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

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

Measuring Developer Goals

Benjamin Ferrari-Church and Carolyn Egelman

UNDERSTANDING AND EFFECTIVELY measuring developer goals is critical for enhancing developer experience and productivity. By focusing on durable, consistent, relatable, sensical, and observable goals, we create a more robust view into our developers' days.

Introduction

In this installment of our column, we're writing about the value of goal-based measurement of the developer experience. Developer tooling teams are often faced with questions like: "How frequently do developers have to debug a failing test?" "How long does setting up a new server take a developer?" "How many developers are using a given tool in their workflow to test code quality?" "Are we actually improving the ways developers' get their work done?" among others. Anchoring measurement around user goals (in this case, developer goals) helps to answer these questions by keeping metrics human-centered,[1] and encouraging measurement thinking that spans product boundaries: Many developers have to work with multiple tools to complete their development goals. In this article, we'll outline our process for articulating and refining goals, provide our list of 30 rigorously-tested developer goals, and share a little bit about how we leverage both sentiment and behavioral data to measure and understand goals through different lenses.

Value of Articulating and Measuring Developer Goals

Measuring user goals is not a new practice at Google. Since 2014, Google has relied on critical user journeys (CUJs) to drive product excellence in our billion user products.[2] A CUJ captures who the user is, their critical goal, and the journey of tasks the user undertakes to achieve that goal. A canonical example would be "As a <user>, I want to <goal>, so I <task 1>, <task 2>, and <task 3>". The value proposition of CUJs is fairly straightforward: By understanding, articulating, and measuring the way that users employ tools in tasks that accomplish their goals, you can make sure you focus on improving the highest value areas with the biggest opportunities. While goal-oriented measurement has a 10+ year history at Google, the practice of using this framework is relatively new in the developer experience space.

Historically, our measurement of CUJs relied on lab-based studies and task-based metrics to quantify the effectiveness and value of our products and infrastructure. For example, we might run a lab-based UX study where users are asked to "get directions to configure a new server" and determine how difficult it is for users to achieve the goal. While we could take a similar approach for developer journeys, we found that many developer journeys are looping and iterative and many span across multiple developer tools. Even our most straightforward tasks are made complex through their dependence on context (particularly, the intent or goal). For example, "searching for documentation" is a common and relatively simple task, but whether you are "exploring technical solutions" as part of system design or looking to "understand the context to complete a work item" changes everything about your journey and how well supported you feel while accomplishing it. This situation presented a challenge: To truly understand the value of our developer tools, we needed to shift away from measuring individual tasks and instead focus on understanding and measuring the overarching goals that drive

Digital Object Identifier 10.1109/MS.2024.3410830
Date of current version: 13 August 2024

0740-7459/24©2024IEEE

developer behavior and the context in which those goals are accomplished.

Goals That Span the Developer Experience

There are many journeys that developers take while attempting to complete their work, and we could define and measure all of them. However, we sought to also have a concise list, which are truly the "critical" developer journeys that we want to track and measure over time to ensure that our tools and infrastructure are helping developers achieve their goals. When we set out to define a collection of developer goals to track and measure, we set the following success criteria for well-defined individual goals and a useful collection:

- *Each goal needs to be durable:* We hope they will feel as relevant five years from now as they did five years ago. We can never anticipate what changes the world will throw at how we accomplish our work. We didn't anticipate the rise of work-from-home due to COVID[3], which changed how people work, although not their goals. Now our industry is seeing how artificial intelligence (AI) is changing developer workflows, and we need goals that will be durable to that new context too.
- *Each goal needs to make sense to our developers:* We knew we were going to measure these via a survey, so having goals that our developers could intuit and see as part of their workflows was critical.
- *Each goal needs to connect to observable developer behaviors:* We also knew that we were going to use logs to measure these goals, so being able to map them to specific developer actions was vital.
- *Collectively, goals need to be consistent in altitude or scope:* We tested our goals to ensure they feel similar in breadth even when they are distant in developer workflows.
- *Collectively, goals need to be comprehensive:* We anchored our goals to the software development lifecycle to ensure coverage.

