Editor: **Ciera Jaspan**
Google
ciera@google.com

Editor: **Collin Green**
Google
colling@google.com

Developer Productivity for Humans, Part 6: Measuring Flow, Focus, and Friction for Developers

Adam Brown, Alison Chang, Ben Holtz, and Sarah D'Angelo

IN A FIELD where behavior is often measured through technology, it's important to remember that behind the signals we use to create developer productivity metrics, there are humans at work. Otherwise, it can be easy (and often tempting) to lean into the most available (but least human) signals when measuring developer behavior. For example, number of builds is an easily accessible metric that on some level reflects work being done, but it fails to capture how the work is being done. Was the developer rapidly iterating through a flow state? Or were they stuck on some frustrating problem, encountering friction at every step? These aspects of the experience contribute to the larger story of developer productivity. While we are sadly still unable to read developers' minds and must rely on logs-based signals, our human-centered approach in developing these metrics enables us to contextualize the data and amplify the voice of the developer.

In this installment of the "Developer Productivity for Humans" column, we present two lines of research that take this approach to emphasizing the human experience in measuring developer productivity, specifically:

- developer experience of flow or focus
- developer experience of friction.

In each case, these experiences do not represent productivity on their own, but they are important inputs to eventual productivity,[1,2] as developers are happier and more productive when they are able to complete their work without experiencing friction and/or while frequently achieving flow. Unlike other measures of productivity, like lines of code or rounds of review, the flow and friction metrics we defined are grounded in human judgment rather than assumptions made based on certain tools being used or actions being taken.

For example, when thinking about friction, the first instinct may be to immediately describe any number of possible speed bumps (slow builds, flaky tests, etc.) as *friction*. However, slow builds may not always feel like friction, and fast builds might not always help productivity[3]; a slow build might be the proverbial tree falling in an empty forest: if the developer doesn't notice the delay, then it may not constitute friction. People have expectations about their environments and experiences that build up over time; developers get used to how often their tests fail and have expectations about how long a build might take (even if these expectations aren't always accurate), and

Digital Object Identifier 10.1109/MS.2023.3305718
Date of current version: 1 December 2023

 0740-7459/23©2023IEEE

for better or worse, they often accept some degree of slowdown as normal. Consequently, while a single slow build is easy to detect quantitatively, hastily labeling it as friction runs the risk of crying wolf and claiming friction without considering the developer's judgment or experience.[3]

Similarly, when thinking about flow and focus, a naive initial approach that only considers tool usage may assume that moving from an integrated development environment (IDE), to documentation, to the code repository, and then back to the IDE represents several context switches, which means that the pattern of behavior can't possibly demonstrate focused work. But if we zoom out and think about the developer's motivations behind these actions, these tools may all support a single goal (e.g., developing a new piece of code) and could be focused after all. The question is then how to identify actions across tools as falling within the same task or not. While biases around what "being productive" looks like might lead us to assume that any IDE usage at all counts as focused work and the presence of any chat messages means lack of focus, the broader context is again important here, as chat could be a means to an end to unblock other work and iterative debugging in an IDE debugging could be slow and frustrating.

In the following sections, we will describe the similar approaches we took to understanding and measuring the human experience of flow and friction. While it ultimately comes down to leveraging logs-based metrics and not directly accessing developers' thoughts and feelings, we do take a generalizable human-centered approach that emphasizes the developer's point of view by: 1) understanding developers' subjective experiences, 2) identifying logs-based signals that most closely represent these experiences, and 3) validating our metrics against self-reported data. We hope our approach and these examples can be used to promote a more holistic look at developer experience that contains the human in the loop.

Measuring Flow and Focus

Achieving a state of flow has been defined as the *optimal experience*[4] and is often linked to feeling productive, focused, and accomplishing goals. However, as mentioned before, flow is a personal experience and for many years has been difficult to measure in nonintrusive ways.[1] As part of our team's effort to develop a holistic perspective on developer productivity, we sought to develop a metric that measures when developers experience flow in their daily work, starting with a human-centered, qualitative approach.

We conducted a diary study with follow-up interviews to hear from developers directly and identify generalizable characteristics of flow that informed our logs-based metric. This phase revealed three primary themes that shaped our definition:

- developers would describe their experience as flow only if they felt positively about the work they were completing
- developers described experiencing flow across a variety of tasks, not just writing code, but also when responding to emails, drafting design docs, and reading documentation
- once established, flow can withstand small distractions.

