McKinsey & Company

Technology, Media & Telecommunications Practice

Yes, you can measure software developer productivity

Measuring, tracking, and benchmarking developer productivity has long been considered a black box. It doesn't have to be that way.

This article is a collaborative effort by Chandra Gnanasambandam, Martin Harrysson, Alharith Hussin, Jason Keovichit, and Shivam Srivastava, representing views from McKinsey's Digital and Technology, Media & Telecommunications Practices.

August 2023

Compared with other critical business functions such as sales or customer operations, software development is perennially undermeasured. The long-held belief by many in tech is that it's not possible to do it correctly—and that, in any case, only trained engineers are knowledgeable enough to assess the performance of their peers. Yet that status quo is no longer sustainable. Now that most companies are becoming (to one degree or another) software companies, regardless of industry, leaders need to know they are deploying their most valuable talent as successfully as possible.

There is no denying that measuring developer productivity is difficult. Other functions can be measured reasonably well, some even with just a single metric; whereas in software development, the link between inputs and outputs is considerably less clear. Software development is also highly collaborative, complex, and creative work and requires different metrics for different levels (such as systems, teams, and individuals). What's more, even if there is genuine commitment to track productivity properly, traditional metrics can require systems and software that are set up to allow more nuanced and comprehensive measurement. For some standard metrics, entire tech stacks and development pipelines need to be reconfigured to enable tracking, and putting in place the necessary instruments and tools to yield meaningful insights can require significant, long-term investment. Furthermore, the landscape of software development is changing quickly as generative AI tools such as Copilot X and ChatGPT have the potential to enable developers to complete tasks up to two times faster.

To help overcome these challenges and make this critical task more feasible, we developed an approach to measuring software developer productivity that is easier to deploy with surveys or existing data (such as in backlog management tools). In so doing, we built on the foundation of existing productivity metrics that industry leaders have developed over the years, with an eye toward revealing opportunities for performance improvements.

This new approach has been implemented at nearly 20 tech, finance, and pharmaceutical companies, and the initial results are promising. They include the following improvements:

- 20 to 30 percent reduction in customer-reported product defects
- 20 percent improvement in employee experience scores
- 60-percentage-point improvement in customer satisfaction ratings

Now that most companies are becoming software companies, leaders need to know they are deploying their most valuable talent as successfully as possible.

Leveraging productivity insights

With access to richer productivity data and insights, leaders can begin to answer pressing questions about the software engineering talent they fought so hard to attract and retain, such as the following:

— What are the impediments to the engineers working at their best level?

— How much does culture and organization affect their ability to produce their best work?

— How do we know if we're using their time on activities that truly drive value?

— How can we know if we have all the software engineering talent we need?

Understanding the foundations

To use a sufficiently nuanced system of measuring developer productivity, it's essential to understand the three types of metrics that need to be tracked: those at the system level, the team level, and the individual level. Unlike a function such as sales, where a system-level metric of dollars earned or deals closed could be used to measure the work of both teams and individuals, software development is collaborative in a distinctive way that requires different lenses. For instance, while deployment frequency is a perfectly good metric to assess systems or teams, it depends on all team members doing their respective tasks and is, therefore, not a useful way to track individual performance.

Another critical dimension to recognize is what the various metrics do and do not tell you. For example, measuring deployment frequency or lead time for changes can give you a clear view of certain outcomes, but not of whether an engineering organization is optimized. And while metrics such as story points completed or interruptions can help determine optimization, they require more investigation to identify improvements that might be beneficial.

In building our set of metrics, we looked to expand on the two sets of metrics already developed by the software industry. The first is DORA metrics, named for Google's DevOps research and assessment team. These are the closest the tech sector has to a standard, and they are great at measuring outcomes. When a DORA metric returns a subpar outcome, it is a signal to investigate what has gone wrong, which can often involve protracted sleuthing. For example, if a metric such as deployment frequency increases or decreases, there can be multiple causes. Determining what they are and how to resolve them is often not straightforward.

Our approach seeks to identify what can be done to improve how products are delivered and what those improvements are worth, without the need for heavy instrumentation.

The second set of industry-developed measurements is SPACE metrics (satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow), which GitHub and Microsoft Research developed to augment DORA metrics. By adopting an individual lens, particularly around developer well-being, SPACE metrics are great at clarifying whether an engineering organization is optimized. For example, an increase in interruptions that developers experience indicates a need for optimization.

On top of these already powerful metrics, our approach seeks to identify what can be done to improve how products are delivered and what those improvements are worth, without the need for heavy instrumentation. Complementing DORA and SPACE metrics with opportunity-focused metrics can create an end-to-end view of software developer productivity (Exhibit 1).

These opportunity-focused productivity metrics use a few different lenses to generate a nuanced view of the complex range of activities involved with software product development.

