



Article development led by [acmqueue](https://queue.acm.org)  
queue.acm.org

## There's more to it than you think.

BY NICOLE FORSGREN, MARGARET-ANNE STOREY,  
CHANDRA MADDILA, THOMAS ZIMMERMANN,  
BRIAN HOUCK, AND JENNA BUTLER

# The SPACE of Developer Productivity

DEVELOPER PRODUCTIVITY IS complex and nuanced, with important implications for software development teams. A clear understanding of defining, measuring, and predicting developer productivity could provide organizations, managers, and developers with the ability to make higher-quality software—and make it more efficiently.

Developer productivity has been studied extensively. Unfortunately, after decades of research and practical development experience, knowing how to measure productivity or even define developer productivity has remained elusive, while myths about the topic are common. Far too often teams or managers attempt to measure developer productivity with simple metrics, attempting to capture it all with “one metric that matters.”

One important measure of productivity is personal perception;<sup>1</sup> this may resonate with those who claim to be in “a flow” on productive days.

There is also agreement that developer productivity is necessary not just to improve engineering outcomes, but also to ensure the well-being and satisfaction of developers, as productivity and satisfaction are intricately connected.<sup>12,20</sup>

Ensuring the efficient development of software systems and the well-being of developers has never been more important as the Covid-19 pandemic has forced the majority of software developers worldwide to work from home,<sup>17</sup> disconnecting developers and managers from their usual workplaces and teams. Although this was unexpected and unfortunate, this change constitutes a rare “natural experiment” that statisticians can capitalize upon to study, compare, and understand developer productivity across many different contexts. This forced disruption and the future transition to hybrid remote/colocated work expedites the need to understand developer productivity and well-being, with wide agreement that doing so in an efficient and fair way is critical.

This article explicates several common myths and misconceptions about developer productivity. *The most important takeaway from exposing these myths is that productivity cannot be reduced to a single dimension (or metric!).* The prevalence of these myths and the need to bust them motivated our work to develop a practical multidimensional framework, because only by examining a constellation of metrics in tension can we understand and influence developer productivity. This framework, called SPACE, captures the most important dimensions of developer productivity: satisfaction and well-being; performance; activity; communication and collaboration; and efficiency and flow. By recognizing and measuring productivity with more than just a single dimension, teams and organizations can better understand how people and teams work, and they can make better decisions.

The article demonstrates how this framework can be used to understand productivity in practice and why using



it will help teams better understand developer productivity, create better measures to inform their work and teams, and may positively impact engineering outcomes and developer well-being.

### Myths and Misconceptions About Developer Productivity

A number of myths about developer productivity have accumulated over the years. Awareness of these misconceptions leads to a better understanding of measuring productivity.

**Myth: Productivity is all about developer activity.** This is one of the most common myths, and it can cause undesirable outcomes and developer dissatisfaction. Sometimes, higher volumes of activity appear for various reasons: working longer hours may signal developers having to “brute-force” work to overcome bad systems or poor planning to meet a predefined release schedule. On the other hand,

increased activity may reflect better engineering systems, providing developers with the tools they need to do their jobs effectively, or better collaboration and communication with team members in unblocking their changes and code reviews.

Activity metrics alone do not reveal which of these is the case, so they should never be used in isolation either to reward or to penalize developers. Even straightforward metrics such as number of pull requests, commits, or code reviews are prone to errors because of gaps in data and measurement errors, and systems that report these metrics will miss the benefits of collaboration seen in peer programming or brainstorming. Finally, developers often flex their hours to meet deadlines, making certain activity measures difficult to rely on in assessing productivity.

**Myth: Productivity is only about individual performance.** While individual

performance is important, contributing to the success of the team is also critical to measuring productivity. Measures of performance that balance the developer, team, and organization are important. Similar to team sports, success is judged both by a player’s personal performance as well as the success of their team. A developer who optimizes only for their own personal productivity may hurt the productivity of the team. More team-focused activities such as code reviews, on-call rotations, and developing and managing engineering systems help maintain the quality of the code base and the product/service. Finding the right balance in optimizing for individual, team, and organizational productivity, as well as understanding possible trade-offs, is key.

**Myth: One productivity metric can tell us everything.** One common myth about developer productivity is that it produces a universal metric, and that

this “one metric that matters” can be used to score teams on their overall work and to compare teams across an organization and even an industry. This isn’t true. Productivity represents several important dimensions of work and is greatly influenced by the context in which the work is done.

