in addition to the task tracker mechanisms mentioned previously.

Although the data from these organization's internal task tracking systems were not available, they made the code repositories available. While the low number of data points is a limitation of this study, this unique dataset it is not easy to obtain. Table 2 provides general data of team leaders per software project obtained and organization. The second column lists the projects lead by each of the team leaders at the time of our empirical study. The third column identifies each team leader to keep them anonymous. The fourth and fifth columns present age and the number of years that the team leaders had been working in the organization. Almost all team leaders had a degree in Computer Science, except for L8, who has a degree in Electrical Engineering but worked as a software developer for their entire career.

All team leaders work for their organizations for at least three years (about nine years, on average). Thus, they are quite familiar with development activities and processes employed within their organizations. Before being leaders, all of them worked as developers of the organizations, having experience working with most other developers. These team leaders develop their leadership activities in their respective software projects, sharing the same physical environment with their developers. They actively participate in development because, in addition to allocating tasks, working to help to solve some development problems faced by their developers.

Table 3 summarizes the version history data computed for all nine projects analyzed. We have calculated some necessary information to support our discussions: numbers of developers that contributed to the project development (second column), number of commit operations applied along the project history (third column), number of source code files that constitute the project (fourth column), percentage of files implemented in the predominant programming language, i.e., Java for all projects except P7 (fifth column), and months of project development. P6 and P8 were already in production (in maintenance phase); the other projects

---

<sup>2</sup> <http://slack.com>

**Table 2** Team leaders allocated by organization and software project

Organization	Project	Team Leader	Age	Years of Work
Organization 1	P1	L1	29	7
	P2	L2	44	8
	P3	L3	34	8
	P4	L4	30	3
	P5	L5	27	6
Organization 2	P6	L6	33	10
	P7, P8	L7	32	6
	P9	L8	40	26

were not deployed to production at the time of the study. As it is possible to observe, the ratio commits/month is higher for P1 to P5 (Organization 1) than P6 to P9 (Organization 2). A possible explanation is that Organization 1 adopted weekly sprints while Organization 2 adopted bi or triweekly sprints, and Organization 2's projects have more developers than Organization 1's.

**Table 3** Version history data by software project

Project	Developers	Commits	Source Code Files Absolute	Files %*	Development Months
P1	18	7,894	964	100.0	8
P2	10	4,052	1,494	99.9	6
P3	8	7,127	1,463	99.1	12
P4	7	4,474	512	94.6	11
P5	5	2,308	681	99.8	7
P6	7	7,011	1,788	100.0	73
P7	5	780	512	100.0	6
P8	4	777	241	98.8	76
P9	4	196	512	95.0	11

\*At least 90% of source code files implemented in Java or PHP (project P7)

## 4 Study Results

This section reports the results of our mixed-method empirical study. Section 4.1 present the correlations between team leaders' perceptions and the productivity metrics presented in Section 2. Section 4.2 presents the results of the interviews concerning the importance and adoption of productivity metrics from the team leaders' perspective. All identifiers for organizations, software projects, and team leaders used in this section are inherited from Tables 2 and 3.

### 4.1 Correlation of Metrics and Team Leaders Perceptions

**Research Question 1 (RQ<sub>1</sub>):** *Do developer productivity metrics correlate with team leaders' perceptions of developer productivity?*

Table 4 presents Kendall's  $\tau$  correlation coefficient by software project under analysis. The first column lists the six developer productivity metrics selected for the study. The following nine columns present the correlation coefficients for

projects P1 to P9 regardless of the software development organization. Each coefficient is annotated with \* and \*\* whenever they were statistically significant with  $\alpha = 0.10$  (i.e., 90% confidence level) and  $\alpha = 0.05$  (i.e., 95% confidence level), respectively. The table data shows that code-based metrics presented statistical significance in 11 cases for the most strict confidence level (i.e.,  $\alpha = 0.05$ ), against only 6 cases regarding the commit-based metrics.

**Table 4** Kendall’s  $\tau$  correlation coefficients by software project

Metric	Software Project								
	P1	P2	P3	P4	P5	P6	P7	P8	P9
CodeOwned/Time	0.57**	0.38	0.64**	0.71**	0.00	0.14	1.00**	0.55	0.33
HalsteadEff/Time	0.50**	0.42*	0.29	0.62**	0.80**	0.52*	0.80**	0.55	0.67
SLOC/Time	0.57**	0.47*	0.36	0.81**	0.40	0.43	1.00**	0.55	0.33
Commits/Time	0.31*	0.51**	0.07	0.14	0.40	0.59*	-0.2	0.00**	1.00**
CommittedChars/Time	0.48**	0.29	0.29	0.05	0.20	0.59*	-0.2	0.00**	0.00
CommittedSLOC/Time	0.32*	0.24	0.36	0.24	-0.2	0.59*	0.20	0.00**	0.00

\*  $\alpha = .10$ , \*\*  $\alpha = .05$

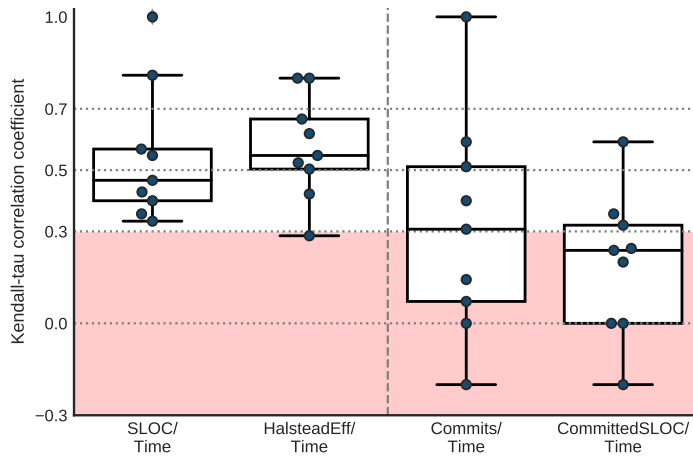
Although the metrics used in this study use different ways to calculate developer productivity, each group of metrics shares a common strategy for their calculation using either the source code files characteristics or the commits characteristics (Sections 2.2 and 2.3). Therefore, it is possible that these metrics intercorrelate with each other, turning the analysis of strongly correlated metrics unnecessary. CodeOwned/Time and CommittedChars/Time metrics are strongly positively correlated (Table 5, in red) with the other code-based and commit-based metrics, respectively. Thus, to simplify the presentation of results, these metrics will be omitted from the next tables and figures in this section.