Inner/outer loop time spent. To identify specific areas for improvement, it's helpful to think of the

Exhibit 1

Adding a focus on opportunities to software developer productivity metrics can offer clearer paths to improvement.

Focus areas by level

● DORA[1] metrics ● SPACE[2] metrics ● Opportunity-focused metrics

	Outcomes focus *Are you delivering products satisfactorily?*	**Optimization focus**[3] *Are you delivering products in an optimized way?*	**Opportunities focus** *Are there specific opportunities to improve how you deliver products, and what are they worth?*
System level	● Deployment frequency ● Customer satisfaction ● Reliability (uptime)	● Code-review timing ● Velocity/flow through the system	● Satisfaction with engineering system ● Inner/outer loop time spent
Team level	● Lead time for changes ● Change failure rate ● Time to restore service ● Code-review velocity	● Story points completed ● Handoffs	● Quality of documentation ● Developer Velocity Index benchmark[4] ● Contribution analysis
Individual level	● Developer satisfaction ● Retention	● Interruptions	● Contribution analysis ● Talent capability score

[1]Google's DevOps research and assessment team, which developed these outcome metrics.
[2]Satisfaction and well-being, performance, activity, communication and collaboration, and efficiency and flow; GitHub and Microsoft Research developed these metrics, which aim to look at developer well-being as a measurement at the individual level.
[3]Nonexhaustive.
[4]Benchmarks an organization's technology, working practices, and organizational enablement; see Shivam Srivastava, Kartik Trehan, Dilip Wagle, and Jane Wang, "Developer Velocity: How software excellence fuels business performance," McKinsey, Apr 20, 2020.

McKinsey & Company

activities involved in software development as being arranged in two loops (Exhibit 2). An inner loop comprises activities directly related to creating the product: coding, building, and unit testing. An outer loop comprises other tasks developers must do to push their code to production: integration, integration testing, releasing, and deployment. From both a productivity and personal-experience standpoint, maximizing the amount of time developers spend in the inner loop is desirable: building products directly generates value *and* is what most developers are excited to do. Outer-loop activities are seen by most developers as necessary but generally unsatisfying chores. Putting time into better tooling and automation for the outer loop allows developers to spend more time on inner-loop activities.

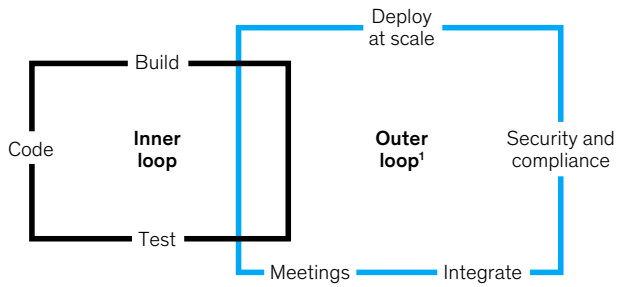
Top tech companies aim for developers to spend up to 70 percent of their time doing inner-loop activities. For example, one company that had previously completed a successful agile transformation learned that its developers, instead of coding, were spending too much time on low-value-added tasks such as provisioning infrastructure, running manual unit tests, and managing test data. Armed with that insight, it launched a series of new tools and automation projects to help with those tasks across the software development life cycle.

Developer Velocity Index benchmark. The Developer Velocity Index (DVI) is a survey that measures an enterprise's technology, working practices, and organizational enablement and benchmarks them against peers. This comparison helps unearth specific areas of opportunity, whether in backlog management, testing, or security and compliance.[1] For example, one company, well known for its technological prowess and all-star developers, sought to define standard working practices more thoughtfully for cross-team collaboration after discovering a high amount of dissatisfaction, rework, and inefficiency reported by developers.

Exhibit 2

Software development can be broadly divided into two sets, or loops, of tasks; the less time spent on less fulfilling, outer-loop activities, the better.

Software development activities



[1]Activities listed are nonexhaustive.

McKinsey & Company

[1] To read more about McKinsey's DVI survey, see Shivam Srivastava, Kartik Trehan, Dilip Wagle, and Jane Wang, "Developer velocity: How software excellence fuels business performance," McKinsey, April 20, 2020; and Chandra Gnanasambandam, Neha Jindal, Shivam Srivastava, and Dilip Wagle, "Developer velocity at work: Key lessons from industry leaders," McKinsey, February 22, 2021.

Contribution analysis. Assessing contributions by individuals to a team's backlog (starting with data from backlog management tools such as Jira, and normalizing data using a proprietary algorithm to account for nuances) can help surface trends that inhibit the optimization of that team's capacity. This kind of insight can enable team leaders to manage clear expectations for output and improve performance as a result. Additionally, it can help identify opportunities for individual upskilling or training and rethinking role distribution within a team (for instance, if a quality assurance tester has enough work to do). For example, one company found that its most talented developers were spending excessive time on noncoding activities such as design sessions or managing interdependencies across teams. In response, the company changed its operating model and clarified roles and responsibilities to enable those highest-value developers to do what they do best: code. Another company, after discovering relatively low contribution from developers new to the organization, reexamined their onboarding and personal mentorship program.