The first insight presented one of our main challenges in this effort. How can we infer human sentiment from logs-based metrics? Well, we can't. But at the end of the day, our high-level goal is to understand developer productivity, and we decided that having a measure that could potentially capture both focus and flow, while not being about to tease the two apart, was far better than abandoning this work altogether. This drove us to consider both flow and focused work, with the view that humans achieve flow states if and only if they are doing focused work, but that they can do focused work without achieving flow. The second and third insights introduced human elements of the experience of flow and focus that we could incorporate into our work, as there is more to flow for developers than staying immersed writing code in a single tool for long periods of time.

Defining the Focus Time Metric

Our qualitative insights formed the foundation for the next phase, identifying a logs-based signal that was agnostic to task, robust to small distractions, flexible on duration, and independent of the perceived outcome of the work. We wanted the metric to cast a wide net and reflect time spent engaging in focused work, which helped us further understand flow without assigning a value judgment to the task at hand (i.e., we do not need access to an individual's internal states). Figure 1 shows our conceptual model for how flow and focused work are related to our metric, which we call *focus time*.

We hypothesized that a proxy for focus could look at task similarity: performing a number of related actions in a given window of time indicates flow or focus, whereas performing a number of unrelated actions indicates a lack of flow or focus. This approach also accounted

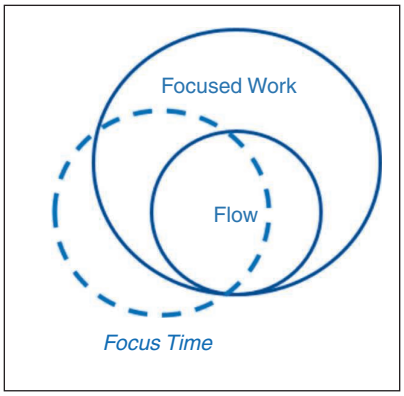

FIGURE 1. Diagram representing the hypothesized relationship among flow, focus, and focus time, acknowledging space for error in which we capture some work that is neither.

for our finding that flow is task-independent and tool-independent. In addition, it aligned with our learning that flow is robust to small interruptions, especially if the interruptions occurred in related tools.

We derived task similarity using a natural language processing technique called *word2vec*.[5] Just as the relatedness of words can be derived by comparing their occurrences in written sentences, we used sequences of logs generated by a range of commonly used tools to model sequences of tasks.[6] We were able to use the output of this model to group the time developers spent into periods where we thought they were experiencing focus and periods where they were not.

Validating the Focus Time Metric

Having a metric that labels periods of time as being focused or not was one thing, but having an individual agree with these values was another. To ensure that our metric accurately reflected the human experience, we validated that our metric was capturing the behaviors of interest using diary data and quarterly survey data. Following our standard diary data collection process,[6] we asked developers to record tasks as they completed them throughout the workday, and to aid this particular project we also had them indicate whether they felt that the task they completed felt "in flow or focused." We found high agreement between our metric and the diary data. We also found our metric to be a positive correlate to the survey item: "How often are you able to reach a high level of focus or achieve 'flow' during development tasks?" The correlation held even when controlling for other measures of developer activity (e.g., the total number of logged events and sessions). Together, these findings suggested that our metric was capturing focus and flow, both in the moment (as seen in diary data) and across longer periods of time (as seen using data from a quarterly longitudinal survey).

Despite pivoting from measuring flow alone to considering both flow and focus, our metric development experience reaffirmed the value of taking a human-centered approach. If we had started with the logs-based data available to us, we might have considered tool switching, interruptions, and duration to be more critical, similar to prior work in this area. Our multiphased approach enabled us to understand that developers have the ability to withstand minor disruptions, contextualize their work within tooling suites, and experience flow at a variety of durations. This gave us the criteria needed to develop and validate a logs-based metric that accurately reflects when developers experience flow or focused work.

Measuring Friction

Our interest in measuring developer friction stemmed from our desire to better understand the conditions that lead to productive and happy developers. We sought to design a metric that could provide high-level descriptions of friction across groups of developers (think something like "50% of developers experienced friction last week"), while maintaining enough granularity to point out areas for potential improvement. To achieve this second goal, we opted to model friction as a composite of simpler components, each of which potentially detracts from the developer experience. For example, if we found that slow build times were a key driver of friction for developers, we could increase resources to reduce build times. This kind of thinking led us to develop a friction metric that had the following characteristics:

- The metric contains a number of components that span key behaviors within the core development workflow.
- Each component is aggregated to the developer within a time period (e.g., average build latency per day for each developer).
- Each aggregated component value gets compared to a threshold value to label whether the developer experiences friction within that component. Thresholds are defined through research and analysis centered around developer perception, rather than distributional properties of the data (e.g., we don't just use the 90th percentile).
- If a developer has friction in any component, then we say that they experienced friction on that day.

In the spirit of transparency, we didn't start at zero. Teams at Google have been putting together friction

metrics for quite some time; some use these metrics to understand the benefit their infrastructure or tools provide, while others use these metrics to better understand what impedes the progress of their developers. These different goals have led to fairly different metrics, but all attempt to capture the same thing: hindrances to productivity.