**Table 5** Developer productivity metrics intercorrelation

Metrics	M1	M2	M3	M4	M5	M6
CodeOwned/Time (M1)	-	-0.04	<b>0.77</b>	-0.71	-0.54	0.20
HalsteadEff/Time (M2)	-0.04	-	0.44	-0.01	-0.55	-0.58
SLOC/Time (M3)	<b>0.77</b>	0.44	-	-0.68	-0.54	0.08
Commits/Time (M4)	-0.71	-0.01	-0.68	-	0.36	-0.05
CommittedChars/Time (M5)	-0.54	-0.55	-0.54	0.36	-	<b>0.59</b>
CommittedSLOC/Time (M6)	0.20	-0.58	0.08	-0.05	<b>0.59</b>	-

Figure 2 provides a general data view of Table 4 using boxplots, presenting with a dot each correlation for all combinations of project and metric. The x-axis lists the six developer productivity metrics selected. The y-axis indicates the Kendall’s  $\tau$  correlation coefficients per project. We have adopted the assessment guidelines presented by Cohen in his work [10] for understanding the correlation power of productivity metrics and team leaders’ perceptions:  $0.3 \leq \tau < 0.5$  indicates moderate correlation, and  $\tau \geq 0.5$  indicates a strong correlation. This figure also presents a red background for correlations with less than moderate strength ( $\tau < 0.3$ )

From this guideline, we can draw the following observations. The code-based metrics medians were strong and moderate, wherein only one project (P3) did not get a moderate or strong correlation (HalsteadEff/Time). The commit-based



**Fig. 2** Distribution of Kendall's  $\tau$  correlations for all projects

metrics medians were moderate and weak. The medians of commit-based metrics were weak and moderate, wherein the moderate median was very close to the lower bound of a moderate correlation. Overall, the productivity code-based metrics had a smaller variance than commit-based metrics. These results suggest that, in general, the analyzed code-based metrics (at least partially) can better reflect the team leaders' perceptions of developer productivity

Table 6 ranks the productivity metrics analyzed by median correlation coefficient. The first column lists all productivity metrics. The second and third column presents the median correlation coefficients computed strictly based on coefficients validated with  $\alpha = .05$  and  $\alpha = .10$ , respectively. The fourth column presents the median correlation based on all coefficients regardless of the statistical significance. The ranking of the fifth column followed this sorting precedence: second, third, and fourth columns. Each coefficient is followed, in parentheses, by the rate of projects (out of the nine projects) included in the median computation. As a result, the top-two of metrics is represented by code-based metrics solely.

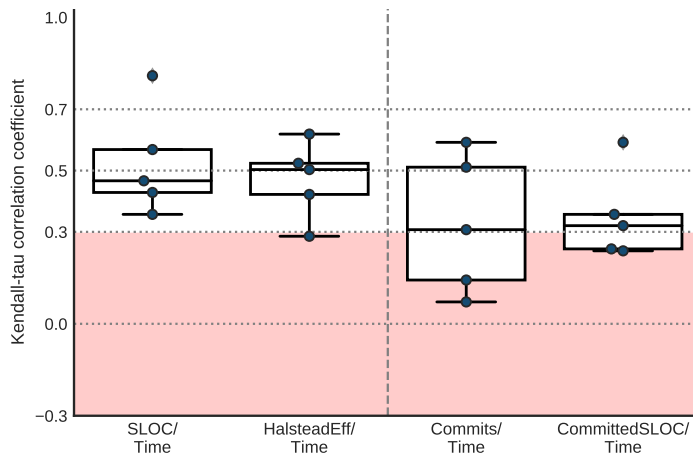
**Table 6** Ranking of productivity metrics by median correlation

Metric	Median Correlation Coefficient			Rank
	$\alpha = .05$	$\alpha = .10$	All	
SLOC/Time	.81 (3/9)	.69 (4/9)	.47 (9/9)	1 <sup>st</sup>
HalsteadEff/Time	.71 (4/9)	.57 (6/9)	.55 (9/9)	2 <sup>nd</sup>
Commits/Time	.51 (3/9)	.51 (5/9)	.31 (9/9)	3 <sup>rd</sup>
CommittedSLOC/Time	.00 (1/9)	.32 (3/9)	.24 (9/9)	4 <sup>th</sup>

*Considering projects with seven or more developers* – Some of the projects obtained have a small number of developers. Nevertheless, the probability of a leader ranking being perfectly correlated (equal rankings) to that of a metric in a 4-developer project is 1 in 24 possibilities, increasing to 1 in 120 in a 5-developer project. Still, the variation of the correlation coefficient value is small, i.e., small changes in

rankings have a significant effect on the correlation coefficient; for example, with a single change, the correlation coefficient may change from 0 (no correlation) to 0.33 (moderate correlation). This variation may represent a noisy in the data analysis. Therefore, we decided to analyze only the projects with seven or more developers, making the chance of having two equal rankings at least 1 in 5040.

Figure 3 presents, via boxplots, the distribution of correlation coefficients by metric for the five software projects (P1, P2, P3, P4, and P6). As before, the x-axis lists the developer productivity metrics selected; the y-axis indicates the Kendall's  $\tau$  correlation coefficients per project. Overall, the variance of all metrics decreased, and in commit-based metrics, this decrease was more pronounced. Some correlations at the extremes of metric distributions disappeared, suggesting confirmation of the idea of noise in data analysis with projects with few developers. Negative correlations and the perfect correlation of commit-based metrics are gone.



**Fig. 3** Distribution of Kendall's  $\tau$  correlations for projects with seven or more developers

Commit-based metrics showed a strong correlation in only one project (P6). In most projects, code-based metrics correlated more strongly than commit-based metrics. In general, the analysis previously described (with all projects) remained when analyzing projects with seven or more developers separately. Finally, it is worth mentioning that our results for these projects achieve statistical significance with  $\alpha = 0.10$  for at least one code-base metric per project.