Talent capability score. Based on industry standard capability maps, this score is a summary of the individual knowledge, skills, and abilities of a specific organization. Ideally, organizations should aspire to a "diamond" distribution of proficiency, with the majority of developers in the middle range of competency.[2] This can surface coaching and upskilling opportunities, and in extreme cases call for a rethinking of talent strategy. For example, one company found a higher concentration of their developers in the "novice" capability than was ideal. They deployed personalized learning journeys based on specific gaps and were able to move 30 percent of their developers to the next level of expertise within six months.

Avoiding metrics missteps

As valuable as it can be, developer productivity data can be damaging to organizations if used incorrectly, so it's important to avoid certain pitfalls. In our work we see two main types of missteps occur: misuse of metrics and failing to move past old mindsets.

Misuse is most common when companies try to employ overly simple measurements, such as lines of code produced, or number of code commits (when developers submit their code to a version control system). Not only do such simple metrics fail to generate truly useful insights, they can have unintended consequences, such as leaders making inappropriate trade-offs. For example, optimizing for lead time or deployment frequency can allow quality to suffer. Focusing on a single metric or too simple a collection of metrics can also easily incentivize poor practices; in the case of measuring commits, for instance, developers may submit smaller changes more frequently as they seek to game the system.

To truly benefit from measuring productivity, leaders and developers alike need to move past the outdated notion that leaders "cannot" understand the intricacies of software engineering, or that engineering is too complex to measure. The importance of engineering talent to a company's success, and the fierce competition for developer talent in recent years, underscores the need to acknowledge that software development, like so many other things, requires measurement to be improved. Further, attracting and retaining top software development talent depends in large part on providing a workplace and tools that allow engineers to do their best work and encourages their creativity. Measuring productivity at a system level enables employers to see hidden friction points that impede that work and creativity.

[2] Klemens Hjartar, Peter Jacobs, Eric Lamarre, and Lars Vinter, "It's time to reset the IT talent model," *MIT Sloan Management Review*, March 5, 2020.

Getting started

The mechanics of building a developer productivity initiative can seem daunting, but there is no time like the present to begin to lay the groundwork. The factors driving the need to elevate the conversation about software developer productivity to C-level leaders outweigh the impediments to doing so.

The increase in remote work and its popularity among developers is one overriding factor. Developers have long worked in agile teams, collaborating in the same physical space, and some technology leaders believe that kind of in-person teamwork is essential to the job. However, the digital tools that are so central to their work made it easy to switch to remote work during the pandemic lockdowns, and as in most sectors, this shift is hard to undo. As remote and hybrid working increasingly becomes the norm, organizations will need to rely on broad, objective measurements to maintain confidence in these new working arrangements and ensure they are steadily improving the function that could easily determine their future success or failure. The fact that the markets are now putting greater emphasis on efficient growth and ROI only makes it more important than ever to know how they can optimize the performance of their highly valued engineering talent.

Another key driver of this need for greater visibility is the rapid advances in AI-enabled tooling, especially large-language models such as generative AI. These are already rapidly changing the way work is done, which means that measuring software developers' productivity is only a first step to understanding how these valuable resources are deployed.

But as critical as developer productivity is becoming, companies shouldn't feel they have to embark on a massive, dramatic overhaul almost overnight. Instead, they can begin the process with a number of key steps:

Learn the basics. All C-suite leaders who are not engineers or who have been in management for a long time will need a primer on the software development process and how it is evolving.

Assess your systems. Because developer productivity has not typically been measured at the level needed to identify improvement opportunities, most companies' tech stacks will require potentially extensive reconfiguration. For example, to measure test coverage (the extent to which areas of code have been adequately tested), a development team needs to equip their codebase with a tool that can track code executed during a test run.

Build a plan. As with most analytics initiatives, getting lost in mountains of data is a risk. It's important to start with one area that you know will result in a clear path to improvement, such as identifying friction points and bottlenecks. Be explicit about the scope of such a plan, as even the best approaches, no matter how comprehensive, will not be a silver bullet.

Remember that measuring productivity is contextual. The point is to look at an entire system and understand how it can work better by improving the development environment at the system, team, or individual level.

No matter the specific approach, measuring productivity should ideally create transparency and insights into key improvement areas. Only then can organizations build specific initiatives to drive impact for both developer productivity and experience—impact that will benefit both those individuals and the company as a whole.

Chandra Gnanasambandam and **Martin Harrysson** are senior partners in McKinsey's Bay Area office, where **Alharith Hussin** and **Shivam Srivastava** are partners; and **Jason Keovichit** is an associate partner in the New York office.

The authors wish to thank Pedro Garcia, Diana Rodriguez, and Jeremy Schneider for their contributions to this article.

Designed by McKinsey Global Publishing