**Myth: Productivity measures are useful only for managers.** Developers often say that productivity measures aren’t useful. This may come from the misuse of measures by leaders or managers, and it’s true that when productivity is poorly measured and implemented, it can lead to inappropriate usage in organizations. It’s disappointing that productivity has been co-opted this way, but it’s important to note that developers have found value in tracking their own productivity—both for personal reasons and for communicating with others.

By remembering that developer productivity is personal,<sup>7</sup> developers can leverage it to gain insights into their work so they can take control of their time, energy, and days. For example, research has shown that high productivity is highly correlated with feeling satisfied and happy with work.<sup>12,20</sup> Finding ways to improve productivity is also about finding ways to introduce more joy, and decrease frustration, in a developer’s day.

**Myth: Productivity is only about engineering systems and developer tools.** While developer tools and workflows have a large impact on developer productivity, human factors such as environment and work culture have substantial impact too. Often the critical work needed to keep the environment and culture healthy can be “invisible” to many members of the organization or to metrics traditionally used for measuring productivity. Work such as morale building, mentoring, and knowledge sharing are all critical to supporting a productive work environment and yet are often not measured. The “invisible” work that benefits the overall productivity of the team is just as important as other more commonly measured dimensions.<sup>21</sup>

### SPACE: A Framework for Understanding Developer Productivity

Productivity is about more than the individual or the engineering systems; it

cannot be measured by a single metric or activity data alone; and it isn’t something that only managers care about. The SPACE framework was developed to capture different dimensions of productivity because without it, the myths just presented will persist. The framework provides a way to think rationally about productivity in a much bigger space and to choose metrics carefully in a way that reveals not only what those metrics mean, but also what their limitations are if used alone or in the wrong context.

**Satisfaction and well-being.** *Satisfaction* is how fulfilled developers feel with their work, team, tools, or culture; *well-being* is how healthy and happy they are, and how their work impacts it. Measuring satisfaction and well-being can be beneficial for understanding productivity<sup>20</sup> and perhaps even for predicting it.<sup>15</sup> For example, **productivity and satisfaction** are correlated, and it is possible that satisfaction could serve as a leading indicator for productivity; a decline in satisfaction and engagement could signal upcoming burnout and reduced productivity.<sup>13</sup>

For example, when many places shifted to mandatory work from home during the pandemic, an uptick occurred in some measures of productivity (for example, code commits and speed to merge pull requests).<sup>8</sup> Qualitative data, however, has shown that some people were struggling with their well-being.<sup>3</sup> This highlights the importance of balanced measures that capture several aspects of productivity: While some activity measures looked positive, additional measures of satisfaction painted a more holistic picture, showing that productivity is personal, and some developers were approaching burnout. To combat this, some software groups in large organizations implemented “mental health” days—essentially, free days off to help people avoid burnout and improve well-being.

It is clear that satisfaction and well-being are important dimensions of productivity. These qualities are often best captured with surveys. To assess the satisfaction dimension, you might measure the following:

► *Employee satisfaction.* The degree of satisfaction among employees, and

whether they would recommend their team to others.

► *Developer efficacy.* Whether developers have the tools and resources they need to get their work done.

► *Burnout.* Exhaustion caused by excessive and prolonged workplace stress.

**Performance** is the outcome of a system or process. The performance of software developers is hard to quantify, because it can be difficult to tie individual contributions directly to product outcomes. A developer who produces a large amount of code may not be producing high-quality code. High-quality code may not deliver customer value. Features that delight customers may not always result in positive business outcomes. Even if a particular developer’s contribution can be tied to business outcomes, it is not always a reflection of performance since the developer may have been assigned a less impactful task, instead of having agency to choose more impactful work. Furthermore, software is often the sum of many developers’ contributions, exacerbating the difficulty in evaluating the performance of any individual developer. In many companies and organizations, software is written by teams, not individuals.

For these reasons, **performance is often best evaluated as outcomes instead of output.** The most simplified view of software developer performance could be, Did the code written by the developer reliably do what it was supposed to do? Example metrics to capture the performance dimension include:

► *Quality.* Reliability, absence of bugs, ongoing service health.

► *Impact.* Customer satisfaction, customer adoption and retention, feature usage, cost reduction.

**Activity** is a count of actions or outputs completed in the course of performing work. Developer activity, if measured correctly, can provide valuable but limited insights about developer productivity, engineering systems, and team efficiency. Because of the complex and diverse activities that developers perform, their activity is not easy to measure or quantify. In fact, it is *almost impossible to comprehensively measure and quantify all the facets of developer activity across engineering systems and environments.* A well-designed

engineering system, however, will help in capturing activity metrics along different phases of the software development life cycle and quantify developer activity at scale. Some of the developer activities that can be measured and quantified relatively easily are:

- *Design and coding.* Volume or count of design documents and specs, work items, pull requests, commits, and code reviews.

