MEASURING SOFTWARE DEVELOPER PRODUCTIVITY: REVIEW OF APPROACHES AND THEIR LIMITATIONS

Vladimir Handzhiev
MSc in Software Engineering and Management
Varna Free University "Chernorizets Hrabar", Bulgaria

Abstract: *This article examines modern approaches for measuring software developer productivity, focusing on the tools used, the available data, and the applied metrics. The main sources of information in the development process are analyzed, including task management systems and version control platforms, as well as different groups of metrics – classical, DevOps, and experimental.*

Special attention is given to the limitations of these approaches. It is shown that a significant part of the real work of developers remains outside the available data and is not reflected in the used metrics. Key problems are discussed, such as "invisible work", lack of context in the data, dependence on team dynamics, and the influence of metrics on behavior.

Based on the analysis, it is concluded that productivity cannot be adequately measured using single quantitative indicators. Instead, a broader and more integrated approach is needed, which takes into account both the technical and the social aspects of work in software teams.

Keywords: *developer productivity, software engineering, metrics, DevOps, DORA, invisible work, data analysis, software teams*

I INTRODUCTION

Software developer productivity is an important topic both in practice and in research. Organizations rely on different metrics and tools to evaluate the effectiveness of their teams, to optimize processes, and to make management decisions. However, the question remains to what extent these measurements reflect the real work of developers.

Modern tools such as issue tracking systems and version control systems generate detailed data about activity within software projects. Based on this data, various metrics are created, ranging from classical indicators such as number of lines of code and completed tasks to more modern DevOps metrics (DORA metrics) and experimental approaches based on controlled studies.

Despite this progress, there are serious limitations in the way productivity is measured. Part of the work of developers remains outside the available data, and the used metrics often reflect only visible activity. This raises questions about the validity of the evaluations and creates a risk of incorrect conclusions.

The aim of this article is to provide a systematic review of the existing approaches for measuring developer productivity, focusing on the tools used, the available data, and the applied metrics. Special attention is given to the limitations of these approaches and to the role of so-called "invisible work", which is not reflected in standard systems but has an important role in the development process.

II CONCEPTUALIZATION OF DEVELOPER PRODUCTIVITY

Software developer productivity (developer productivity) is a widely used but difficult to define term. In practice, it is often reduced to measurable results such as number of completed tasks, written code, or time for execution. However, in the scientific literature there is no single and universally accepted definition (Hernández-López et al., 2013).

The classical approach considers productivity as a ratio between output and time spent. In the context of software engineering, this usually means measuring through indicators such as number of lines of code, number of completed tasks, or functionalities per unit of time. This model is widely used, but it is strongly simplified and does not reflect the real complexity of development (Hernández-López et al., 2013).

The main problem of this approach is that it treats development as a linear and measurable activity, similar to a production process. In reality, software development can be seen as a form of intellectual work (knowledge work), where a significant part of the effort is related to thinking, analysis, decision making, and problem solving (Forsgren et al., 2021). These activities are difficult to measure using quantitative indicators.

Modern research offers a broader view of productivity. One of the most influential models is the SPACE framework, which considers productivity as a multi-dimensional concept, not as a single indicator (Forsgren et al., 2021). According to this model, productivity includes five main dimensions:

- Satisfaction and well-being;
- Performance;
- Activity;
- Communication and collaboration;
- Efficiency and flow.

This approach clearly shows that traditional metrics based on activity (for example commits or completed tasks) reflect only a small part of real productivity (Forsgren et al., 2021). For example, a developer may have low "visible" activity but still contribute significantly through architectural decisions, helping colleagues, or coordination within the team.

Another important aspect is that productivity is not entirely an individual characteristic. It depends on the context in which the work is performed, including team structure, processes, tools, and organizational culture (Cataldo et al., 2006). This means that the same developer may show different levels of productivity in different environments.

