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

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

# Developer Productivity for Humans, Part 5: Onboarding and Ramp-Up

Collin Green, Ciera Jaspan, Maggie Hodges, Lanting He, Demei Shen, and Nan Zhang

**IN OUR COLUMN** thus far, we've focused on understanding and measuring productivity in a human-centered manner.[1] Along the way, we have noted that the productivity of less senior and less tenured developers is, at least in some cases, sensitive to different pressures (or differentially sensitive to the same pressures) as that of their more senior and more tenured colleagues.[2] This finding is intuitive: developers that are earlier in their career are typically assigned different tasks, they have less variety of experience to draw upon when faced with technical or organizational obstacles, and they may be less familiar with relevant tools, infrastructure, languages, libraries, and processes when compared to their more experienced fellow engineers. But how does a developer go from a rookie to a veteran? What facilitates or hinders developer onboarding and ramp-up? How can one assess interventions aimed at speeding up or otherwise improving developer training and education so that new engineers are enabled to hit their productivity stride quicker and more easily?

How well software development organizations can onboard new engineers is critical to their productivity and success. Measuring the onboarding experience of newly hired engineers and tracking it over time enables leaders to assess the impact of developer onboarding resources, practices, and programs to drive improvements. Such tracking also reveals the relative difficulty of ramping up in different parts of the organization, within different development specialties, or on certain skills, to target interventions appropriately. Enhancing the new engineer experience has the potential to positively impact engineering productivity, satisfaction, hiring, and retention.

In this installment of our column, we describe some recent research on onboarding software developers, including some of the work that we've done with colleagues at Google to understand and measure developer onboarding and ramp-up at Google.

## Understanding Ramp-Up Experience

Many researchers have sought to understand engineers' "ramp-up journey," with a particular focus on what facilitates or hinders ramp-up.[3] A wide variety of factors can aid or hinder a new engineer's onboarding and ramp-up,[4] and numerous investigations have revealed that technical skills, social and organizational factors, background and professional experience, and onboarding and training processes may speed or slow the development of expertise in a new software engineering role.[3,5]

To understand the onboarding and ramp-up process of Google's engineers, we worked with an external research vendor to interview engineers across the tech industry to understand common and unique onboarding practices. Then later, we surveyed new engineers at Google to understand who they are, what they experienced during onboarding and ramp-up, and what helped or hindered their ramp-up.

Our research revealed a variety of insights, many of which are mundane in hindsight: they are about

*Digital Object Identifier 10.1109/MS.2023.3291158*
*Date of current version: 13 October 2023*

how humans learn new concepts, rather than anything specific to software development specifically. We found, for example, the need for tailored onboarding specific to each team's work and each individual's prior knowledge. We also found that the COVID-19 pandemic made it harder to ramp-up, and that starter projects needed to be selected to maximize learning potential.

> Enhancing the new engineer experience has the potential to positively impact engineering productivity, satisfaction, hiring, and retention.

Although companywide onboarding programs engaged nearly all new developers, many noted that they could use more training specific to their team's tools and processes; this concern showed up in interviews and was also one of the top-three challenges in survey responses. New developers reported that they felt better supported when they had a mentor who could provide context for their work, and when they had a manager who could guide them on a pathway to ramping up on the team. These insights are not new or unique to Google; Microsoft also did extensive research and found that having a mentor or "onboarding buddy" was useful for most new employees.[5]

Developers who joined the company with prior work experience also reported having unique needs that were not always addressed by companywide onboarding. These developers would benefit from a more tailored onboarding process that allows them to opt-out of sessions that cover common industry best practices. Experienced developers preferred to learn about tools, processes, and systems by diving into real tasks on their primary projects, rather than running through sample tutorials or contrived starter projects.

For all developers, open responses from the survey indicate that a major challenge was the selection of an appropriate starter project that would facilitate learning key skills. This also aligns with the research done at Microsoft[5] that highlighted the importance of thoughtfully chosen onboarding tasks, the benefit of tailoring those tasks to developer level, and three strategies for doing so.

Reviewing the research that we and others have conducted to understand the ramp-up experience for new developers is useful because it provides ideas about what is working well, what is not working well, and consequently, where we might act to improve things. But how might we determine whether the changes made to onboarding are having the desired effect? How might we estimate the time saved or the efficiency gained in ramp-up that results from improvements? To do this, we need to have a measurement that assesses how long it takes new employees to ramp-up.