- *Continuous integration and deployment.* Count of build, test, deployment/release, and infrastructure utilization.


- *Operational activity.* Count or volume of incidents/issues and distribution based on their severities, on-call participation, and incident mitigation.

These metrics can be used as waypoints to measure some tractable developer activities, but they should never be used in isolation to make decisions about individual or team productivity because of their known limitations. They serve as templates to start with and should be customized based on organizational needs and development environments. As mentioned earlier, many activities that are essential to developing software are intractable (such as attending team meetings, participating in brainstorming, helping other team members when they encounter issues, and providing architectural guidance, to name a few).


#### **Communication and collaboration.**

*Communication and collaboration* capture how people and teams communicate and work together. Software development is a collaborative and creative task that relies on extensive and effective communication, coordination, and collaboration within and between teams.<sup>11</sup> Effective teams that successfully contribute to and integrate each other's work efficiently rely on high transparency<sup>5</sup> and awareness<sup>6</sup> of team member activities and task priorities. In addition, how information flows within and across teams impacts the availability and discoverability of documentation that is needed for the effective alignment and integration of work. Teams that are diverse and inclusive are higher performing.<sup>22</sup> More effective teams work on the right problems, are more likely to be successful at brainstorming new ideas and will choose better solutions from all the alternatives.

Work that contributes to a team's



**Productivity and satisfaction are correlated, and it is possible that satisfaction could serve as a leading indicator for productivity.**



outcomes or supports another team member's productivity may come at the expense of an individual's productivity and their own ability to get into a state of flow, potentially reducing motivation and satisfaction. Effective collaboration, however, can drive down the need for some individual activities (for example, unnecessary code reviews and rework), improve system performance (faster pull request merges may improve quality by avoiding bugs), and help sustain productivity and avoid (or conversely, if not done right, increase) burnout.

Understanding and measuring team productivity and team member expectations are, however, complicated because of items that are difficult to measure such as invisible work<sup>21</sup> and articulation work for coordinating and planning team tasks.<sup>18</sup> That said, the following are examples of metrics that may be used as proxies to measure communication, collaboration, and coordination:

- Discoverability of documentation and expertise.

- How quickly work is integrated.

- Quality of reviews of work contributed by team members.

- Network metrics that show who is connected to whom and how.

- Onboarding time for and experience of new members.

**Efficiency and flow.** Finally, *efficiency* and *flow* capture the ability to complete work or make progress on it with minimal interruptions or delays, whether individually or through a system. This can include how well activities within and across teams are orchestrated and whether continuous progress is being made.

Some research associates productivity with the ability to get complex tasks done with minimal distractions or interruptions.<sup>2</sup> *This conceptualization of productivity is echoed by many developers when they talk about "getting into the flow" when doing their work—or the difficulty in finding and optimizing for it, with many books and discussions addressing how this positive state can be achieved in a controlled way.*<sup>4</sup> For individual efficiency (flow), it's important to set boundaries to get productive and stay productive—for example, by blocking off time for a focus period. Individual efficiency is of-

ten measured by uninterrupted focus time or the time within value-creating apps (for example, the time a developer spends in the integrated development environment is likely to be considered “productive” time).

At the team and system level, efficiency is related to value-stream mapping, which captures the steps needed to take software from idea and creation to delivering it to the end customer. To optimize the flow in the value stream, it is important to minimize delays and handoffs. The DORA (DevOps Research and Assessment) framework introduced several metrics to monitor flow within teams<sup>9</sup>—for example, deployment frequency measures how often an organization successfully releases to production, and lead time for changes measures the amount of time it takes a commit to get into production.

In addition to the flow of changes through the system, the flow of knowledge and information is important. Certain aspects of efficiency and flow may be difficult to measure, but it is often possible to spot and remove inefficiencies in the value stream. Activities that produce no value for the customer or user are often referred to as software development waste<sup>19</sup>—for example, duplicated work, rework because the work was not done correctly, or time-consuming rote activities.

Some example metrics to capture the efficiency and flow dimension are:

- Number of handoffs in a process; number of handoffs across different teams in a process.

- Perceived ability to stay in flow and complete work.

- Interruptions: quantity, timing, how spaced, impact on development work and flow.

- Time measures through a system: total time, value-added time, wait time.