In summary, developer productivity cannot be fully described by a single quantitative indicator. It represents a complex combination of technical results, cognitive efforts, and social interactions. This creates serious limitations for attempts to measure it accurately using data from systems such as Jira or Git-based platforms (Forsgren et al., 2021).

III DATA SOURCES FOR MEASURING ACTIVITY

Modern approaches for measuring developer productivity are mainly based on data generated by the tools used in the development process. Most often these are issue tracking systems such as Jira and version control systems such as Git, GitHub, and GitLab.

These systems produce a large amount of structured information about developer activity. In task management systems there is data about creation, changes, and completion of tasks, as well as the participants involved in them. On the other hand, Git-based platforms provide information about commits, merge requests (or pull requests), changes in the code, as well as history of interactions between developers through comments and code review.

This data is the basis for a large part of modern research in software engineering, especially in areas such as Mining Software Repositories (MSR). Research in this area shows that code and the related activity are not random, but follow certain patterns and repetition. For example, analysis of program code shows that it is highly structured and predictable, which allows the use of statistical and machine learning methods for its study (Hindle et al., 2012).

This makes these data sources very attractive for measurement and analysis. They are easily accessible, automatically generated, and allow large-scale analysis without the need for additional data collection. As a result, many organizations and studies use these data as a basis for evaluating productivity.

Despite these advantages, there is an important limitation. Data from such systems mainly reflects visible activity, that is, actions that leave a trace in the tools (Hernández-López

et al., 2013; Forsgren et al., 2021). This includes writing code, closing tasks, or participating in discussions, but does not cover all aspects of development.

A significant part of developer work is not recorded in these systems (Meluso et al., 2024). This includes activities such as problem analysis, making architectural decisions, informal communication, helping colleagues, and gaining knowledge. These activities are often critical for the success of the project, but remain outside the available data.

Therefore, although systems such as Jira and Git provide valuable and rich information, they give only a partial picture of real activity. This limitation has a direct impact on all further attempts to measure productivity, because the measurement is based on incomplete data (Hernández-López et al., 2013).

IV METRICS FOR DEVELOPER PRODUCTIVITY

The measurement of developer productivity is based on different metrics, which are used both in scientific literature and in practice. These metrics can be generally divided into three groups: classical, modern (DevOps), and experimental (Hernández-López et al., 2013).

4.1 Classical metrics

Classical metrics are based on a simple model: they compare output to time spent. The most commonly used indicators include:

- number of lines of code;
- number of completed tasks;
- time to complete a task.

This approach is widely used, but it has serious limitations. It assumes that more code or more completed tasks means higher productivity. In real situations this is not always true. For example, less code can be more efficient, and complex tasks often require more time without meaning lower productivity (Hernández-López et al., 2013).

In addition, these metrics do not take into account the quality of the result, the complexity of the tasks, or the effort involved. As shown in the systematic review by Hernández-López et al. (2013), such measurements cover only part of the real activity and do not include all relevant factors.

4.2 Modern metrics (DevOps and DORA)

With the development of DevOps practices, new metrics appear that try to give a more realistic evaluation of productivity at team and organizational level (Forsgren et al., 2018). The most well-known are the DORA metrics (DevOps Research and Assessment), which include:

- deployment frequency;

- lead time;
- mean time to recovery;
- change failure rate.

These metrics provide a better view of process efficiency and the ability of the team to deliver value. They are widely used in industry and are considered a good indicator of development maturity.

However, they also have limitations. DORA metrics mainly measure the delivery process, but do not provide information about individual contribution, cognitive effort, or internal team dynamics. They show "what is happening", but do not explain "why it is happening" (Forsgren et al., 2021).

4.3 Experimental metrics

With the introduction of new technologies such as artificial intelligence, experimental approaches for measuring productivity also appear. An example is the study of GitHub Copilot, where productivity is measured through time to complete a specific task and the level of successful completion.

The results show a significant improvement in execution speed when using an AI tool, as participants with Copilot complete the task on average 55.8% faster (Peng et al., 2023).