## Measuring Ramp-Up Time

Measuring ramp-up time would enable leadership to assess the impact of developer onboarding resources, practices, and programs to drive improvements. It would also reveal the relative difficulty of ramping up in different parts of the organization, within different development specialties, or on certain skills; this would allow the company to target interventions appropriately. In short, measurement of ramp-up time is a tool to help optimize the onboarding and ramp-up experience.

It's important to differentiate the construction of quantitative measures of "ramp-up time" from a more general notion of ramp-up experience. If we wanted to use logs-based metrics to understand productivity holistically during onboarding, we would review multiple metrics that capture the different facets of productivity (e.g., speed, ease, and quality of engineering work) and allow for consideration of tradeoffs that may exist among those facets.[6,7,8] But that wasn't our goal in this work; our goal in creating ramp-up-time metrics was to facilitate the evaluation of different interventions (e.g., different orientation and training programs) and the impact of external events (e.g., a global pandemic) on ramp-up times. In short, our ramp-up-time metrics are intended to provide information about the context of productivity, not productivity per se.

## Constructing Ramp-Up-Time Metrics

The idea of building an objective measure of ramp-up is not new. There is previous work that measures ramp-up

speed using the first time to code change, frequency of code changes, and size of code changes.[4,9,10] (We use historically similar metrics at Google: our own onboarding training regularly uses "time to first code change" as a means to evaluate whether the new training program is a success.) In the previous studies on ramp-up, researchers calculated how long it took for new developers' activity metrics to be comparable to those of their already-ramped-up colleagues. That is, they used more tenured engineers' metrics to develop a benchmark against which new engineers' metrics were compared.

We took a different approach, seeking to avoid scenarios where developers might be individually assessed or compared to each other. Such scenarios might incentivize developers to rush their onboarding or deliberately "game" our metrics.[11,12] We also wanted the ability to control for intrinsic individual differences and differences in the nature, complexity, and requirements of different engineering roles. So, rather than use measurements of tenured developers as a benchmark to evaluate new developers, we validated our metrics against engineers' own perceptions of the extent to which they were ramped-up on a variety of key engineering tasks and normalized progress to each individual engineer, rather than to his or her colleagues. That is, we created ramp-up metrics based on the time required for new developers to reach their own stable "cruising speed."

First, we reviewed a set of candidate metrics that might be useful proxies for engineers' ramp-up speed. We built upon our developer logging system, InSession,[13] which ingests logs from multiple developer tools to build a picture of a developer's behavior during his or her workday. We initially evaluated a set of nine different metrics that could be useful in understanding ramp-up time for engineers. Preliminary analyses showed some of these metrics had a clear progressive improvement over time and also stabilized around the same time, which increased our confidence that these measures reflect, in general, developers' overall comfort in their new roles, i.e., their "ramp-up." Ultimately, we selected three metrics [active coding time per line of code (LOC), reviewed LOC, and submitted LOC] to further evaluate for their utility in capturing ramp-up.

We created individually normalized versions of the candidate metrics. We looked at each engineer's weekly aggregated metrics, and then normalized each week's value by converting it to a percentage of the stable value it eventually reached (the engineer's "cruising speed"). We then aggregated all new engineers who started within the same cohort and designated the ramp-up time for the cohort as the number of weeks required for the aggregate metric to be (approximately) 0%, different from the cruising velocity, with the requirement that it remain so for at least four consecutive weeks.

## Validating Metrics Against Experience

These logs-based metrics compare developers to themselves rather than some generic threshold, thereby controlling for effects that result from language, team, and individual differences. Although we knew they had the shape we might expect, we still needed to know the extent to which the metrics reflected new developers' larger holistic experience of ramping up.

To validate these metrics, we invited every developer who started at Google in a 40-week time period to complete a survey about his or her onboarding experience. We used a cross-sectional design to understand the progress of onboarding: developers' hire date was used to specify cohorts of developers that were at different time points (by week) in their ramp-up trajectory. We ended up with a sample of more than 3,000 developers, which was representative of the population of new engineers.