**Efficiency is related to all the SPACE dimensions.** Efficiency at the individual, team, and system levels has been found to be positively associated with increased satisfaction. Higher efficiency, however, may also negatively affect other factors. For example, maximizing flow and speed may decrease the quality of the system and increase the number of bugs visible to customers (performance). Optimizing for individual efficiency by reducing interruptions may decrease the ability to collaborate, block others’ work, and reduce the ability of the team to brainstorm.

### Framework in Action

To illustrate the SPACE framework, Figure 1 lists concrete metrics that fall into each of the five dimensions. The figure provides examples of individual-, team- or group-, and system-level

measures. Three brief discussions about these metrics follow: First, an example set of metrics concerning code review is shown to cover all dimensions of the SPACE framework, depending on how they are defined and proxied. Next, additional examples are provided for two select dimensions of the framework: activity, and efficiency and flow. The section closes with a discussion of how to use the framework: combining metrics for a holistic understanding of developer productivity, as well as cautions. The accompanying sidebar shows how the framework can be used for understanding productivity in incident management.

Let’s begin with code review as an example scenario that presents a set of metrics that can cover all five dimensions of the SPACE framework, depending on how it is framed and which metric is used:

- *Satisfaction.* Perceptual measures about code reviews can reveal whether developers view the work in a good or bad light—for example if they present learning, mentorship, or opportunities to shape the codebase. This is important, because the number of code reviews per developer may signal dissatisfaction if some developers feel they are consistently assigned a disproportionate amount of code re-

#### Example metrics.

Level	Satisfaction and well-being How fulfilled, happy, and healthy one is	Performance An outcome of a process	Activity The count of actions or outputs	Communication and collaboration How people talk and work together	Efficiency and flow Doing work with minimal delays or interruptions
<b>Individual</b> One person	<ul style="list-style-type: none"> <li>► Developer satisfaction</li> <li>► Retention†</li> <li>► Satisfaction with code reviews assigned</li> <li>► Perception of code reviews</li> </ul>	<ul style="list-style-type: none"> <li>► Code review velocity</li> </ul>	<ul style="list-style-type: none"> <li>► Number of code reviews completed</li> <li>► Coding time</li> <li>► # Commits</li> <li>► Lines of code†</li> </ul>	<ul style="list-style-type: none"> <li>► Code review score (quality or thoughtfulness)</li> <li>► PR merge times</li> <li>► Quality of meetings†</li> <li>► Knowledge sharing, discoverability (quality of documentation)</li> </ul>	<ul style="list-style-type: none"> <li>► Code review timing</li> <li>► Productivity perception</li> <li>► Lack of interruptions</li> </ul>
<b>Team or Group</b> People that work together	<ul style="list-style-type: none"> <li>► Developer satisfaction</li> <li>► Retention†</li> </ul>	<ul style="list-style-type: none"> <li>► Code review velocity</li> <li>► Story points shipped†</li> </ul>	<ul style="list-style-type: none"> <li>► # Story points completed†</li> </ul>	<ul style="list-style-type: none"> <li>► PR merge times</li> <li>► Quality of meetings†</li> <li>► Knowledge sharing or discoverability (quality of documentation)</li> </ul>	<ul style="list-style-type: none"> <li>► Code review timing</li> <li>► Handoffs</li> </ul>
<b>System</b> End-to-end work through a system (like a development pipeline)	<ul style="list-style-type: none"> <li>► Satisfaction with engineering system (e.g., CI/CD pipeline)</li> </ul>	<ul style="list-style-type: none"> <li>► Code review velocity</li> <li>► Code review (acceptance rate)</li> <li>► Customer satisfaction</li> <li>► Reliability (uptime)</li> </ul>	<ul style="list-style-type: none"> <li>► Frequency of deployments</li> </ul>	<ul style="list-style-type: none"> <li>► Knowledge sharing, discoverability (quality of documentation)</li> </ul>	<ul style="list-style-type: none"> <li>► Code review timing</li> <li>► Velocity/ flow through the system</li> </ul>

† Use these metrics with (even more) caution — they can proxy more things.

views, leaving them with less time for other work.

► *Performance.* Code-review velocity captures the speed of reviews; because this can reflect both how quickly an individual completes a review and the constraints of the team, it is both an individual- and a team-level metric. (For example, an individual could complete a review within an hour of being assigned, but a team could have a policy of leaving all reviews open for 24 hours to allow all team members to see the proposed changes.)

► *Activity.* Number of code reviews completed is an individual metric capturing how many reviews have been completed in a given time frame and contributes to the final product.

► *Communication and collaboration.* Code reviews themselves are a way that developers collaborate through code, and a measure or score of the quality or thoughtfulness of code reviews is a great qualitative measure of collaboration and communication.