However, this approach is also limited. The measurement is done in a controlled environment, with a clearly defined task and without real team work. In addition, the focus is again on speed, without considering code quality, task complexity, or the long-term effect of using such tools (Peng et al., 2023).

It should be noted that the three groups of metrics reflect different aspects of productivity, but do not provide a single and consistent framework for its measurement. Classical metrics focus on individual output, DevOps metrics focus on process efficiency, and experimental approaches focus on the execution of specific tasks in a controlled environment. This difference shows that there is a lack of consensus about what "productivity" means in the context of software engineering.

Summary

Despite the variety of metrics, there is a common problem: all of them measure only part of productivity. Classical metrics are too simplified, modern DevOps metrics focus on the process, and experimental approaches are often limited to specific scenarios.

None of these groups of metrics is able to cover the full range of activities that make up the real work of developers. This raises the question whether productivity can be adequately measured only through the available data and tools (Forsgren et al., 2021).

V LIMITATIONS OF CURRENT APPROACHES TO MEASUREMENT

Despite the presence of various metrics and rich data sources, there are significant limitations in the way developer productivity is measured. These limitations come not so much from lack of data, but from its incompleteness and from the nature of software development itself.

5.1 Invisible work

One of the most serious problems is the presence of a significant amount of work that is not reflected in the used tools. Research shows that a large part of activities in software projects remains outside formal systems for task and code management. For example, analysis of open-source projects shows that approximately half of the work performed is not directly visible through standard indicators such as commits or tasks (Meluso et al., 2024).

This "invisible work" includes activities such as:

- analysis and understanding of problems;
- architectural decisions;
- informal communication;
- help and mentoring;
- coordination between participants.

Such activities are critical for the success of the project, but do not leave a direct trace in systems such as Jira or Git. As a result, they are not included in standard measurements of productivity (Forsgren et al., 2021).

In addition, even at the level of program code there are activities that are not directly related to functionality but require significant effort. An example of this is formal specifications and annotations used to prove the correctness of programs. They do not add new functionality, but are important for the quality and reliability of the system (Lathouwers & Huisman, 2022).

5.2 Coordination and collaboration

Software development is rarely an individual activity. In most cases it is done in a team environment, where interactions between participants play a key role. Research shows that the alignment between technical dependencies in the system and communication between developers has a direct impact on productivity (Cataldo et al., 2006).

This means that part of productivity is determined by how well the team coordinates its work, not only by the individual contribution of each participant. However, standard metrics rarely take this aspect into account. They are mainly focused on individual activity and do not capture the complexity of team dynamics (Forsgren et al., 2021).

5.3 Incomplete representation of data

The data used to measure productivity represents only part of real activity (Hernández-López et al., 2013). Systems such as Jira and Git record specific actions, but do not contain information about the context in which these actions are performed.

For example, a commit may look small, but it may be the result of a long process of analysis and experiments. On the other hand, a large amount of code may be generated quickly, without significant intellectual complexity. Without additional context, these differences remain invisible.

A similar problem is observed in other areas where evaluation depends on the quality of available data. If key elements are missing, the final evaluation may be misleading, regardless of the metrics used.

5.4 Influence of metrics on behavior

Another important problem is that the metrics themselves can influence the behavior of developers. When certain indicators are used for evaluation, people naturally start to optimize their behavior according to them.

For example:

- if the number of lines of code is measured, it may encourage writing more, but not necessarily better code;
- if the number of completed tasks is measured, easier tasks may be preferred instead of more complex and important ones;
- if speed is measured, quality may be sacrificed.

This effect is well known in the literature and shows that metrics are not neutral – they can distort the real picture and lead to unwanted behavior.

Summary

The limitations of current approaches can be summarized in several main points: a significant part of the work remains invisible; productivity is strongly dependent on team dynamics, which is difficult to measure; the available data is incomplete and lacks context; metrics can influence behavior and distort results.