Table 7 ranks the productivity metrics based on the median correlation coefficient, similarly to Table 6. The top-two of metrics kept composed by code-based metrics only. The different between rankings is a little variation in the correlations median value of all metrics, with a decrease for the code-based metrics, and a slight increase for the CommittedSLOC/Time metric. The Commits/Time metric has not changed.

*Considering projects with ten or more developers* – We now focus only on the two biggest projects in our dataset. These projects (P1 and P2) have ten and eighteen developers, in which the probability of a leader ranking being equal to that



**Table 7** Ranking of productivity metrics by median correlation

Metric	Median Correlation Coefficient			Rank
	$\alpha = .05$	$\alpha = .05$ or $.10$	All	
SLOC/Time	.69 (2/5)	.57 (3/5)	.47 (5/5)	1 <sup>st</sup>
HalsteadEff/Time	.56 (2/5)	.51 (4/5)	.50 (5/5)	2 <sup>nd</sup>
Commits/Time	.51 (1/5)	.51 (3/5)	.31 (5/5)	3 <sup>rd</sup>
CommittedSLOC/Time	N/A (0/5)	.46 (2/5)	.32 (5/5)	4 <sup>th</sup>

N/A: Not applicable (division by zero)

calculated by the metric is one in 3,628,800 and one in 6,402,373,705,728,000, respectively. Therefore, by reducing, even more, the noise in the data, we can have more confidence in interpreting the results, even though the sample consists of only two projects.

Table 8 lists the correlation coefficients by productivity metric for these two projects. This table is a cut from Table 4, focusing only on the biggest projects and including the number of developers. At the time of data gathering, these two projects were new developments (started in less than a year).

**Table 8** Kendall's  $\tau$  correlation coefficients for software project with ten or more developers

Metric	Software Project	
	P1 (18 devs.)	P2 (10 devs.)
HalsteadEff/Time	0.50**	0.42*
SLOC/Time	0.57**	0.47*
Commits/Time	0.31*	0.51**
CommittedSLOC/Time	0.32*	0.24

\*  $\alpha = .10$ , \*\*  $\alpha = .05$

Concerning the correlation distribution of projects with seven or more developers (Figure 3), the correlations of these two projects (P1 and P2) are the central points: project P1 correlation is the median for metrics HalsteadEff/Time, Commits/Time and CommittedSLOC/Time; while project P2 correlation is the median for metric SLOC/Time. In addition, Code-based metrics were the only ones to get a strong correlation ( $\tau \geq 0.5$ ) for the biggest project (P1), while Commits/Time, a commit-based metric, was the only one to get a strong correlation for project P2. The CommittedSLOC/Time was the only metric that did not get at least a moderate correlation ( $0.3 \leq \tau < 0.5$ ) for both projects, and still the only one to have a weak correlation ( $\tau < 0.3$  for project P2). Table 9 ranks the productivity metrics by correlation coefficient median.

**Table 9** Ranking of productivity metrics by median correlation

Metric	Median Correlation Coefficient			Rank
	$\alpha = .05$	$\alpha = .10$	All	
SLOC/Time	.57 (1/2)	.52 (2/2)	.52 (2/2)	1 <sup>st</sup>
Commits/Time	.51 (1/2)	.41 (2/2)	.41 (2/2)	2 <sup>nd</sup>
HalsteadEff/Time	.50 (1/2)	.46 (2/2)	.46 (2/2)	3 <sup>rd</sup>
CommittedSLOC/Time	N/A (0/2)	.32 (1/2)	.28 (2/2)	4 <sup>th</sup>

N/A: Not applicable (division by zero)

When data in Table 9 is compared with the previous metrics rankings (Table 7 and Table 6), the main difference is: Commits/Time outperformed HalsteadEff/Time by a minimal margin (0.01). However, despite this latest Commits/Time metric result, code-based metrics correlated most strongly with the team leaders' perceptions.

**Summary of RQ<sub>1</sub>:** Code-based metrics outperformed commit-based metrics, in reflecting team leaders' perceptions of developer productivity. Moreover, commit-based metrics showed even negative correlations, indicating an inverse rank concerning the team leader's rank.

## 4.2 Analysis of Interviews with Team Leaders

**Research Question 2 (RQ<sub>2</sub>):** *How do team leaders perceive the relevance and adoption of developer productivity metrics in practice?*







According to their perceptions, eight leaders estimated the developer ranking of their team. We interviewed these leaders, showing the ranking that he/she had provided and the rankings that we computed using the developer productivity metrics. Thus, this section presents the interview data involving the same eight leaders. In the sequence, we discuss our qualitative results based on the two interview questions defined in Section 3.3 (Step 7).

### 4.2.1 On the Relevance of the Six Productivity Metrics

After explaining each of the six productivity metrics selected to the team leaders, we showed the metric-based rankings to the respective team leaders. We then asked **Question 1** to each team leader: *do these metrics make sense to you?*

By analyzing the leaders' responses based on the procedures documented in Section 3, we captured both positive and negative perceptions of each productivity metric. We refer to answers similar to "I think that this metric makes sense in the context of developer productivity assessment" as positive perceptions. The opposite applies to negative perceptions. Table 10 summarizes both positive and negative perceptions of the team leader (L1 to L8). In the table, we marked with \* the metrics that presented the highest Kendall's  $\tau$  correlation coefficient for a given team leader (according to data of Section 4.1).

