International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 8 Issue 5, Sep-Oct 2024 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

A Multi-Faceted Approach to Measuring Engineering Productivity

Manish Sanwal, Ishan Deva

Engineering Department, News Corporation, New York, USA

ABSTRACT

Measuring engineering productivity in software development is a complex and nuanced challenge. This paper introduces a comprehensive framework designed to estimate a productivity score based on a diverse set of metrics, including Agile practices, code contributions, code quality, review activities, QA efforts, and deployment metrics. The proposed calculator serves as a self-assessment tool for teams to enhance their processes, rather than as a means to compare individual or team performances. We provide a detailed explanation of each metric, its assigned weight, and discuss potential risks and considerations. Real-world examples are included to illustrate the practical application of the calculator.

KEYWORDS: *Engineering Productivity, Software Metrics, Agile, Code Quality, Software Engineering*

How to cite this paper: Manish Sanwal | Ishan Deva "A Multi-Faceted Approach to Measuring Engineering Productivity" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-8 | Issue-5, October 2024, pp.516-521, URL: www.ijtsrd.com/papers/ijtsrd69393.pdf

IJTSRD69393

I. INTRODUCTION

Measuring software developer productivity has long been a topic of interest and debate within the software engineering community. Traditional metrics, such as lines of code written or the number of commits, often fail to capture the multifaceted nature of software development activities [1]. Recent studies have sought to understand and quantify developer productivity in more meaningful ways.

For instance, a study by GitHub quantified the impact of GitHub Copilot on developer productivity and satisfaction, demonstrating that AI-assisted coding can significantly enhance both [2]. The research found that developers using Copilot completed tasks faster and reported higher levels of satisfaction, highlighting the importance of considering tools and environmental factors when measuring productivity.

Understanding the challenges of measuring developer productivity is essential. As noted in The Pragmatic Engineer newsletter, measuring productivity is not straightforward due to the intangible and creative aspects of software development [3]. Factors such as problem complex-ity, collaboration, and innovation play significant roles that are difficult to quantify with simple metrics.

Given these complexities, there is a need for a more holistic approach to measuring devel-oper productivity that encompasses various aspects of the development process. This paper introduces a productivity calculator that incorporates multiple metrics across different domains, aiming to provide a balanced assessment of team performance. The calculator is designed to help teams identify areas for improvement and foster a culture of continuous enhancement, without using the metrics to compare individuals or teams unfairly.

II. THE PROBLEM

Measuring developer productivity is a critical aspect of software development, yet it remains a complex and often frustrating challenge for engineering leaders [5, 6]. Traditional metrics, such as lines of code written or time spent coding, fail to capture the nuances of a developer's work and can incentivize negative behaviors.

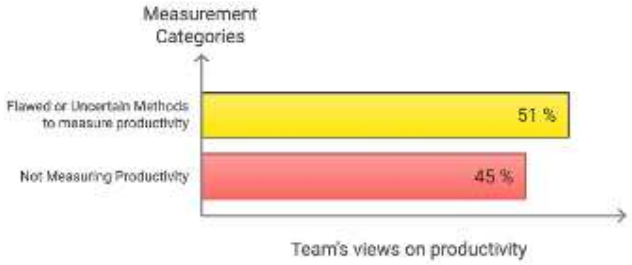

Figure 1: Challenge in Measuring Developer Productivity [6]

II.1. Invisible Bottlenecks

Invisible bottlenecks are subtle obstacles that hinder progress and are not immediately apparent to team members or stakeholders. They reduce team productivity and can delay timelines if not identified and addressed. These bottlenecks may include issues such as longer Pull Request review times, bulky code commits, blocked tickets, or inefficient processes that are not easily quantifiable. Identifying and addressing these invisible bottlenecks is crucial for improving productivity but requires a comprehensive understanding of the development process beyond traditional metrics.