► *Efficiency and flow.* Code review is important but can cause challenges if it interrupts workflow or if delays cause constraints in the system. Similarly, having to wait for a code review can delay a developer's ability to continue working. Batching up code reviews so they don't interrupt a developer's coding time (which would impact individual measures), while also not causing delays in the throughput of the system (which impacts system measures), allows teams to deliver code efficiently (team-level measures). Therefore, measuring the effects of code-review timing on the efficiency and flow of individuals, teams, and the system is important—this can be done through perceptual or telemetry measures that capture the time to complete reviews and the characteristics of interruptions (such as timing and frequency).

Let's examine the SPACE framework in more depth by looking further at the dimensions of activity and efficiency and flow. In this example, the activity measures are individual-level metrics: number of commits, coding time (total time spent or times of day), and number of code reviews completed. These best describe work that directly contributes to the final product, understanding that work

## SPACE and SRE: The Framework in Incident Management

The SPACE framework is relevant for SREs (site reliability engineers) and their work in IM (incident management). An incident occurs when a service is not available or is not performing as defined in the SLA (service-level agreement). An incident can be caused by network issues, infrastructure problems, hardware failures, code bugs, or configuration issues, to name a few.

Based on the magnitude of the impact caused by an incident, it is typically assigned a severity level (sev-1 being the highest). An outage to the entire organization's customer-facing systems is treated differently than a small subset of internal users experiencing a delay in their email delivery.

Here are some of the common myths associated with IM:

► **MYTH:** Number of incidents resolved by an individual is all that matters. Like a lot of other activities in the SDLC (software development life cycle), IM is a team activity. A service that causes a lot of outages and takes more hours to restore reflects badly on the entire team that develops and maintains the service. More team-focused activities such as knowledge sharing, preparing troubleshooting guides to aid other team members, mentoring juniors and new members of the team, doing proper handoffs and assignment/re-assignments are important aspects of IM.

► **MYTH:** Looking at one metric in isolation will tell you everything. It is important to understand the metrics in context: the number of incidents, how long they took to resolve—the volume and resolution times of sev-1 incidents compared with sev-4, and other factors relevant to understanding incidents and how to improve both the system and the team's response. So, there is no “one metric that matters.”

► **MYTH:** Only management cares about incident volume and meeting SLAs. With the rise of DevOps, developers are also doing operations now. IM (a part of operations) can take away a significant chunk of developers' time and energy if the volume and severity of the incidents are high. As important as it is to management and executives to guarantee SLAs and reduce incident volume and resolution times, it is equally important to the individual developers who are part of the IM process.

► **MYTH:** Effective IM is just about improving systems and tools. Better monitoring systems, ticketing systems, case-routing systems, log-analysis systems, etc. will help make developers productive. While tools, guides, and workflows have a large impact on productivity, the human factors of the environment and work culture have substantial impact too. Mentoring new members of the team and morale building are important. If developers are constantly being paged in the night for sev-1 incidents while working from home during COVID-19, these “invisible” factors are especially helpful to make them more productive.

Incident management is a complex process that involves various stakeholders performing several individual and team activities, and it requires support from different tools and systems, so it is critical to identify metrics that can capture various dimensions of productivity:

► *Satisfaction:* How satisfied SREs are with the IM process, escalation and routing, and on-call rotations are key metrics to capture, especially since burnout is a significant issue among SREs.

► *Performance:* These measures focus on system reliability; monitoring systems' ability to detect and flag issues faster, before they hit the customer and become an incident. MTTR (mean time to repair) overall, and by severity.

► *Activity:* Number of issues caught by the monitoring systems, number of incidents created, number of incidents resolved—and their severity distribution.

► *Communication and collaboration:* People included in resolving the incident, how many teams those people came from, and how they communicate during an incident. Incident resolution documentation outlines the steps involved in resolving incidents; this can be measured by completeness (to check if any resolution data was entered) or quick quality scores (for example, thumbs up/down). Teams may also include a metric that measures the percentage of incidents resolved that reference these guides and documentation.

► *Efficiency and flow:* Incident handoffs, incident assignment/reassignment, number of hops an incident has to take before it is assigned to the right individual or team.

patterns and behaviors are influenced by the teams and environments in which developers work.

Efficiency and flow have a broader mix of metrics. Self-reported measures of productivity are best captured at the individual level: asking a developer whether the team is productive is subject to blind spots, while asking if that

member felt productive or was able to complete work with minimal distractions is a useful signal. You can also measure the flow of work—whether code, documents, or other items—through a system, and capture metrics such as the time it takes or the number of handoffs, delays, and errors in the software delivery pipeline. These

would constitute system-level metrics, because their values would capture the journey of the work item through the entire workflow, or system.