**Table 10** Positive and negative perceptions of productivity metrics by team leader

Metric	Team Leader								Total		
	L1	L2	L3	L4	L5	L6	L7	L8	+	-	
CodeOwned/Time	+*	+	+*	+	-	+	+*	+		7	1
HalsteadEff/Time	-	+	+	+	+*	+	+*	+		5	1
SLOC/Time	+*	-	+	*	+	+	+*	-		5	1
Commits/Time	-	*	-	-	-	*	-	-*		0	5
CommittedChars/Time	-	-	-	-	-	*	-	-		0	5
CommittedSLOC/Time	-	-	-	-	-	*	-	-		0	5

Positive perception (+); Negative perception (-); Greatest  $\tau$  correlation coefficient (\*)

The data presented in Table 10 suggests an overall negative perception about commit-based metrics: none of these metrics shown at least a single positive comment. Conversely, the majority of team leaders perceived the code-based metrics as positive to the developer productivity assessment. Especially, CodeOwned/Time was pointed out as positive by seven out of the eight team leaders interviewed. Additionally, half of the team leaders (L1, L3, L5, and L7) perceived positively the productivity metrics whose correlation coefficients best fit their perceptions. Three leaders (L2, L4, and L6) did not point out as positive the metrics whose correlations best fit their perceptions. Curiously, L8 had a negative impression of Commits/Time, though this was the only metric whose correlation best fit his perception. We hypothesize that the unfamiliarity with productivity metrics may have made leaders reluctant to reveal their positive perceptions on these metrics.

**Criticism regarding code-based metrics** – L1 and L5 reported criticism about using code-based metrics to assess developer productivity. L1 disagreed that HalsteadEff/Time can successfully capture the complexity of produced source code, while L5 pointed out that CodeOwned/Time does not always reflect the developer productivity. The excerpts representing their perspectives are presented below:

*I do not know if HalsteadEff/Time can measure complexity of the produced code by simply counting operands and operators.* – L1

*I believe that CodeOwned/Time does not reflect well the reality. In fact, the top-one developers should not be there.* – L5

**Criticism regarding commit-based metrics** – L3, L4 and L8 expressed their criticism about the productivity assessment enabled by commit-based metrics, such as Commits/Time. Particularly, L8 suggested that depending on the commit policy adopted by the developer, commit frequency says little about developer productivity:

*“I found these commit-based metrics irrelevant because, sometimes, developers commit their source code simply to not lose it; it has nothing to do with developer productivity.”* – L8

**About the noise in productivity metrics data** – L2, L3, and L5 gave some feedback about noises that may have affected the metric-based productivity rankings. The leaders illustrated scenarios (e.g. when refactoring code, fixing bugs or running code formatting routines) in which one or another metric may not have succeeded in reflecting the developer productivity. This was clearly exemplified by L2, when the participant said:







*“I remember that one developer refactored the source code, thereby performing many commits. It may have affected the metrics based on Code Ownership.”* – L2

#### 4.2.2 On the Potential Adoption of the Six Productivity Metrics in Practice

After asking the team leaders about the relevance of the six productivity metrics analyzed, we asked **Question 2** to each team leader: *would you adopt any of these metrics?*

From the leaders’ responses, we captured the potential adoption (or non-adoption) of each productivity metric from a team leader perspective. The interest of a leader to adopt a given productivity metric was captured from sentences such as *I would adopt this metric in practice*. The opposite applied to the non-adoption of a metric. Table 11 summarizes both potential adoption and non-adoption of each metric by team leader. In the table, we marked with \* whenever a metric presented the highest Kendall’s  $\tau$  correlation coefficient for a given team leader (cf. data of Section 4.1). We discuss below our main observations.

**Table 11** Potential adoption of productivity metrics by team leader

Metric	Team Leader								Total ✓	
	L1	L2	L3	L4	L5	L6	L7	L8		
HalsteadEff/Time			✓	✓	✓*		✓*	✓	5	
SLOC/Time	✓*		✓	✓*		✓			4	
CodeOwned/Time	✓*		✓*	✓			*		3	
Commits/Time		*		✓		*		*	1	
CommittedChars/Time				✓		*			1	
CommittedSLOC/Time				✓		*			1	

Adoption (✓); Non-adoption (X); Greatest  $\tau$  correlation coefficient (\*)

L4 proved to be the most enthusiastic team leader concerning the practical adoption of productivity metrics: he pointed out interest in adopting all six metrics. Conversely, L2 showed lack of interest in adopting productivity metrics—even the only one he previously pointed out as relevant, i.e., CodeOwned/Time (Table 10). Most of the leaders (L1, L3, L4, L5, and L7) showed interest in adopting at least one metric whose  $\tau$  correlation coefficient with their perceptions was the greatest. We highlight that only L4 would adopt commit-based metrics, which reflects an overall rejection of these metrics in practice.

**Practical benefits of using productivity metrics** – L1, L3, L5, and L6 provided us with insights on how beneficial can be the adoption of productivity metrics in practice, such as revealing not observed productive developers, productive history, other aspects of developer’s productivity. The following excerpts from L1 and L5 summarize this point of view:

*“Productivity metrics can guide us and reveal aspects of productivity not yet observed.”* – L1

*“This metric [HalsteadEff/Time] may be revealing the most productivity developer [contrary to expectations].”* – L5

**Combining productivity metrics** – L3 and L4 suggested that combining two or more productivity metrics can be useful to guide the productivity assessment. In this sense, L4 mentioned:

*“In practice, I would need to use two or more productivity metrics combined rather than only one metric.”* – L3

**Applicability of productivity metrics** – L3, L4 and L8 also provided insights about the applicability of these metrics in particular industry cases. It was interesting to see their inputs in this regard:

*“I would apply these metrics in practice because I do not want to be unfair with any developer of my team.” – L3*

*“The metric results suggest that I should pay special attention to a given developer. I could give more recognition to her work.” – L4*

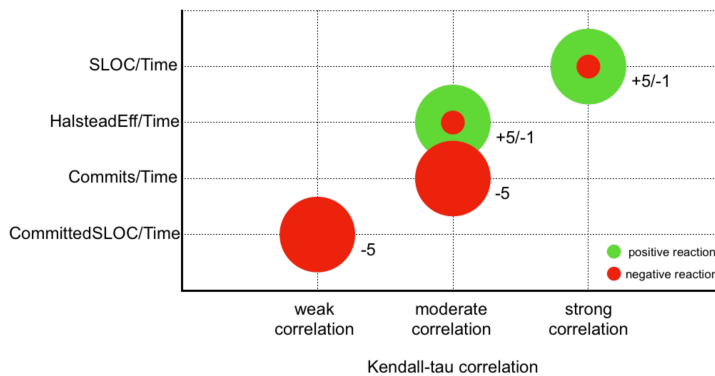
*“Productivity metrics can help me in enhancing the human resource management, so that developers are allocated to tasks in which they are more likely to be productive.” – L8*

In summary, our qualitative results showed that the team leaders’ perceptions of developer productivity are aligned with code-based metrics rather than commit-based metrics. The eight interviewed team leaders showed a general rejection of using commit-based metrics to assess developer productivity. Such rejection is often motivated by noise derived from varying commit policies and practices. However, some leaders (especially L3 and L4) pointed out that combining different metrics can be useful to enhance productivity assessment. We highlight that further investigation is required to understand how the six productivity metrics should be combined to support productivity assessment effectively.