III. GUIDELINES FOR THE PRODUCTIVITY CALCULATOR

The productivity calculator is designed with the following guidelines:

1. Purpose: The calculator aims to estimate a productivity score that reflects team activities during a given period (e.g., a sprint in Agile methodology). It provides quantitative insights into various aspects of the development process, helping teams understand their performance in a structured manner.
2. Non-Comparative Use: The calculator should not be used to measure individual engineer competency or to compare different teams' performance. Productivity is influenced by numerous factors, including project complexity, technology stack, team dynamics, and organizational culture. Comparing productivity scores without considering these contextual elements can lead to misleading conclusions and negatively impact team morale.
3. Self-Assessment Tool: The productivity score is intended for internal team use, similar to sprint velocity in Agile methodology, to help identify trends and areas for improvement. It serves as a tool for reflection and continuous improvement within the team. Productivity scores generated for different teams should not be compared. Each team operates under unique circumstances, with varying goals, challenges, and workflows. By focusing on their own productivity trends over time, teams can set realistic targets, recognize achievements, and address specific issues affecting their performance. This approach encourages a growth mindset and fosters a supportive environment where teams can collaborate effectively without the pressure of external comparisons.

IV. METRICS AND METHODOLOGY

The productivity calculator evaluates team performance by integrating a diverse set of metrics across six key categories: Agile practices, code contributions, code quality, review activities, QA efforts, and deployment metrics. Each variable within these metrics is assigned a specific weight, reflecting its relative importance in the software development process and towards the team's goals. This multi-faceted approach ensures a holistic assessment of the team's productivity by capturing various dimensions of their work.

The productivity score is calculated using the following formula:

$$\text{Productivity} = \frac{\sum_i v_i w_i}{N} \quad (1)$$

where:

- v_i denotes each individual metric value being measured.
- w_i represents the weight factor associated with each metric.
- N is the normalization factor, which in our case is the number of team members avail-able for the sprint duration. The normalization ensures that the productivity index is not adversely affected by team availability.

Assigning weights to each variable enhances the flexibility of the productivity calculator, al-lowing it to be tailored to the specific needs and priorities of different teams. This adaptability is crucial, as not all software development teams operate under the same objectives or constraints. By assigning variable weights, teams can focus on metrics that are most relevant to their unique workflows and goals.

For instance, a research-oriented team, whose primary focus is exploring new technologies or solving complex problems, may place higher importance on the completion of Spike tickets within a sprint. Spike tasks often involve significant uncertainty and require deep investigation, which may not immediately yield visible results but are critical for long-term innovation. By increasing the weight of this metric, the team can better reflect their productivity in terms of exploratory work and knowledge acquisition, rather than just deliverables.

Conversely, a delivery-focused team, where the emphasis is on rapidly deploying features and ensuring a continuous flow of value to end users, might prioritize deployment metrics such as deployment frequency and lead time for changes. These metrics directly measure how quickly and efficiently code is delivered to production, reflecting the team's ability to maintain a stable and effective deployment pipeline. By adjusting the weight toward these metrics, the calculator becomes a more accurate representation of the team's operational performance and delivery efficiency.

This customizable approach allows teams to align their productivity assessment with their specific mission, whether that involves innovation, stability, speed, or any other strategic objective. The ability to fine-tune the calculator's metrics ensures that it remains relevant across a wide variety of team structures, fostering meaningful insights and promoting improvements that are aligned with each team's goals.

IV.1. Agile Practices

Agile practices focus on iterative development, collaboration, and adaptability. Metrics in this category measure how effectively the team plans and executes work during a sprint. For in-stance, tracking the number of story points started and completed provides insight into the team's capacity planning and execution efficiency. Currently, Agile methodologies are among the most effective and widely adopted approaches for measuring productivity in the industry.

The following variables are measured within Agile practices:

- **Story Points Completed:** The total number of story points completed in an Agile sprint. This metric reflects the team's capacity and throughput.
- **Number of Blocked Tickets:** The number of tickets that are blocked during an Ag-ile sprint. This variable acknowledges the work that engineers have invested in tickets before they became blocked. Often, engineering teams expend effort on a task until it is blocked, but this work is not counted in Story Points Completed since the ticket is not completed. By including this variable, we recognize the effort contributed toward blocked tickets. However, a higher number of blocked tickets may indicate impediments affecting productivity.
- **Number of Spikes Executed:** The number of spike tasks (research or exploration tasks) executed during the sprint. Spikes often involve uncertainty and can cause context switching. Teams can assign weight to this metric based on their goals, particularly if innovation or research is a priority.
- **Priority Tickets/Bugs Undertaken Mid-Sprint:** The number of high-priority tasks ad-dressed that were introduced after sprint planning. This variable acknowledges the disruption caused and the context switching required by unexpected work, as well as the team's adaptability in handling these tasks.
- **Cycle Time:** The average time taken to complete a task from start to finish. A shorter cycle time indicates a more efficient workflow.