### How To Use the Framework

To measure developer productivity, teams and leaders (and even individuals) should capture several metrics across multiple dimensions of the framework—at least three are recommended. For example, if you are already measuring commits (an activity measure), don't simply add the number of pull requests and coding time to your metrics dashboard, as these are both activity metrics. Adding these can help round out the way you capture the activity dimension of productivity, but to really understand productivity, add at least one metric from two different dimensions: perhaps perception of productivity and pull request merge time.

Another recommendation is that at least one of the metrics include perceptual measures such as survey data. By including perceptions about people's lived experiences, a more complete picture of productivity can be constructed. Many times, perceptual data may provide more accurate and complete information than what can be observed from instrumenting system behavior alone.<sup>10</sup>

Including metrics from multiple dimensions and types of measurements often creates metrics in tension; this is by design, because a balanced view provides a truer picture of what is happening in your work and systems. This more balanced view should help to reinforce smarter decisions and trade-offs among team members, who may otherwise understandably focus on one aspect of work to the detriment of the whole system.

One example is story points, a metric commonly used in Agile development processes to assess team-level progress. If a team is rated only on story points, members will focus on optimizing their own points, to the detriment of completing potentially invisible work that is important to other developers' progress and to the company if that means collaborating with other teams or onboarding future developers. And if leaders measured progress using story points without asking developers about their ability to work quickly, they wouldn't be able to

identify if something wasn't working and the team was doing workarounds and burning out, or if a new innovation was working particularly well and could be used to help other teams that may be struggling.

This leads to an important point about metrics and their effect on teams and organizations: They signal what is important. One way to see indirectly what is important in an organization is to see what is measured, because that often communicates what is valued and influences the way people behave and react. For example, companies that care about employee health, well-being, and retention will likely include the satisfaction and well-being dimension in their productivity measures. As a corollary, adding to or removing metrics can nudge behavior, because that also communicates what is important.

For example, a team where "productivity = lines of code" alone is very different from a team where "productivity = lines of code AND code review quality AND customer satisfaction." In this case, you have kept a (problematic, but probably embedded) metric about productivity and output, but nudged perceptions about productivity in a direction that also values both teamwork (by valuing thoughtful code reviews) and the end user (by valuing customer satisfaction).

Metrics shape behavior, so by adding and valuing just two metrics, you've helped shape a change in your team and organization. This is why it's so important to be sure to pull from multiple dimensions of the framework: it will lead to much better outcomes at both the team and system levels. In this example, as the teams continue to improve and iterate, they could exchange the activity metric *lines of code* for something like *number of commits*.

### What to Watch For

Having too many metrics may also lead to confusion and lower motivation; not all dimensions need to be included for the framework to be helpful. For example, if developers and teams are presented with an extensive list of metrics and improvement targets, meeting them may feel like an unattainable goal. With this in mind, note that a good measure of productivity consists of a handful of metrics across at least

three dimensions; these can prompt a holistic view, and they can be sufficient to evoke improvement.

Any measurement paradigm should be used carefully because no metric can ever be a perfect proxy. Some metrics are poor measures because they are noisy approximations (some examples are noted in Figure 1). Retention is often used to measure employee satisfaction; however, this can capture much more than satisfaction—it can reflect compensation, promotion opportunities, issues with a team, or even a partner's move. At the team level, some managers may block transfers to protect their own retention ratings. Even if retention did reflect satisfaction, it is a lagging measure, and teams don't see shifts until it is too late to do anything about it. We have written elsewhere about the limitations inherent in the use of story points,<sup>9</sup> which could give teams incentive to focus on their own work at the expense of collaborating on important projects.

Teams and organizations should be cognizant of developer privacy and report only anonymized, aggregate results at the team or group level. (In some countries, reporting on individual productivity isn't legal.) Individual-level productivity analysis, however, may be insightful for developers. For example, previous research shows that typical developer work shifts depend on the phase of development, and developers may have more productive times of day.<sup>14</sup> Developers can opt in to these types of analyses, gaining valuable insights to optimize their days and manage their energy.

Finally, any measurement paradigm should check for biases and norms. These are external influences that may shift or influence the measures. Some examples are included here, but they aren't exhaustive, so all teams are encouraged to look for and think about external influences that may be present in their data:

- *Peer review and gender.* Research shows that women are more likely to receive negative comments and less likely to receive positive comments in their code reviews.<sup>16</sup> Any analysis of satisfaction with the review process should check for this in your environment. Understand that developers are likely influenced by the broader tech

industry even if the patterns are not in your organization or team. Take these effects into account.