Critically, the survey asked respondents to review the following set of 17 common developer tasks performed by new engineers:

1. writing code
2. reviewing other people's code
3. running builds and tests
4. finding examples of application programming interface (API) use by others at Google
5. searching for and using developer documentation (not API examples in code)
6. working with dependent changelists (a chain of changelists)
7. investigating the cause for the code/product behaving in a way you did not expect prior to release
8. creating or maintaining unit tests
9. reviewing other people's software design/architecture
10. tracking bugs and product issues
11. writing documentation for other engineers (e.g., tool, API, and service)
12. tracking your work items/development tasks
13. triaging or prioritizing feature requests or bugs
14. documenting decisions and the rationale for those decisions

15. finding the right person or team to contact for relevant expertise
16. feeling proficient with the developer tools
17. needing minimal assistance from other engineers to complete work.

For each task, respondents answered the question, "Thinking about your experience during the past week, which statement best describes how you feel about each of the following tasks?"

- "I haven't done this task."
- "I expect to get much more efficient at performing this task."
- "I expect to get somewhat more efficient at performing this task."
- "I expect to get slightly more efficient at performing this task."
- "I already perform this task as efficiently as I expect to."

Similar to the logs-based metrics we examined, the survey data display a pattern where engineers reported ramping up quickly at first, with progress slowing down and gradually reaching a plateau. To validate our candidate ramp-up efficiency metrics, we calculated how closely the survey-based ramp-up curves for each engineering task correlated with our logs-based ramp-up curves.

There were significant negative correlations between normalized active coding time per LOC and 15 of the 17 key engineering tasks that were included in the survey; that is, developers were coding faster and rating themselves as more ramped-up on these tasks as they gained tenure. Skills related to coding and documentation had the strongest correlations with active coding time per LOC, while skills around project management, knowledge discovery, and code review had lower correlations (but were still correlated). This result indicates that normalized active coding time per LOC is a reasonable proxy metric for engineers' ramp-up on most of the key engineering tasks.

Although this is a great result, it's not one that's useful for most software organizations as the ability to measure active coding time per LOC is somewhat unique. However, we did find that a simpler metric also performed decently: normalized submitted LOC was positively correlated with six of the 17 engineering tasks. That is, as engineers self-reported being able to complete the task as efficiently as they thought they could, they were also submitting more LOC to the codebase. The six tasks it was correlated with were reviewing others peoples' code, creating or maintaining unit tests, tracking bugs and product issues, writing documentation for other engineers, tracking development tasks, and triaging or prioritizing feature requests or bugs.

In brief, the following two metrics were reasonable proxies for engineers' ramp-up in general:

1. normalized active coding time per LOC
2. normalized submitted LOC.

Given that both metrics correlate with engineers' self-reported ramp-up across multiple engineering tasks, it might seem prudent to rely solely on the stronger metric (normalized active coding time per LOC). There are several reasons we chose to retain normalized submitted LOC as a second metric for ramp-up time. First, it is convenient to have two different metrics for ramp-up time that can agree (or not) on the impact of events or interventions that affect onboarding. Second, the previous research examined both the rate and volume of development work as ways of measuring ramp-up, and the two metrics roughly correspond to these constructs. Finally, many organizations might find it inconvenient to calculate active coding time. Calculating normalized submitted LOC per engineer is likely to be simpler and faster for others seeking to adopt our approach.

But a word of warning before adopting these metrics: we've found these metrics to be highly erratic and noisy on an individual level. These metrics should not be used to evaluate individual ramp-up speed as they're only valid for evaluating the quality of a training program across a cohort of developers.

## Measuring the Impact of an Unplanned Change (COVID-19)

All of our data validation took place during the time period impacted by the COVID-19 pandemic and remote onboarding policies; all the engineers in our survey joined and onboarded in a fully remote setting. However, relevant logs data are available extending back several years, so this allows us to calculate ramp-up-time metrics for cohorts of engineers who were hired well before the pandemic. To demonstrate the sensitivity of these metrics to a substantive change in onboarding practices and ramp-up experience, and to address a timely and interesting problem, we examined the question of whether engineering ramp-up times were impacted by the shift to remote onboarding practices immediately following the beginning of the COVID-19 pandemic. We thus were able to evaluate whether

the work-from-home (WFH) policies and remote onboarding that started in mid-March 2020 impacted engineer ramp-up efficiency by comparing ramp-up times for engineers who joined Google in the three months after WFH (April–June 2020) to the corresponding months in 2018 and 2019.