For example, teams that are interested in the impact their tools have on friction have developed metrics that look at counts or percentage of "bad" events across a large collection of events (e.g., the number of flaky tests across all of Google). These teams are invested in lowering these values overall, so these metrics make sense for their use case. However, it isn't clear how developers are represented here: top-level metrics can rise or fall precipitously and the impact on developers is not immediately apparent (e.g., even if the number of flakes increased, did this lead to friction for many developers or just a few on select teams?). We suggest that it is critical to aggregate metrics to the developer, not merely count "bad" events.

We got the sense that metrics that don't center on the developer were capturing something about productivity, but maybe not developer productivity. It was challenging to tell a comprehensive story by making apples-to-apples comparisons between these metrics that were about artifacts (like change lists or test behavior) and other metrics about developers. If we cared about the experience of the developer, then why weren't we making things about the developer instead of individual interactions with various tools? With this in mind, we set out to define a developer friction metric that captures when an individual encounters issues while making code changes, releasing code, or debugging. Surely friction occurs in other phases of development (red tape, anyone?), but we started with these pieces of the development workflow and plan to expand it to cover other aspects of development over time.

Letting Developers Define Friction

In the first phase of developing this metric, we let developers tell us where they experienced friction. We didn't guide them toward specifically talking about issues with flaky tests or which tool blocked their work. Instead, we let developers define friction for themselves as anything that slowed down their progress. Similar to our investigation of flow and focus, this was an opportunity for us to cast a wide net in terms of things we might consider as components of friction, as well as to build an understanding of the frequency and form of the friction developers encounter.

To do this, we surveyed a sample of developers at the end of their workday each day for a week to better understand whether they experienced friction, what they were working on when this friction occurred, and how they resolved the friction. We also leveraged the same diary technique described earlier, but for this work we had developers indicate if they experienced friction while completing the task.

We found that friction was quite common in our sample of developers; during the survey period, developers reported friction on 77% of their days. When asked how they were able to avoid friction on a given day, one participant shared, "I think I got lucky." We also found that the components that tended to map onto the issues described by our sample were associated with build and test latency, flaky tests, and issues with code changes being blocked due to continuous integration failures. That is, the sources of friction that our participants reported were largely the same as had been assumed (by our team and others), but hearing from developers directly enabled us to improve upon categorizing these signals as *friction* or *not friction*, as well as better aggregation strategies. The results enabled us to lean into these components with more confidence, knowing they hold actual meaning to developers themselves, while still being relevant to infrastructure teams.

During this phase, we looked for relationships between the reports of friction from developers in our sample and existing friction metrics (average build latency, number of flaky tests). We considered these metrics as potential components of our eventual metric. Often, we found negligible or small relationships between these components and actual reports of friction. To some extent, these weak correlations made sense considering the mismatch that we've presented throughout this column: these metrics are typically about some artifact a developer interacts with and not the developer themself. However, we saw that if we aggregated these values directly to the developer, we could increase the agreement between these metrics and reports of friction. This provided some evidence that these components were likely good signals to use for measuring developer friction, but their aggregation and thresholding required additional tuning.

We conducted a number of follow-up interviews with developers about their diaries and their perspectives on friction. Interestingly, we found that the measurements often associated with friction were thought of as

"part of the job" and did not always immediately register as an issue. These developers did admit that at a certain point (e.g., after too many flaky tests) they would consider these experiences as friction. Furthermore, although these experiences may not have immediately registered as friction, they were still hypothesized to actually lead to lower productivity (i.e., a single flaky test likely does slow down progress toward submitting a code change some amount), which means the developer was still impacted. Due to this, the remainder of our research for developing this metric focused on combining attitudinal and behavioral signals to derive a metric for friction that tracked both how developers told us they felt and the metrics we use to measure productivity.

Balancing Self-Report and Logs Data

In an effort to strike a balance between developer reports of friction and logs-based indicators of friction, we identified components where higher values were negatively related to two sources of information:

- developer sentiment items from our quarterly survey that we hypothesized are related to friction (e.g., lower ratings of satisfaction with code complexity or project velocity)
- productivity metrics (e.g., fewer change lists, longer iteration loops).

These components included the latencies of local builds and tests, the latencies of testing that are associated with submitting change lists, and issues with flaky tests and blocked submission attempts. Critically, these values were aggregated to the individual developer. We did not label a developer as experiencing friction if they had a single long build that exceeded some threshold; rather, we found the average build latency that a developer experienced on a single day and compared this value to a threshold. We believe that this subtle difference puts the behavior within a broader context of an individual's day: having one long build may be frustrating if it is the only build that is run that day, but if there are tens of builds that run extremely quickly, this one long build may not be something memorable. In this second case, unsurprisingly, our data suggested that it depends on how long that build actually takes. Using the average of these latencies allowed for outliers to meaningfully drag the summary statistic upward, which meant that a single long build might or might not cause a developer's value to cross the threshold, but it depended on the rest of the distribution.

