Measuring developer productivity with the DX Core 4



Measuring developer productivity with the DX Core 4

The DX Core 4 is a unified framework for measuring developer productivity that encapsulates DORA, SPACE, and DevEx.

Whether you're a C-level leader or a frontline manager, there is no denying that measuring developer productivity—whether to understand performance or guide improvement—is a daunting challenge.

As the authors of frameworks like DevEx and SPACE, we're constantly being asked by engineering leaders: "How do we effectively measure productivity?" Or, "Between DevEx, SPACE, and DORA, which one should we use?". The leaders we speak to have sometimes spent months or even years wading through dashboards, but still find themselves lost. Or, they've established an initial set of metrics, but are struggling to get alignment and traction.

To help simplify the landscape, we've developed a unified approach to measuring developer productivity called the DX Core 4. The DX Core 4 encapsulates DORA, SPACE, and DevEx and includes four dimensions: speed, effectiveness, quality, and business impact.

The DX Core 4 provides a focused set of metrics that work effectively at any sized organization, and can be augmented with additional metrics for specific goals.

Our approach has been implemented at over 300 tech, finance, retail, and pharmaceutical companies delivering successful results including the following:

- 3%-12% overall increase in engineering efficiency
- 14% increase in R&D time spent on feature development
- 15% improvement in employee engagement scores

The foundation for data-driven improvements

With access to productivity data and insights, leaders can address complex questions about their engineering organizations, such as the following:

- How can we innovate and deliver software faster?
- What are the top areas of friction that are slowing engineers down?
- Is developer time being spent on activities that drive the most value?

The DX Core 4 (Table 1) offers a practical way for organizations to get immediate, actionable insights into the above questions. The DX Core 4 dimensions and metrics are based on decades of research combined with proven methods from the world's leading technology companies.

Below are several features of the DX Core 4 that are critical to success:

Multi-dimensional. Multiple dimensions are needed to capture software development comprehensively because changes to one dimension such as speed may negatively affect others (e.g., quality or effectiveness). The DX Core 4 includes four counterbalanced dimensions with metrics that encompass DORA, SPACE, and DevEx.

Supports all levels of the organization. The DX Core 4 metrics are useful for discussion and decision making at all levels of the organization, from the boardroom down to frontline teams. This alignment ensures focus on clear goals, helping drive coordinated action and improvements.

Table 1: DX Core 4

	Speed	Effectiveness	Quality	Impact
Key metric	Diffs per engineer* (PRs or MRs) *Not at individual level	Developer Experience Index (DXI) Measure of key engineering performance drivers, developed by DX	Change failure rate	Percentage of time spent on new capabilities
Secondary metrics	Lead time Deployment frequency Perceived rate of delivery	Time to 10th PR Ease of delivery Regrettable attrition* *Only at organizational level	Failed deployment recovery time Perceived software quality Operational health and security metrics	Initiative progress and ROI Revenue per engineer* R&D as percentage of revenue* *Only at organizational level

Deployable in weeks, not months. Organizations can spend months or even years building metrics dashboards, while not realizing value from these efforts. The DX Core 4 leverages readily-available system-based metrics and self-reported data, minimizing time and cost to stand up. By using an automated platform like DX, organizations can fully deploy metrics in weeks.

Avoid fear and gamification. Speed and throughput metrics, when used in isolation, often incite fear and counterproductive behaviors from developers. The DX Core 4 equally weights speed and output metrics with the Developer Experience Index (DXI) and additional experience data, fostering balanced conversations about developer productivity.

One of the DX Core 4 key metrics, diffs per engineer, requires caution. We at DX – along with many leading industry voices – have written extensively on the dangers and pitfalls of engineering throughput metrics.

We have found, however, that diffs per FTE is a useful signal when utilized carefully. Many of the organizations we work with, and leading technology companies like Meta, Microsoft, and Uber, rely on this metric as a key input for understanding and improving productivity.

Organizations can effectively utilize diffs per FTE successfully under three preconditions: first, by counterbalancing with oppositional metrics like the Developer Experience Index. Second, by not setting targets or rewards tied to them. Last, by properly communicating and rolling out metrics in such a way that does not result in abuse.