Engineers who joined Google after the beginning of the pandemic were distinct from those who joined before the pandemic when we examined normalized active coding time per LOC. For engineers who joined prior to the pandemic, their eventual cruising speeds were 2.5-times faster than their initial values, but for engineers who joined after, their eventual cruising speed was only 1.5-times faster than their initial speeds. Engineers who joined after the pandemic also took longer to ramp-up: their ramp-up time was approximately 22 weeks compared to 19 weeks for engineers who joined before the pandemic, a difference of three weeks. We similarly saw a longer ramp-up time when measured using submitted LOC per week. New engineers who joined before the pandemic ramped-up to their stable rate of submitted LOC per week in roughly 12 weeks, six-weeks faster than new engineers who joined after the pandemic, who took 18 weeks to reach a stable rate. Combined, these results provide concrete indicators that it was harder for new engineers to ramp-up in the pandemic environment.

In summary, our ramp-up-time metrics were sensitive to the change in onboarding practices that coincided with the onset of COVID-19 and emergency WFH and suggest that remote onboarding (at least as implemented hastily in the early days of the pandemic) resulted in slower ramp-up for engineers on the order of three to six weeks. The slower pace of ramp-up during remote onboarding is corroborated by the respondents' answers to our open-ended survey question about challenges and support for onboarding, which described how WFH impacted onboarding and what might have improved their experience.

Combined, these results provide concrete indicators that it was harder for new engineers to ramp-up in the pandemic environment.

## Future Applications

We didn't set out to measure the impact of the pandemic, of course. It merely provided a large-scale change to onboarding at Google that allowed us to test out our ramp-up-time metrics. Our plans for employing our metrics and the findings from our other research are still forming, but we're thinking about applications in concert with our colleagues who run education and training for new Google developers.

External research (ours and others') and our own survey data indicated that the three top hindrances to ramping up were learning a new technology, poor or missing documentation, and finding expertise. A qualitative analysis of open responses regarding challenges during onboarding indicate that designing on-team starter projects to better facilitate learning key skills, offering a more structured curriculum, and offering differentiated training for teams, technical specialties, and levels beyond the basics covered in standard onboarding programs all present opportunities to improve the onboarding experience. The intersecting challenges related to ramping up remotely were the most common theme in open-ended responses, including barriers to ask questions, a need for much more live coding collaboration and mentoring, and a lack of opportunities for casual interaction, relationship building, and learning through observation.

There are a few "easy wins" that organizations can do right now to address common onboarding hindrances. New developers described barriers to asking questions, learning through observation, and building rapport with their team, particularly in remote or distributed team contexts. The teams that are onboarding developers should encourage their existing team members to spend time collaborating live with the new developers during their first months; this could include screen-sharing code or pair programming during video or in-person meetings as well as meeting face to face to discuss their questions and projects.

Longer term, we think that organizations should evolve onboarding to better address developer needs. New developers are not a homogenous group, and a one-size-fits-all approach to onboarding may be less

effective than a differentiated approach to onboarding. Tailoring developer onboarding to the level of the engineer, his or her years of previous experience, type of previous professional experience as a software developer, and development specialties may support their ramp-up. This could take several forms, such as

- a structured consideration of developers' previous experience (both amount and nature) and matching them with appropriate mentors. It's possible that they would benefit most from being connected with those who have navigated transitions similar to them (for example, from a small company to an enterprise company, or among certain development specialties), or that they would benefit most from particular pairings (for example, being mentored by a developer one level above their own).
- creating cohort groups for these varied previous experiences, or creating documentation-compiling tips and resources aimed at these groups.
- constructing a survey, chatbot, or 1:1 conversation guide to take previous experience and other factors into account and route developers to courses or programs accordingly.

New developers sometimes experience a lack of structure and a sense of being overwhelmed while onboarding. They also sometimes feel that the initial tasks they are assigned are not correctly sized given their goals and knowledge of the company or team practices. To address this, consider,

- streamlining developer onboarding documentation so that more of it lives in a centralized landing page, which is outlined and sequenced in such a way that developers feel empowered to browse what's available and make strategic choices about what to prioritize in the near term, what to return to in the future, and what is not relevant to their role. This resource should provide a sufficient summary of various topics so that it acts as a self-contained curriculum, rather than a series of links routing developers to other documentation that was not written with new developers in mind.
- developing, publishing, and disseminating more guidance for teams on how to design and implement an appropriately scoped and leveled starter project for new developers that provides them with opportunities to practice the breadth of key skills they're most interested in acquiring when joining a new team (for example, pushing their first change to production or orienting themselves with the architecture of their codebase).