Once we had articulated these criteria, we completed an extensive and iterative goal development process. First, we gathered subject matter experts to create a list of goals that spanned across each of the software development phases. We did this with a small group at first to ensure focus, keep the list within our target size of 30 goals, and to maintain a consistent altitude. Second, we mapped the draft list of goals to the historical data from the longitudinal, quarterly engineering satisfaction (EngSat) survey[4] related to a (much longer) list of development tasks. Comparing our draft goal list to the existing, extensive task list from the survey helped ensure we had sufficient coverage of the breadth of development tasks. Third, we went through a round of cross-functional feedback followed by two rounds of user research. The research was a moderated card-sort study with six internal Google developers, which enabled us to evaluate the clarity of the goals and the organization of those goals into phases. After the moderated card sort, we ran a larger (n = 40) unmoderated card-sort to build confidence in how the goals were organized into phases. Finally, we ended the development of our goal list with a final round of cross-functional feedback and user research in the form of a cognitive user testing during the EngSat launch process.[4]

> A CUJ captures who the user is, their critical goal, and the journey of tasks the user undertakes to achieve that goal.

Durable Developer Goals

Figure 1 shows the final list of developer goals that we created. The bold categories are the development phases, which comprise our view of the software development lifecycle and serve as the organizing principle for our goals.

We also needed to contextualize these goals within our existing ecosystem. Different teams across our organization had already identified their own developer tool-specific sets of users, goals, and tasks, and we needed a way to organize and communicate the relationship among our consistent end-to-end goals and the various developer tool-centric sets of goals that existed if we wanted to enable consistent measurement. Because of the altitude and cross-product nature of our goals, we stationed them as the highest point in the information architecture. Specific developer tool goals and their