Data collection and analysis

The DX Core 4 metrics are collected through several methods including system metrics, self-report, and experience sampling, as listed in Table 2 below.

Table 2: Measurement methods

Method	Benefits	Challenges
System metrics	Objective metrics in real-time	Cross-system visibility and data normalization
Self-reported metrics	Rapid data collection and experience metrics	Question design, participation rates
Experience sampling	Targeted in-the moment insights	Complexity of setup, time to collect data

Self-reported metrics provide fast and comprehensive measurements in areas where system metrics are unavailable or do not apply[1]. For example, self-reported metrics are critical for perceptual measures of developer experience, as well as useful for collecting data about software quality that is difficult to measure objectively.

System metrics provide precise and continuous data, making them the preferred form of measurement where feasible. System metrics work well for capturing metrics such as diffs per engineer, where data can be easily extracted. In other cases, however, getting end-to-end system data can be difficult, requiring instrumentation and normalization of data across disparate tools and teams.

Failed deployment recovery time, for example, is a metric that we recommend collecting either through self-report or systems, depending on the organization. A small startup may be able to quickly measure using just an issue tracker such as Jira, whereas a larger organization will likely need to cross-attribute data across systems in order to gain end-to-end system visibility. This can be a lengthy effort, whereas capturing self-reported data can provide a baseline quickly.

Experience sampling is a method of collecting self-reported data from developers while they are in the flow of work. This provides targeted data points that can be tied to specific behaviors or tasks.

For example, experience sampling is a highly effective way of measuring concrete time savings being achieved through tools like Copilot.

By leveraging data collected through all three of these methods, organizations can gain the fullest picture of developer productivity. While it is important to start as early as possible to get the benefits of system data, organizations should establish baselines with self-reported data, without stalling while efforts to build out system-based metrics are underway.

Getting started

Companies are feeling increased pressure to measure and improve developer productivity as markets are putting greater emphasis on efficient growth and ROI

At the same time, the industry has gone through a paradigm shift in the last few years. Remote and hybrid work have become the norm, and the rapid advances of AI-enabled tools are beginning to transform the way software is developed. Leaders are left with big questions to answer about how to take the right actions to navigate these changes and remain competitive in the market.

[1] Noda, A., & Cochran, T. (2023, July 11). Measuring Developer Productivity via Humans. MartinFowler.com

As critical as these questions are, companies shouldn't feel they have to embark on a dramatic overhaul in order to get data-informed answers. The DX Core 4 metrics are designed to help you get started quickly by leveraging existing data and avoiding the need for expensive or time-intensive custom systems.

Establish baselines now. It's important to get started with baseline collection immediately, and for many organizations that will mean collecting self-reported data while efforts to capture and correlate system data are underway. In the absence of complete system measurements, self-reported metrics can provide a comprehensive view of engineering organization relatively quickly (i.e., within several weeks).

Start small. Look at the data to spot common issues and areas for improvement. Identify what changes would make the most significant impact, trying to keep the footprint of these initiatives as small as possible to avoid getting lost in the data.

Communicate transparently to your teams and leaders. The DX Core 4 metrics are as relevant to individual development teams as they are to the business. Create a plan to communicate how these metrics are collected and how they will be used with all members of your organization.

More than ever, having an efficient and effective engineering organization is critical to winning in the market. The DX Core 4 provides a proven path toward understanding and optimizing productivity to drive higher impact per developer.

The DX Core 4 has been deployed successfully at hundreds of organizations across tech, financial services, consumer goods, and pharma. To learn more about the DX Core 4 and DX platform, please get in touch with a representative from DX.

About the authors

This article is a collaborative effort by Abi Noda, Laura Tacho, Dr. Margaret-Anne Storey, Dr. Michaela Greiler, and Dr. Nicole Forsgren representing views from DX's research and consulting practices.

The authors wish to thank Thomas Zimmermann, Will Larson, Tim Cochran, Max Kanat-Alexander, Gergely Orosz, Kent Beck, and Pat Kua for their contributions to this article.