> New developers are not a homogenous group, and a one-size-fits-all approach to onboarding may be less effective than a differentiated approach to onboarding.

Onboarding new developers effectively and efficiently is important, especially for growing organizations. Surveys and qualitative research provide useful information about what works and what doesn't and provides ideas for what to change about onboarding and ramp-up processes. Quantifying the impact of improvements isn't required but can inform decision making about how to augment or refine onboarding. We developed quantitative measures of ramp-up time at Google, but when we apply them we're careful to reinforce the idea that these are merely good proxies for ramp-up (i.e., they correlate with self-reported ramp-up on key engineering tasks) and should be complemented by other evidence that indicates education, training, mentorship, and onboarding are effective. In our view, this is a sensible, human-centered approach to building a better onboarding experience.

## References

1. 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.
2. C. Jaspan and C. Green, "Developer productivity for humans, part 2: Hybrid productivity," *IEEE Softw.*, vol. 40, no. 2, pp. 13–18, Mar./Apr. 2023, doi: 10.1109/MS.2022.3229418.

## ABOUT THE AUTHORS

**COLLIN GREEN** is the user experience research lead for the Engineering Productivity Research team at Google, Mountain View, CA 94043 USA. Contact him at https://research.google/people/107023/ or colling@google.com.

**LANTING HE** is a software engineer for the Engineering Productivity Research team at Google, New York, NY 10011 USA. Contact her at lantinghe@google.com.

**CIERA JASPAN** is the software engineering lead for the Engineering Productivity Research team at Google, Mountain View, CA 94043 USA. Contact her at https://research.google/people/CieraJaspan/ or ciera@google.com.

**DEMEI SHEN** is a senior data scientist and product analyst at Google, Kirkland, WA 98033. Contact her at demei@google.com.

**MAGGIE HODGES** is a user experience researcher on the Engineering Productivity Research team at Google, Mountain View, CA 94043 USA. Contact her at https://research.google/people/108095/ or hodgesm@google.com.

**NAN ZHANG** is the data scientist lead for Google developer productivity at Google, Mountain View, CA 94043 USA. Contact her at nanzh@google.com

3. A. Begel and B. Simon, "Novice software developers, all over again," in *Proc. 4th Int. Workshop Comput. Educ. Res.*, 2008, pp. 3–14, doi: 10.1145/1404520.1404522.
4. A. Rastogi et al., "Ramp-up journey of new hires: Tug of war of aids and impediments," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Oct. 2015, pp. 1–10, doi: 10.1109/ESEM.2015.7321212.
5. A. Ju, H. Sajnani, S. Kelly, and K. Herzig, "A case study of onboarding in software teams: Tasks and strategies," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, 2021, pp. 613–623, doi: 10.1109/ICSE43902.2021.00063.
6. E. Murphy-Hill et al., "What predicts software developers' productivity?" *IEEE Trans. Softw. Eng.*, vol. 47, no. 3, pp. 582–594, Mar. 2021, doi: 10.1109/TSE.2019.2900308.
7. C. Jaspan and C. Sadowski, "No single metric captures productivity," in *Rethinking Productivity in Software Engineering*, C. Sadowski and T. Zimmermann, Eds., Berkeley, CA, USA: Apress, 2019, pp. 13–20, doi: 10.1007/978-1-4842-4221-6_2.
8. N. Forsgren et al., "The SPACE of developer productivity: There's more to it than you think," *ACM Queue*, vol. 19, no. 1, pp. 20–48, Feb. 2021, doi: 10.1145/3454122.3454124.
9. A. Mockus, "Succession: Measuring transfer of code and developer productivity," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, Vancouver, BC, Canada, 2009, pp. 67–77, doi: 10.1109/ICSE.2009.5070509.
10. A. Rastogi et al., "Ramp-up journey of new hires: Do strategic practices of software companies influence productivity?" in *Proc. 10th Innov. Softw. Eng. Conf.*, 2017, pp. 107–111, doi: 10.1145/3021460.3021471.
11. "Campbell's law." Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Campbell%27s_law
12. "Goodhart's law." Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Goodhart%27s_law
13. 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.