**Summary of RQ<sub>2</sub>:** Most of the interviewed team leaders pointed out code-based metrics as relevant to assess developer productivity. Although they are not ultimately enthusiastic about adopting productivity metrics in practice, the leaders showed interest in combining multiple metrics to guide their daily work in allocating tasks to developers.

#### 4.3 Comparing quantitative results with qualitative results

Finally, we compared the results of correlations between metrics and team leaders’ productivity perceptions with team leaders’ positive or negative views of the same metrics (Figure 4).



**Fig. 4** Team leader’s perception of metrics versus metrics’ correlation strength

For this analysis, we chose to use Cohen’s interpretation of the metrics’ correlation values (see Table 9) for  $\alpha = .10$ . We decided this way because the correlation data for  $\alpha = .05$ , in addition to one of productivity metrics do not have a valid correlation (CommittedSLOC/Time), all other metrics had a valid correlation value for only one project, but not the same project. The numbers shown next to each bubble in the figure are the totals of positive (+) and negative (−) perceptions of team leaders (see Table 10).

Code-based metrics had the highest number of positive perceptions and most strongly correlated with developer productivity rankings reported by team leaders. Only a single leader indicated a negative perception for each of code-based metrics. Conversely, all leaders had only negative perceptions about commit-based metrics. Also, commit-based metrics lagged behind all comparisons made so far, except for the last comparison when Commits/Time was ahead of HalsteadEff/Time by a minimal margin.

**Summary:** Code-based metrics are those that had the most positive reactions from team leaders and also those that had the strongest correlations with team leaders’ perception of developers’ productivity. Conversely, commit-based metrics are those that had all negative reactions from team leaders and had the correlations outperformed by code-based metrics, except for the Commit/Time metric.

## 5 Lessons Learned

In this section, we present and discuss a set of lessons that we have learned by conducting this study, which may have implications in research and practice.

**Assessing developer productivity via commit-based metrics is tricky** – Our quantitative data revealed that code-based metrics outperformed commit-based metrics in reflecting the team leaders’ perceptions of developer productivity (Section 4.1). Complementarily, our qualitative data revealed a certain rejection of team leaders in adopting commit-based metrics to assess productivity. The eight interviewed leaders are unanimous in stating that commit-based metrics strongly dependent on developers’ commit habits; thus, these metrics are quite unreliable (Section 4.2). Particularly, team leaders expressed their concern about being unfair with developers who commit less but produce more complex program features than other developers. Therefore, we have learned that using commit-based metrics to assess developer productivity is tricky in practice.

**Code ownership as a key to assessing developer productivity** – We computed code-base metrics based on code files owned by a software developer (Section 2.2). Differently of past research, such as [7] and [31], which manually computed code ownership, we used a recent approach based on software repository mining [4]. Our results were encouraging for code-based metrics: all code-based metrics are strongly correlated with team leaders’ perceptions of developer productivity. Despite criticisms of previous work (e.g., [45] and [49]), even SLOC/Time was successful in reflecting the leaders’ perceptions. Conversely, the isolated use of commit frequency by the commit-based metrics showed ultimately unsuccessful (Tables 4 and 10). As the CodeOwned/Time metric strongly correlated with the SLOC/Time metric

(Table 5), we hypothesize that code ownership plays an important role, suggesting that team leaders intuitively see developers working frequently (owner of code) on *the most important* (possible biggest or most complex code) part of the project source code as the most productive ones. Future work could further investigate this hypothesis in practice.

**To what extent code-based and commit-based can complement the team leaders' perceptions of their developer' productivity?** Although developer productivity is a subjective concept [1], it is essential to the success of software development organizations [2] [28]. Thus, any additional support to team leaders in assessing developer productivity can benefit the management of development teams and enhance team productivity as a whole. One of the goals of our study was to understand whether productivity metrics could complement the perceptions of team leaders in assessing developers productivity. Our results showed that, even with the leaders' explicit rejection to adopt specific metrics (Section 4.2), productivity based on code-based metrics are usually aligned with the leaders' subjective perception of productivity (Section 4.1). In summary, our results suggest that productivity metrics, especially code-based metrics, can complement the subjective perception of team leaders. The leaders themselves highlighted the benefits: revealing aspects of developer productivity not previously known, boost the fairness of productivity assessment, and acknowledge those developers that are productive but underestimated (Section 4.2). Ultimately, this work provides empirical evidence that productivity metrics can be triangulated with the team leaders' perceptions towards a more accurate productivity assessment.

## 6 Related Work

Developer productivity assessment has been extensively investigated by previous work in order to support organizations, project managers, and team leaders in their needs [35] [36] [40] [49]. Some studies (e.g., [35] [36]) aimed to understand how developer productivity has been computed and managed from a practical perspective. Section 6.1 discusses these studies. Additionally, other studies (e.g., [40] [49]) assess what project managers and team leaders think about developer productivity. Section 6.2 discusses these studies. We compare our study results with those obtained by previous work as much as possible.

### 6.1 General Studies on Perceptions of Developer Productivity

A first study [35] investigated the productivity perceptions of agile teams. That study relies on two industry case studies with 13 team members, including developers and team leaders. Through semi-structured interviews and informal discussions, the study revealed the lack of a common perception of productivity. Developers and team leaders have mixed perceptions that range from time spent to produce code to the quality and quantity of code created. An example of factors usually perceived as harmful to productivity is the poor team organization, which may hinder knowledge sharing among members. Unfortunately, that study [35] did not investigate details about how developers and team leaders measure productivity using time spent, quality, and quantity of produced code. In this work, we

address this literature gap by capturing the perception of team leaders in agile teams about developer productivity metrics. We rely on six metrics that compute either on the characteristics of produced code or the commit activity. Our results point out that code-based metrics better represent the team leaders perceptions of developer productivity.