These factors show that measuring productivity through standard tools and metrics gives only a partial and often misleading picture. This creates the need for a broader and more integrated approach to analyzing work in software teams (Forsgren et al., 2021).

VI DISCUSSION

The analysis of the existing sources, metrics, and limitations shows that the problem of measuring developer productivity is not only technical, but mainly conceptual (Forsgren et al.,

2021). Despite the significant progress in tools and available data, there is a fundamental mismatch between what can be measured and what actually represents the work of developers.

On one hand, modern systems such as Jira and Git-based platforms provide detailed information about visible activity. This data allows automated analysis and the creation of various metrics. On the other hand, as shown earlier, this data covers only part of the process and misses key aspects such as cognitive effort, informal communication, and coordination (Hernández-López et al., 2013).

This leads to an important conclusion: an increase in the amount of data does not automatically lead to more accurate measurement. Even when there is rich and detailed data, if it does not reflect all relevant aspects of the work, the final evaluation remains incomplete.

Also, different groups of metrics – classical, DevOps, and experimental – try to solve the problem from different angles, but none of them provides a complete solution. Classical metrics are too simplified, DevOps metrics focus on the process, and experimental approaches are often limited to specific scenarios. As a result, there is fragmentation in the way productivity is understood and measured (Forsgren et al., 2021).

Another important aspect is the role of context. Productivity is not a universal value that can be measured in the same way in all situations. It depends on many factors, including task complexity, team structure, used technologies, and organizational environment. This makes direct comparison between developers or teams particularly problematic.

In this context, the concept of “invisible work” can be seen as a missing element in existing models. It explains why a significant part of the effort is not reflected in the available data and why measurement based only on visible activity is not enough.

Instead of searching for a universal metric, a more realistic approach is to accept that productivity is multi-dimensional and context-dependent. This suggests the use of combined approaches that include both quantitative data and qualitative evaluation, as well as better understanding of processes and interactions within the team.

VII: CONCLUSION

In this article, a review of the main approaches for measuring software developer productivity was made, including the tools used, the available data sources, and the applied metrics. The analysis showed that modern systems such as Jira and Git-based platforms provide rich but limited information, which mainly reflects the visible activity in the development process.

It was found that there is a significant gap between measurable indicators and the real work of developers. A large part of the effort remains outside the available data, including cognitive activities, coordination, and informal communication.

The article shows that despite the increasing amount of data and the development of tools, the measurement of productivity remains incomplete and often misleading. The main conclusion is that productivity cannot be fully reduced to a single quantitative indicator, but requires a broader and multi-dimensional approach that takes into account both the technical and the social aspects of work in software teams.

BIBLIOGRAPHY

Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M., 2006. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 353–362.

Forsgren, N., Humble, J., Kim, G., 2018. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution Press.

Forsgren, N., Storey, M.A., Maddila, C., Zimmermann, T., Houck, B., Butler, J., 2021. The SPACE of developer productivity: There's more to it than you think. Queue, 19(1), pp. 20–48.

Hernández-López, A., Colomo-Palacios, R., García-Crespo, Á., 2013. Software engineering job productivity: A systematic review. International Journal of Software Engineering and Knowledge Engineering, 23(8), pp. 1135–1158.

Hindle, A., Barr, E.T., Su, Z., Gabel, M., Devanbu, P., 2012. On the naturalness of software. In: Proceedings of the 34th International Conference on Software Engineering (ICSE), pp. 837–847.

Lathouwers, S., Huisman, M., 2022. Formal specifications investigated: A classification and analysis of annotations for deductive verifiers. Journal of Systems and Software, 186.

Meluso, J., Casari, A., McLaughlin, K., Trujillo, M.Z., 2024. Invisible labor in open source software ecosystems. arXiv preprint arXiv:2401.06889.

Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M., 2023. The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv preprint arXiv:2302.06590.