► *Normalizing measures across time.* Teams should be careful about any methods used to normalize time, especially across long periods. For example, looking at metrics over a year would bias against those taking parental leave.

► *Perceptual measures.* Teams and organizations should be mindful of cultural norms—and embrace these. Some cultures naturally report higher, while some report lower. It doesn't mean perceptual measures can't be trusted; it just means measures from these different cultures will have a different baseline and shouldn't be compared with each other.

## Why This Matters Now

Developer productivity is about more than an individual's activity levels or the efficiency of the engineering systems relied on to ship software, and it cannot be measured by a single metric or dimension. We developed the SPACE framework to capture different dimensions of productivity, because without it, pervasive and potentially harmful myths about productivity may persist.

The SPACE framework provides a way to logically and systematically think about productivity in a much bigger space and to carefully choose balanced metrics linked to goals—and how they may be limited if used alone or in the wrong context. The framework helps illuminate trade-offs that may not be immediately obvious and to account for invisible work and knock-on effects of changes such as increased work if activity is measured at the expense of unfulfilled developers or disruptions to overall flow and efficiency.

The need to understand and measure productivity holistically has never been greater. As the Covid-19 pandemic disrupted work and brought a sudden switch to working from home, many questioned its impact on productivity and posed questions around how to understand and measure this change. As the world slowly returns to a “new normal,” the SPACE framework captures the dimensions of productivity that are important to consider as future changes are proposed and made. The framework is meant to help

individuals, teams, and organizations identify pertinent metrics that present a holistic picture of productivity; this will lead to more thoughtful discussions about productivity and to the design of more impactful solutions.

**Acknowledgments.** We are grateful for the thoughtful review and insightful comments from our reviewers and are confident that incorporating their notes and responses has strengthened the article. **C**

## Related articles on queue.acm.org

### Getting What You Measure

Eric Bouwers, Joost Visser, and Arie van Deursen  
<https://queue.acm.org/detail.cfm?id=2229115>

### DevOps Metrics

Nicole Forsgren and Mik Kersten  
<https://queue.acm.org/detail.cfm?id=3182626>

### Beyond the “Fix-it” Treadmill

J. Paul Reed  
<https://queue.acm.org/detail.cfm?id=3380780>

### People and Process

James Champy  
<https://queue.acm.org/detail.cfm?id=1122687>

## References

1. Beller, M., Orgovan, V., Buja, S., Zimmermann, T. Mind the gap: on the relationship between automatically measured and self-reported productivity. *IEEE Software* (2020); <https://arxiv.org/abs/2012.07428>.
2. Brumby, D. P., Janssen, C. P., Mark, G. How do interruptions affect productivity? *Rethinking Productivity in Software Engineering*. C. Sadowski and T. Zimmermann, eds. Apress, Berkeley, CA, 2019, 85–107; [https://link.springer.com/chapter/10.1007/978-1-4842-4221-6\\_9](https://link.springer.com/chapter/10.1007/978-1-4842-4221-6_9).
3. Butler, J.L., Jaffe, S. Challenges and gratitude: a diary study of software engineers working from home during Covid-19 pandemic (Aug. 2020). Microsoft; <https://www.microsoft.com/en-us/research/publication/challenges-and-gratitude-a-diary-study-of-software-engineers-working-from-home-during-covid-19-pandemic/>.
4. Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper Perennial Modern Classics, 2008.
5. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conf. Computer-supported Cooperative Work* (Feb. 2012), 1277–1286; <https://dl.acm.org/doi/10.1145/2145204.2145396>.
6. Dourish, P., Bellotti, V. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conf. Computer-supported Cooperative Work* (Dec. 1992), 107–114; <https://dl.acm.org/doi/10.1145/143457.143468>.
7. Ford, D. et al. A tale of two cities: software developers working from home during the Covid-19 pandemic, 2020; <https://arxiv.org/abs/2008.11147>.
8. Forsgren, N. Finding balance between work and play. *The 2020 State of the Octoverse*. GitHub; <https://octoverse.github.com/static/github-octoverse-2020-productivity-report.pdf>.
9. Forsgren, N., Humble, J. Kim, G. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018.
10. Forsgren, N., Kersten, M. DevOps metrics. *Commun. ACM* 61, 4 (2018), 44–48; <https://dl.acm.org/doi/10.1145/3159169>.
11. Fuks, H., Raposo, A., Gerosa, M. A., Pimental, M. The 3C collaboration model. *Encyclopedia*