IV.2. Code Contributions

This category evaluates the team's coding activities, emphasizing the importance of regular and manageable code changes. Metrics include the number of commits and pull requests (PRs) opened, which encourage practices such as frequent commits and smaller, more focused PRs.

- **Number of Commits:** The total number of commits made during the sprint. This metric directly correlates with the volume of changes the team is producing. Encouraging smaller, frequent commits aligns with software engineering best practices and facilitates easier code reviews and integrations.
- **Number of Pull Requests (PRs) Opened:** The number of PRs opened by the team during the sprint. Smaller, more frequent PRs are easier to review and integrate, reducing bottlenecks in the development process.
- **Build Success Rate:** The percentage of builds that succeed without errors. A high build success rate indicates stable code and effective integration practices.

IV.3. Code Quality

Code quality metrics assess the maintainability and reliability of the codebase. They consider factors such as code coverage, unit test quality, and the introduction of new security vulnerabilities.

- **Code Coverage:** The percentage of code covered by automated tests. Higher code coverage can lead to more reliable code but may not fully capture test quality. With each sprint, we can measure the change in total code coverage; the team can be incentivized or penalized based on how the code coverage has changed during the given period.
- **Unit Test Quality:** A qualitative assessment of unit tests, possibly measured by mutation testing

or code review. A unit test quality score goes beyond code coverage and measures how effectively the unit tests uncover potential issues. Measuring test quality is complex; future iterations may include this metric.

- **New Security Vulnerabilities Introduced:** The number of new security vulnerabilities introduced during the sprint. Introducing vulnerabilities hampers productivity, as additional work is required to address them; hence, this metric should have a negative weight.

IV.4. Review Activities

Review activities measure the team's engagement in the code review process, which is crucial for knowledge sharing and maintaining code quality. Metrics include the number of PR reviews conducted, comments left on PRs, and PRs merged. Engaging in review activities helps teams ensure code quality and fosters collaboration by identifying potential issues early and promoting best practices across the team.

- **Number of PR Reviews:** The number of PRs reviewed by team members. This metric encourages collaborative code quality improvement.
- **Number of Comments Left:** The number of comments made during code reviews. This metric highlights engagement in the review process; however, excessive comments may not always be beneficial.
- **Number of PRs Merged:** The total number of PRs merged during the sprint. This reflects the team's throughput in integrating changes.
- **PR Review Comments and Requests for Changes:** The number of comments and change requests made during PR reviews. While constructive feedback is valuable, frequent change requests may indicate issues with initial submissions; thus, this metric may have a negative weight.
- **PR Time in Review:** The average time PRs spend in the review process. Extended review times can clog the workflow and block subsequent PRs awaiting merge.

IV.5. QA Efforts

Quality Assurance (QA) efforts focus on testing and verifying that the software meets the re-quired standards before release. Metrics in this category include the number of issues logged and the number of features tested. A proactive approach to QA helps in delivering a reliable product and reduces post-release incidents.

- **Number of Issues Logged:** The number of issues or bugs identified and logged during the sprint. Identifying issues early can save time and resources; however, a high number of issues may reflect underlying quality problems.
- **Number of Features Tested:** The number of features fully tested during the sprint. Com-prehensive testing ensures reliability and customer satisfaction.

IV.6. Deployment Metrics

Deployment metrics provide insight into the team's efficiency in delivering code changes to production. Although some of these metrics are challenging to automate, they are essential indicators of operational performance. Metrics such as deployment frequency and lead time for changes are considered. For example, if the team deploys code to production twice during a sprint, it indicates a healthy deployment pipeline and the team's ability to deliver value continuously. Shorter lead times for changes suggest efficient processes, though measuring this may require additional tools or automation.

- **Deployment Frequency:** How often code is deployed to production. Frequent deployments can indicate a healthy, agile process.
- **Lead Time for Changes:** The time it takes for a commit to reach production. Shorter lead times suggest an efficient pipeline; this metric may have a negative weight to penalize high lead times.
- **Change Failure Rate:** The percentage of deployments causing failures in production. A high failure rate may indicate quality issues; tracking this metric can be challenging. This metric should have a negative weight.
- **Mean Time to Recovery (MTTR):** The average time taken to recover from production failures. Shorter MTTR is desirable; automating measurement can be a challenge.