The last step was to define thresholds for our components. We treated this as a classification problem aimed at identifying individual developers that showed reduced productivity as measured by our logs-based indicators and/or individual developers who reported dissatisfaction with tools and development at Google, dissatisfaction with development speed, or being hindered dealing with code complexity and infrastructure. We searched a space of thresholds until we found threshold values where we were most likely to identify developers that reported experiences consistent with the construct of friction, patterns of behavioral data that were consistent with the construct of friction, or both.

After conducting our multiphase approach, we constructed a measure of developer friction that was defined with developer input, aggregated to their experiences, and blended their sentiments with their actions. We believe that following a human-centered approach gets us closer to understanding the human experience of friction in software development and moves us away from just measuring when something undesirable happens in a given tool.

Flow and friction are fuzzy human constructs. We believe that prior attempts to measure these experiences in the context of software development often overrepresented the development tools (and the signals that they produce) at the expense of more human aspects. Our approach to building the focus and friction metrics put the developers' personal experience front and center, enabling us to build metrics that can look at interventions aimed at increasing focus or decreasing friction through the lens of the end impact on the developers themselves. For example, we can use the focus time metric to measure the impact of calendar management and company-wide interventions. Do no meeting weeks enable developers to experience more time in flow or focus? Can condensing meetings and supporting focus time blocks improve developer productivity? Similarly, examining developers' workflows using our friction metric can enable us to identify areas for improvement. What workflows contribute to the most friction? What tooling improvements in the past have reduced friction? We are still in the early stages of these investigations, but we are excited to see how our human-centered metrics can improve our understanding of

developer experience and identify opportunities for new metrics.

In our work, we took a multiphased and human-centered approach. First, we gathered data directly from developers, including interviews, surveys, and diaries, to get a better understanding of what flow, focus, and friction meant to them and how they experienced them throughout their workday. Then, we leveraged this foundation to identify how aspects of those human experiences manifest in available logs-based signals, rather than jumping directly to data that is most readily available. We were able to generate heuristics that allowed us to transform the signals emitted from development tools into metrics that were more meaningfully related to these experiences. Finally, we validated these metrics against additional self-report and logs-based data to verify that our measures continued to be related to the experiences we cared about. This approach afforded us two new measures that provide insight into how developers get work done and can provide additional context into what makes them happy and productive. We hope this discussion can motivate further exploration into other complex aspects of developer experience with a focus on the human experience.

References

1. A. Brown, S. D'Angelo, B. Holtz, C. Jaspan, and C. Green, "Using logs data to identify when software engineers experience flow or focused work," in *Proc. CHI Conf. Human Factors Comput. Syst.*, Apr. 2023, pp. 1–12, doi: 10.1145/3544548.3581562.
2. C. Jaspan and C. Green, "A human-centered approach to developer productivity," *IEEE Softw.*, vol. 40, no. 1, pp. 23–28, Jan./Feb. 2023, doi: 10.1109/MS.2022.3212165.
3. C. Jaspan and C. Green, "Developer productivity for humans, part 4: Build latency, predictability, and developer productivity," *IEEE Softw.*, vol. 40, no. 4, pp. 25–29, Jul./Aug. 2023, doi: 10.1109/MS.2023.3275268.
4. M. Csikszentmihalyi, "Toward a psychology of optimal experience," in *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*, Dordrecht, The Netherlands: Springer, 2014, pp. 209–226.
5. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.
6. C. Jaspan et al., "Enabling the study of software development behavior with cross-tool logs," *IEEE Softw.*, vol. 37, no. 6, pp. 44–51, Nov./Dec. 2020, doi: 10.1109/MS.2020.3014573.

ABOUT THE AUTHORS

ADAM BROWN is a quantitative user experience researcher on the Engineering Productivity Research team at Google, New York, NY 10011 USA. Contact him at adambrown@google.com.

ALISON CHANG is a software engineer on the Engineering Productivity Research team at Google, Mountain View, CA 94043 USA. Contact her at alisonchang@google.com.

BEN HOLTZ is a software engineer on the Engineering Productivity Research team at Google, Toronto, ON M5H 2G4, Canada. Contact him at benholtz@google.com.

SARAH D'ANGELO is a user experience researcher on the Engineering Productivity Research team at Google, Seattle, WA 98103 USA. Contact her at sdangelo@google.com.