A second study [36] has complimented some key observations provided by the first study [35] but in the context of developer productivity. The authors have performed a mixed-method empirical research based on surveys with 379 experienced developers and interviews with 11 developers. Similarly to the first study [35], the authors have confirmed different perceptions of developers about their productivity with no consensus among developers. These perceptions ultimately rely on the number of achieved goals and completed development tasks, which could be translated to the amount of produced code. As a complement, the authors found that developers consider their working days productive when a lower number of interruptions and working context switches occur. The rate of completed tasks by day and the amount of code produced and bug-fixed were frequently used by developers to compute their productivity. Differently from both studies [35] [36], our current work targets the perceptions of developer productivity from the viewpoint of team leaders. We acknowledge the need for understanding whether developer productivity metrics could help leaders in managing their development teams, once this is the principal leader responsibility in a team.

## 6.2 Developer Productivity: A Project Manager and Team Leader Perspective

A previous study [49] has investigated the perception of developers and project managers about two productivity metrics: Function Points (FP) [22] and Source Lines of Code (SLOC) [34]. FP captures the number of program features realized by the produced source code, and SLOC captures the amount of produced code itself, usually regardless of the features implemented by these lines of code. The authors have surveyed 28 developers and 42 project managers. The results are quite interesting: although only 34% of developers and managers said to be more familiar with SLOC rather than FP, they are more likely to adopt SLOC due to its computation simplicity. As a complement, our current work empirically stated that SLOC produced by time and other code-based metrics fit the team leaders perceptions of developer productivity.

Similarly to the work mentioned above [49], we have recently investigated the project managers perceptions of developer productivity [40]. Our primary goal was achieving a general understanding of what productivity means for project managers and how they have been measuring the productivity of their team developers in practice. For this purpose, we performed semi-structured interviews with 12 project managers from two software development organizations. Our results showed that project managers often perceive the number of tasks delivered on-time and the frequency of produced artifacts as hints of developer productivity. Surprisingly, the interviewed managers said they strongly depend on the team leaders perceptions to assess developer productivity, thereby neglecting the use of productivity metrics. Due to the limited knowledge of how team leaders assess developer productivity in practical settings, we performed the current work.



## 7 Threats to Validity

We carefully designed and performed our empirical study aimed to understand the team leaders perceptions of developer productivity metrics. We employed our best effort in developing appropriate study settings, artifacts, and analysis protocols. Nevertheless, some threats to the study validity should be considered before reproducing our study and adopting our recommendations in practice. Below, we discuss threats to validity based on a previous work [57].

**Construct Validity.** (1) We performed a pilot study that counted on the participation of five team leaders. We recruited them from organizations not exploited in this work. The voluntary participation of all five volunteers helped us to adjust our survey and interview protocols. It has enabled us to reduce threats related to our capability to obtain all data required to address our research questions. (2) We cherry-picked the software projects in this work based on the availability of data for analysis. We asked managers of each organization to indicate the projects available for study. One out of the five projects provided to us by Organization 2 was discarded due to insufficient data of code and commit history data. All other projects were considered for analysis. Thus, we expected to minimize biases in the project selection. (3) We asked managers of each organization to indicate the team leaders with the highest leading experience for participating in our study. Although we expected to minimize bias in selecting team leaders, this procedure may conversely have created a bias. . Thus, we carefully selected those team leaders with the highest leading experience per organization. Therefore, we expected to minimize biases in the selection of team leaders.

**Internal Validity.** (1) We followed strict guidelines for computing the productivity metrics and all data required to rank developers by productivity. For computing the source code metrics like as Source Lines of Codes (SLOC) [34], we considered only the primary programming language used to implement each software project. Additionally, we prioritize projects implemented in Java due to the worldwide language popularity. Finally, we computed code ownership based on a previous work [4], which has been adopted by recent studies like [25] and [50]. Besides, we selected an algorithm to identify code ownership. Thus, we expected to reduce biases in the metrics computation. (2) We discarded all commit operations performed on non-source code files, frequently done by non-developers, thereby reducing noise in the analyzed data. (3) We have computed the developer productivity based on the average productivity of each developer by working week. We relied on a previous work [48] and the fact that, for both exploited organizations, the time interval among commits is usually lower than seven days (in 95% of cases).

**External Validity.** (1) We carefully selected the statistical tests that could help us in understanding the significance of our data. We relied on previous studies [16] [33] for computing correlations of metrics with developer perceptions. At least two paper authors checked all computed data. In case of divergences, we discussed each issue and performed the proper fixes. (2) We followed thematic analysis procedures [12] to analyze the interview data. The first author has joined another paper author for conducting the qualitative analysis of interview data. Thus, we expected to minimize threats like missing and incorrect data. All interview transcriptions were carried out in Portuguese, which is the native language for all paper authors and interviewees. (3) Double-validating the developer rank-

ings computed based on metrics enabled us to identify that four developers (two of project P1 and two of P8) were missing in the rankings. Regarding P1, we asked leader L1 to recompute the rankings, and it was done promptly because he/she just missed those developers. Regarding P8, we realized that leader L8 omitted two developers on purpose because they were no longer part of that project. Once both developers have been part of the project from the beginning, we asked L8 to recompute the rankings, which was done promptly as well.

**Conclusion Validity.** (1) We did our best to invite as many development organizations as possible for participation in our study. For each organization, we asked the project managers to indicate as many projects and team leaders to analyze and survey. Although, we are aware that the selected organizations, projects, and team leaders are not representative of all industry settings. Nevertheless, we expected to achieve a reasonable number of participants for this study. (2) For understanding the team leaders perceptions of developer productivity, we have selected a considerable set of metrics. We carefully picked three code-based and three commit-based metrics for the study. These metrics were identified and categorized in a previous work [41]. Thus, we expected to have a study that is diverse in terms of productivity metrics that may capture the team leaders perceptions. (3) Although all selected organizations are focused on agile development, we expect that our study findings apply to some extent to other development contexts, e.g., the traditional development. We highlight that the two organizations exploited in this work represent typical regional organizations elsewhere.