IV.7. Normalization Factor

Team size fluctuations from sprint to sprint can significantly affect the assessment of the productivity score. To mitigate this, we introduced a normalization factor. Normalization allows for fair assessment regardless of team size or other contributing factors. In our experiments, we used the team size as the normalization factor. To account for team size, the total productivity score is divided by the number of team members available during the sprint.

V. APPLICATION EXAMPLE

For internal use, we created a JIRA plugin that implements the Engineering Productivity Calculator as described above. The idea was to seamlessly

integrate into existing development practices without introducing another tool for logging work.

V.1. Integrations

- We connected the productivity calculator to development tools like GitHub and JIRA to automatically gather metrics such as Agile Practices, Code Contributions, and Review Activities.
- We plan to integrate the productivity calculator with the CI/CD pipeline to gather metrics like code coverage, unit test quality, and vulnerabilities.
- We also intend to integrate the tool with QA systems and production-facing systems that collect DORA metrics.

The following are the different weightings we used for one of our innovation teams:

Table 1: Metric Weightings for an Innovation Team

Metrics	Weight
Jira Points Completed	1
Blocked Tickets	0.25
Spikes Executed	0.5
Priority Tickets/Bugs Undertaken	0.25
Mid-Sprint	
Cycle Time	1
Number of Commits	0.05
Number of Pull Requests	0.5
Build Success Rate	0.1
Code Coverage	0.1
Unit Test Quality	0.1
Security Vulnerabilities Introduced	-0.2
Number of PR Reviews	0.1
Number of PR Comments	0.05
Number of PRs Merged	0.1
Number of Changes Requested	-0.1
Total PR Review Time	-0.01
Number of Issues Logged	1
Number of Features Tested	1
Deployment Frequency	0.5
Lead Time to Change	-0.1
Change Failure Rate	-0.1
Mean Time to Recover (MTTR)	-0.1

We tested the tool with seven different teams over a period of seven sprints, and the results are shown in Figure 2.

V.2. Insights Discovered

- We observed a significant dip around the fourth sprint cycle. Upon investigation, we realized that it was a week with many holidays and vacations. This showed that the normalization factor we initially used did not account for variations in team availability. We subsequently proposed a new normalization factor, as described in Section VI.2.
- We discovered various insights and correlations between different metrics. For some teams, we noticed that the *PR Review Time* was consistently high. Upon closer analysis, we found that the *PR Review Time* was inversely correlated with the *Number of Commits and Number of Pull Requests*. This reinforces the idea that larger PRs and bigger commits are harder to review. Breaking them down into smaller units could significantly reduce the *PR Review Time*.

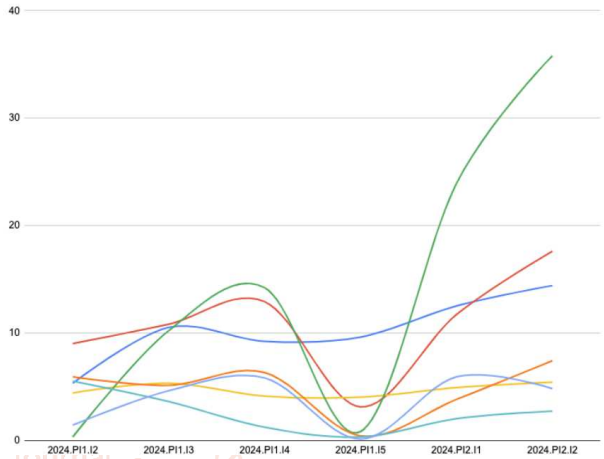

Figure 2: Productivity graph from an experiment with seven different teams over seven sprints.

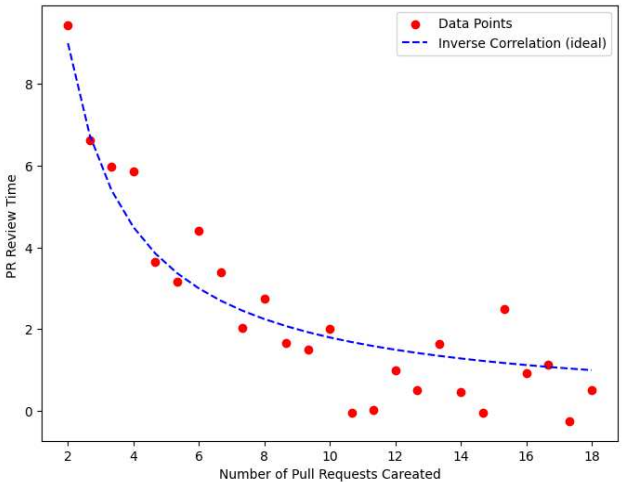

Figure 3: Inverse correlation between the number of PRs and PR Review Time.