- of E-Collaboration. Ned Kock, ed. IGI Global, 2008, 637–644. [https://www.researchgate.net/publication/292220266\\_The\\_3C\\_collaboration\\_model](https://www.researchgate.net/publication/292220266_The_3C_collaboration_model).
12. Gazioti, D., Fagerholm, F. Happiness and the productivity of software engineers. *Rethinking Productivity in Software Engineering*. C. Sadowski and T. Zimmermann, eds. Apress, Berkeley, CA, 2019, 109–124; [https://link.springer.com/chapter/10.1007/978-1-4842-4221-6\\_10](https://link.springer.com/chapter/10.1007/978-1-4842-4221-6_10).
  13. Maslach, C., Leiter, M.P. Early predictors of job burnout and engagement. *J. Applied Psychology* 93, 3 (2008), 498–512; <https://doi.apa.org/doiLanding?doi=10.1037%2F0021-9010.93.3.498>.
  14. Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., Fritz, T. The work life of developers: activities, switches and perceived productivity. *IEEE Trans. Software Engineering* 43, 12 (2017), 1178–1193; <https://dl.acm.org/doi/10.1109/TSE.2017.2656886>.
  15. Murphy-Hill, E. et al. What predicts software developers' productivity? *IEEE Trans. Software Engineering*, 2019; <https://ieeexplore.ieee.org/document/8643844/>.
  16. Paul, R., Bosu, A., Sultana, K.Z. Expressions of sentiments during code reviews: male vs. female. In *IEEE 26th Intern. Conf. Software Analysis, Evolution and Reengineering*, 2019, 26–37; <https://ieeexplore.ieee.org/document/8667987>.
  17. Ralph, P. et al. Pandemic programming: How Covid-19 affects software developers and how their organizations can help. *Empirical Software Engineering* 25, 6 (2020), 4927–4961; [https://www.researchgate.net/publication/344342621\\_Pandemic\\_programming\\_How\\_COVID-19\\_affects\\_software\\_developers\\_and\\_how\\_their\\_organizations\\_can\\_help](https://www.researchgate.net/publication/344342621_Pandemic_programming_How_COVID-19_affects_software_developers_and_how_their_organizations_can_help).
  18. Schmidt, K., Bannon, L. Taking CSCW seriously: Supporting articulation work. *Computer-Supported Cooperative Work* 1, 1 (1992), 7–40; <https://link.springer.com/article/10.1007/BF00752449>.
  19. Sedano, T., Ralph, P., Péraire, C. Software development waste. In *Proceedings of the 39th Inter. Conf. Software Engineering*, 2017, 130–140; <https://dl.acm.org/doi/10.1109/ICSE.2017.20>.
  20. Storey, M. A., Zimmermann, T., Bird, C., Czerwonka, J., Murphy, B., Kalliamvakou, E. Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Trans. Software Engineering*, 2019; <https://ieeexplore.ieee.org/document/8851296>.
  21. Suchman, L. Making work visible. *Commun. ACM* 38, 9 (Sept. 1995), 56–64; <https://dl.acm.org/doi/10.1145/223248.223263>.
  22. Vasilescu, B. et al., Filkov, V. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd Annual ACM Conf. Human Factors in Computing Systems* (Apr. 2015), 3789–3798; <https://dl.acm.org/doi/abs/10.1145/2702123.2702549>.

**Nicole Forsgren** is the VP of Research & Strategy at GitHub. She is an expert in DevOps and the author of the Shingo Publication Award-winning book *Accelerate: The Science of Lean Software and DevOps*. Her work on technical practices and development is used to guide organizational transformations around the world.

**Margaret-Anne Storey** is a professor of computer science at the University of Victoria and a Canada Research Chair in Human and Social Aspects of Software Engineering. She consults with Microsoft to improve developer productivity.

**Chandra Maddila** is a Senior Research Engineer at Microsoft Research. He developed tools and techniques that are used organization-wide at Microsoft.

**Thomas Zimmermann** is a Senior Principal Researcher at Microsoft Research. He is best known for his work on mining software repositories and data science in software engineering.

**Brian Houck** is a Principal Program Manager in the Azure Engineering Systems at Microsoft. His work focuses on improving developer productivity and satisfaction for engineers within the Azure organization.

**Jenna Butler** is an adjunct professor at Bellevue College in the Radiation Therapy department and is a senior software engineer at Microsoft. She is currently working with MSR's Productivity & Intelligence team to study alignment and decision making; working in the services; and the impact of remote work during this time.

Copyright held by authors/owners..