## 8 Final Remarks

To assess the productivity of software developers in real software projects is essential to the success of development organizations [2] [52]. Aimed to support such assessment, various studies have proposed means to guide leaders in managing human resources in their development teams [35] [36] [40] [49]. Especially, many metrics were introduced by past research in order to infer different aspects of developer productivity [24] [31] [48]. These metrics usually rely on either the characteristics of produced source code (code-based metrics) or the commit frequency performed by developers (commit-based metrics). Unfortunately, the current empirical knowledge on to what extent these metrics reflect the team leaders' perceptions of developer productivity is quite scarce if not nonexistent. In this context, we carefully designed and performed a mixed-method empirical study on leaders' perceptions of six productivity metrics proposed in the literature (Section 2). This work has combined a correlational study (Section 4.1) with interviews (Section 4.2) performed with eight team leaders that work for nine software projects (68 developers in total) with a correlational study (Section 4.1). The nine projects were selected from two development organizations with more than 18 years of activity.

In the correlational study, we computed the correlation of developer rankings subjectively calculated by the team leaders with rankings computed via productivity metrics. Although correlation does not necessarily imply causality, our correlational study data suggest a higher correlation of the leaders' perceptions with code-based metrics when compared to commit-based metrics, regardless the development organization. In the interview-based study, we observed that team leaders are more likely to find relevant code-based metrics rather than the commit-based

metrics. We also observed that team leaders are interested in combining two or more metrics information to support their daily work of managing teams, gathering benefits such as: revealing aspects of developer productivity not previously known, boost the fairness of productivity assessment, those developers that are productive but underestimated. Based on these results, productivity metrics, especially code-based metrics, can complement the subjective perception of team leaders. Therefore, *we recommend that team leaders adopt in practical settings productivity metrics, especially code-based metrics, to complement the developers' productivity assessment.*

**Suggestions for future research:** Our empirical study shed light on some opportunities for future research. First, one could perform empirical studies aimed to understand the effect of commit data on the performance of code-based metrics for assessing developer productivity. *To what extent the commit frequency enhances the developer productivity metrics?* is an example of a question that could be addressed based on the discussion presented in Section 5. Second, researchers could perform participatory action research to investigate team leaders that employ the productivity metrics (Section 2) in practice. *How do team leaders employ productivity metrics to manage development teams?* and *What is the effect of using metrics on the productivity of real development teams?* are suggested research questions. Third, one could investigate the human factors underlying the team leaders' perceptions of developer productivity.

## References

1. Amrit, C., Daneva, M., Damian, D.: Human factors in software development: On its underlying theories and the value of learning from related disciplines. a guest editorial introduction to the special issue. *Information and Software Technology (IST)* **56**(12), 1537–1542 (2014)
2. de Aquino Junior, G., Meira, S.: Towards effective productivity measurement in software projects. In: *Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA)*, pp. 241–249 (2009)
3. Basili, V., Rombach, H.D.: The tame project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering (TSE)* **14**(6), 758–773 (1988)
4. Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code! examining the effects of ownership on software quality. In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE)*, pp. 4–14 (2011)
5. Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R.: Cost models for future software life cycle processes: COCOMO 2.0. *Ann. Softw. Eng.* **1**(1), 57–94 (1995). DOI 10.1007/BF02249046
6. Boehm, B.W.: *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, New Jersey (1981)
7. Chand, D., Gowda, R.: An exploration of the impact of individual and group factors on programmer productivity. In: *Proceedings of the Conference on Computer Science (CSC)*, pp. 338–345 (1993)
8. Chávez, A., Ferreira, I., Fernandes, E., Cedrim, D., Garcia, A.: How does refactoring affect internal quality attributes? a multi-project study. In: *Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES)*, pp. 74–83 (2017)
9. Chrissis, M.B., Konrad, M., Shrum, S.: *CMMI for development: guidelines for process integration and product improvement*. Pearson Education (2011)
10. Cohen, J.: *Statistical power analysis for the behavioural sciences*, 2 edn. Routledge (1988)
11. Croux, C., Dehon, C.: Influence functions of the spearman and kendall correlation measures. *Statistical methods & applications* **19**(4), 497–515 (2010)