- As seen in Figure 2, over the period of seven sprints, we observed trends in productivity that corresponded to changes in team practices, confirming the usefulness of the productivity calculator in tracking and improving team performance.

V.3. Benefits

- **Holistic View:** Considers multiple facets of development, not just code output.

- **Customization:** Weightings can be adjusted to align with team priorities.
- **Continuous Improvement:** Identifies areas where the team excels or needs improvement.

V.4. Limitations

- **Measurement Challenges:** Some metrics are difficult to automate or quantify accurately.
- **Overemphasis on Numbers:** Risk of focusing on metrics over meaningful work.
- **Context Ignored:** Does not account for external factors affecting productivity.

VI. FUTURE WORK

VI.1. Expanded Integrations

We plan to expand the calculator's integrations to include QA systems and DORA metrics collection systems. Integrating with QA systems will allow us to automatically gather metrics related to quality assurance efforts, such as test coverage and defect rates. Incorporating DORA metrics, which include deployment frequency, lead time for changes, change failure rate, and mean time to recovery, will provide deeper insights into the team's DevOps performance.

Additionally, we aim to enhance the assessment of unit test quality by integrating advanced testing tools that can evaluate test effectiveness through techniques like mutation testing.

VI.2. Improved Normalization Factor

We recognize that the initial normalization factor, based solely on team size, may not adequately account for variations in team availability due to holidays, vacations, or part-time team members. We propose exploring more sophisticated normalization factors, such as the total team working days (i.e., the sum of the number of working days each team member was available during the sprint). This approach would account for variations in team availability and provide a more accurate normalization of the productivity score.

VI.3. Application Across Business Verticals

We plan to adapt the productivity calculator for use across different business verticals within the organization. By customizing the weightings and metrics to align with the specific goals and processes of various teams, we can extend the applicability of the calculator beyond software development teams to other departments such as marketing, operations, or sales. This cross-functional approach could foster a unified framework for productivity assessment throughout the organization.

VII. CONCLUSION

Measuring developer productivity is inherently complex, and no single metric can capture all aspects of a team's performance. The proposed productivity calculator provides a balanced assessment by considering a range of activities and their impacts across various dimensions of software development. By using this tool as a guide rather than a strict evaluation metric, teams can foster a culture of continuous improvement and collaboration. The calculator encourages teams to reflect on their processes, identify areas for enhancement, and align their efforts with their specific goals and priorities.

ACKNOWLEDGEMENT

We would like to thank the Newscorp Engineering team for their contributions and feedback in refining the productivity calculator.

DISCLAIMER

The analysis and conclusions contained in this paper represent my personal views and do not necessarily reflect the official policies or opinions of NewsCorp.

REFERENCES

[1] Turing, "How to Measure Developer Productivity," 2023. [Online]. Available: https://www.turing.com/resources/how-to-measure-developer-productivity

[2] GitHub, "Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness," 2023. [Online]. Available: https://github.blog/research-quantifying-github-copilots-impact-on-developer-productivity

[3] G. H., "Measuring Developer Productivity," The Pragmatic Engineer, 2023. [Online]. Available: https://newsletter.pragmaticengineer.com/p/measuring-developer-productivity

[4] GitLab, "DevOps Reports," 2023. [Online]. Available: https://about.gitlab. com/developer-survey/previous/2023/#download

[5] Forbes Technology Council, "How To Measure Developer Productivity In The Age Of AI," 2024. [Online]. Available: https://www.forbes.com/councils/forbestechcouncil/2024/02/02/how-to-measure-developer-productivity-in-the-age-of-ai/

[6] GitLab, "CXO Survey Reports," 2024. [Online]. Available: https://about.gitlab.com/press/releases/2024-06-25-gitlab-survey-reveals-tension-around-ai-security-and-devel