12. Cruzes, D., Dyba, T.: Recommended steps for thematic synthesis in software engineering. In: Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275–284 (2011)
13. De Silva, L., Balasubramaniam, D.: Controlling software architecture erosion: A survey. *Journal of Systems and Software (JSS)* **85**(1), 132–151 (2012)
14. Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.: Boa: Ultra-large-scale software repository and source-code mining. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **25**(1), 7 (2015)
15. Fenton, N.E., Neil, M.: Software metrics: Roadmap. *Proc. Conf. Futur. Softw. Eng. ICSE 2000* pp. 357–370 (2000). DOI 10.1145/336512.336588
16. Fernandes, E., Ferreira, L.P., Figueiredo, E., Valente, M.T.: How clear is your code? an empirical study with programming challenges. In: Proceedings of the Ibero-American Conference on Software Engineering: Experimental Software Engineering Track (CIBSE-ESELAW), pp. 1–14 (2017)
17. Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E.: A review-based comparative study of bad smell detection tools. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 18:1–18:12 (2016)
18. Ferreira, A.I.F., Santos, G., Cerqueira, R., Montoni, M., Barreto, A., Barreto, A.O.S., Rocha, A.R.: Applying iso 9001: 2000, mps. br and cmmi to achieve software process maturity: Bl informatica’s pathway. In: 29th International Conference on Software Engineering (ICSE’07), pp. 642–651. IEEE (2007)
19. Ferreira, M., Valente, M.T., Ferreira, K.: A comparison of three algorithms for computing truck factors. In: Proceedings of the 25th International Conference on Program Comprehension (ICPC), pp. 207–217 (2017)
20. Field, A.: *Discovering Statistics using IBM SPSS Statistics*, 3 edn. Sage Publications Ltd. (2009)
21. Fowler, M.: Cannot measure productivity (2003). URL <https://martinfowler.com/bliki/CannotMeasureProductivity.html>. [Online; posted 29-August-2003]
22. Furey, S.: Why we should use function points. *IEEE Software* **14**(2), 28 (1997)
23. Gilpin, A.R.: Table for conversion of kendall’s tau to spearman’s rho within the context of measures of magnitude of effect for meta-analysis. *Educational and Psychological Measurement* **53**(1), 87–92 (1993). DOI 10.1177/0013164493053001007
24. Gousios, G., Kalliamvakou, E., Spinellis, D.: Measuring developer contribution from software repository data. In: Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR), pp. 129–132 (2008)
25. Greiler, M., Herzig, K., Czerwonka, J.: Code ownership and software quality: A replication study. In: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR), pp. 2–12 (2015)
26. Halstead, M.: *Elements of software science*, 1 edn. Elsevier Science (1977)
27. Hernández-López, A., Colomo-Palacios, R., García-Crespo, A.: Software Engineering Job Productivity – A Systematic Review. *Int. J. Softw. Eng. Knowl. Eng.* **23**(03), 387–406 (2013). DOI 10.1142/S0218194013500125
28. Huselid, M.: The impact of human resource management practices on turnover, productivity, and corporate financial performance. *Academy of Management Journal (AMJ)* **38**(3), 635–672 (1995)
29. International Standard Organization: ISO 9001:2015 Quality management systems – Requirements (2015). URL <https://www.iso.org/standard/62085.html>
30. Kendall, M.G.: *Rank correlation methods*, 2 edn. Hafner Publishing (1955)
31. Lawrence, M.: Programming methodology, organizational environment, and programming productivity. *Journal of Systems and Software (JSS)* **2**(3), 257–269 (1981)
32. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Physics* **10**(8), 707–710 (1966)
33. Lokan, C.: An empirical study of the correlations between function point elements. In: Proceedings of the 6th International Software Metrics Symposium (METRICS), pp. 200–206 (1999)
34. Lorenz, M., Kidd, J.: *Object-oriented software metrics: A practical guide*, 1 edn. Prentice Hall (1994)
35. Melo, C., Cruzes, D., Kon, F., Conradi, R.: Agile team perceptions of productivity factors. In: Proceedings of the Agile Conference (Agile2011), pp. 57–66 (2011)
36. Meyer, A., Fritz, T., Murphy, G., Zimmermann, T.: Software developers’ perceptions of productivity. In: Proceedings of the 22nd International Symposium on Foundations of Software Engineering (FSE), pp. 19–29 (2014)

37. Mockus, A., Fielding, R., Herbsleb, J.: Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **11**(3), 309–346 (2002)
38. Mordal, K., Anquetil, N., Laval, J., Serebrenik, A., Vasilescu, B., Ducasse, S.: Software quality metrics aggregation in industry. *Software: Evolution and Process (S:E&P)* **25**(10), 1117–1135 (2013)
39. Munson, J., Elbaum, S.: Code churn: A measure for estimating the impact of code change. In: *Proceedings of the 6th International Conference on Software Maintenance (ICSM)*, pp. 24–31 (1998)
40. Oliveira, E., Conte, T., Cristo, M., Mendes, E.: Software project managers’ perceptions of productivity factors: Findings from a qualitative study. In: *Proceedings of the 10th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 15:1–15:6 (2016)
41. Oliveira, E., Viana, D., Cristo, M., Conte, T.: How have software engineering researchers been measuring software productivity? A systematic mapping study. In: *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS)*, pp. 76–87 (2017)
42. Oliveira, R., de Mello, R., Fernandes, E., Garcia, A., Lucena, C.: Collaborative or individual identification of code smells? On the effectiveness of novice and professional developers. *Information and Software Technology (IST)* **120**, 106242 (2020)
43. Ordonez, M., Haddad, H.: The state of metrics in software industry. In: *Proceedings of the 5th International Conference on Information Technology: New Generations (ITNG)*, pp. 453–458 (2008)
44. Petersen, K.: Measuring and predicting software productivity: A systematic map and review. *Information and Software Technology (IST)* **53**(4), 317–343 (2011)
45. Rahman, F., Devanbu, P.: How, and why, process metrics are better. In: *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pp. 432–441 (2013)
46. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering (EMSE)* **14**(2), 131 (2009)
47. Runeson, P., Host, M., Rainer, A., Regnell, B.: *Case study research in software engineering: Guidelines and examples*, 1 edn. John Wiley & Sons (2012)
48. Scholtes, I., Mavrodiev, P., Schweitzer, F.: From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Software Engineering (EMSE)* **21**(2), 642–683 (2016)
49. Sheetz, S., Henderson, D., Wallace, L.: Understanding developer and manager perceptions of function points and source lines of code. *Journal of Systems and Software (JSS)* **82**(9), 1540–1549 (2009)
50. Thongtanunam, P., McIntosh, S., Hassan, A., Iida, H.: Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pp. 1039–1050 (2016)
51. Trendowicz, A., Münch, J.: Factors influencing software development productivity: State-of-the-art and industrial experiences. *Advances in Computers* **77**, 185–241 (2009)
52. Verner, J., Babar, M., Cerpa, N., Hall, T., Beecham, S.: Factors that motivate software engineering teams: A four country empirical study. *Journal of Systems and Software (JSS)* **92**(1), 115–127 (2014)
53. Wagner, S., Ruhe, M.: A systematic review of productivity factors in software development. In: *Proceedings of the 2nd International Workshop on Software Productivity Analysis and Cost Estimation (SPACE)*, pp. 1–6 (2008)
54. Weber, K.C., Araújo, E.E., da Rocha, A.R.C., Machado, C.A., Scalet, D., Salviano, C.F.: Brazilian software process reference model and assessment method. In: *International Symposium on Computer and Information Sciences*, pp. 402–411. Springer (2005)
55. Wen, M., Wu, R., Cheung, S.C.: Locus: Locating bugs from software changes. In: *Proceedings of the 31st International Conference on Automated Software Engineering (ASE)*, pp. 262–273 (2016)
56. Wloka, J., Ryder, B., Tip, F., Ren, X.: Safe-commit analysis to facilitate team software development. In: *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pp. 507–517 (2009)
57. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: *Experimentation in software engineering*, 1 edn. Springer Science & Business Media (2012)