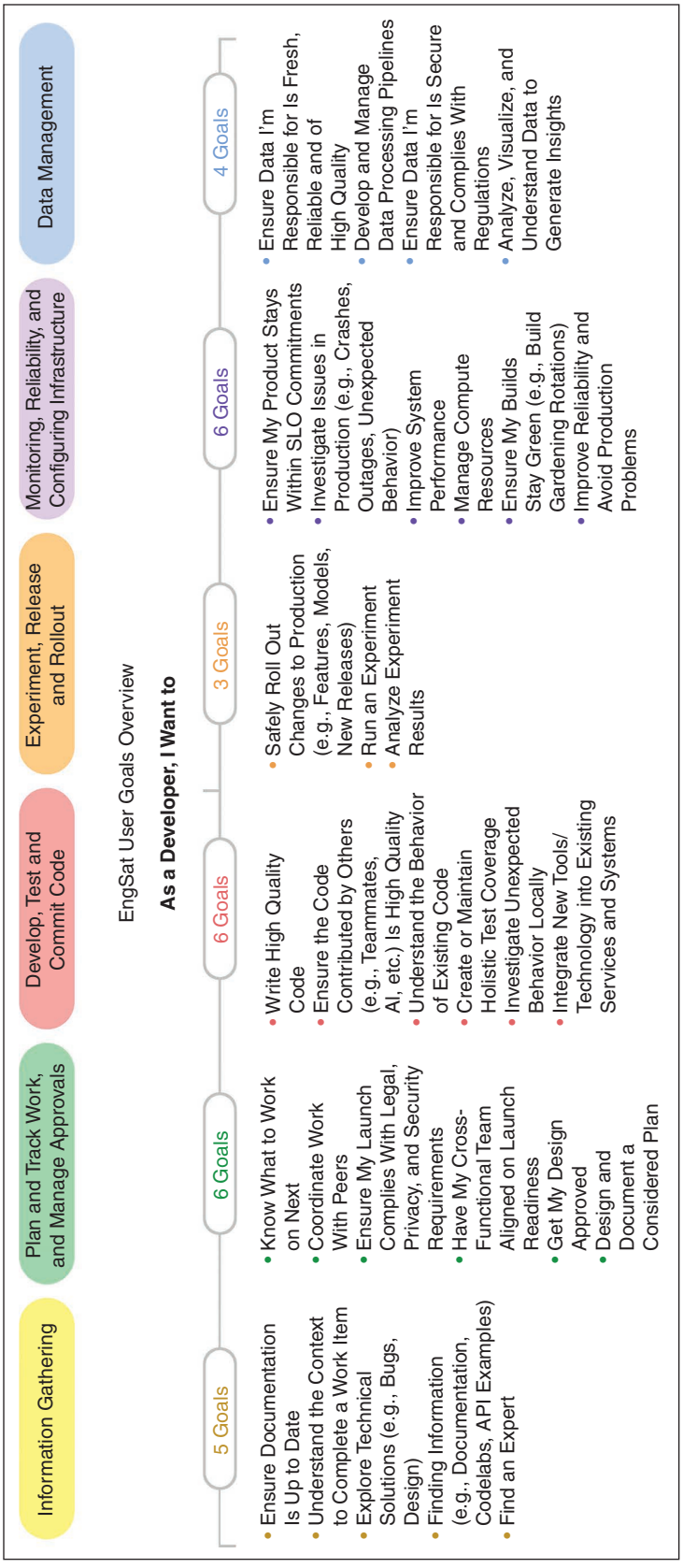

FIGURE 1. Phases of the software development lifecycle and associated developer goals for each phase.

component tasks can map to one or more of our higher-level developer goals. This enables us to provide a many-to-many mapping between developer tool goals and higher-order developer goals that they support.

Figure 2 shows the subset of our final list of developer goals associated with the Develop, Test, and Commit Code phase of the software development lifecycle. We've added the bolded categories of developer tool goals on the right to help explain the many-to-many relationship that can exist between our developer goals and individual developer tools

As mentioned earlier, it's important that our developer goals are durable and will stand the test of time and that they are written in a way that is technology agnostic. Although we want to be able to map our goals to developer tool-specific goals, we know that developer needs and goals are not likely to evolve as quickly as technology. For example, software development practices and tooling are at the beginning of significant transformations as Generative AI reshapes how developers do their work. That transformation will be very apparent in developer tool-specific goals, but will have much less impact on our technology-agnostic goals. By focusing on the latter, we are better positioned to track and understand the changing developer experience as we roll out these AI interventions across the workflows.

Measuring Sentiment and Behavior Surrounding Developer Goals

Survey-Based Attitudinal Measurement of Developer Goals

We measure attitudes toward support of these goals through our longitudinal, quarterly engineering

satisfaction (EngSat) survey.[4] We ask developers how well supported they feel for each of the 30 developer goals and provide one open-ended question where respondents can share any feedback on specific pain points or general dissatisfaction related to development goals.

Logs-Based Behavioral Measurement of Developer Goals

Our team maintains a cross-product logs-based measurement system[5] and spent the past few years extending this system to measure the journeys developers take in pursuit of accomplishing their goals. We have found benefits in the log-based system for developer tool teams beyond just the data outputs. We recognize building and maintaining such a system is not commonplace in most engineering workplaces; however, we want to share the lessons we have learned in working with these logs.

As mentioned above, and by design, our list of goals are mappable to observable logs-based measurement. That means that we have to be precise about what log points indicate the start and the end of a journey, as well as what interesting points might happen to differentiate one journey from another. By working with developer tool teams and translating a natural language version of a developer goal into the set of log points that uniquely identify that a developer is taking action toward that goal forces that developer tool team to be really concrete about what they mean a developer's goal is. For example, heading into measurement definition, one of our teams articulated the developer goal of "As a software engineer, I want to deploy my server into production"; however, after a workshop to translate this goal into the journey a single developer would take and map that to log points along the way, the goal was redefined to be "As a software engineer, I want to identify the necessary configuration settings to deploy my server into production." While the initial articulated goal was broader, it was unlikely to be something a single developer would do by themselves and the reartículated goal was a necessary initial step that a single developer would take in pursuit of the larger goal.

Second, we designed our system to be flexible about the order in which events occur and to provide configuration options that address the iterative nature of software development. Developer goals are often quite

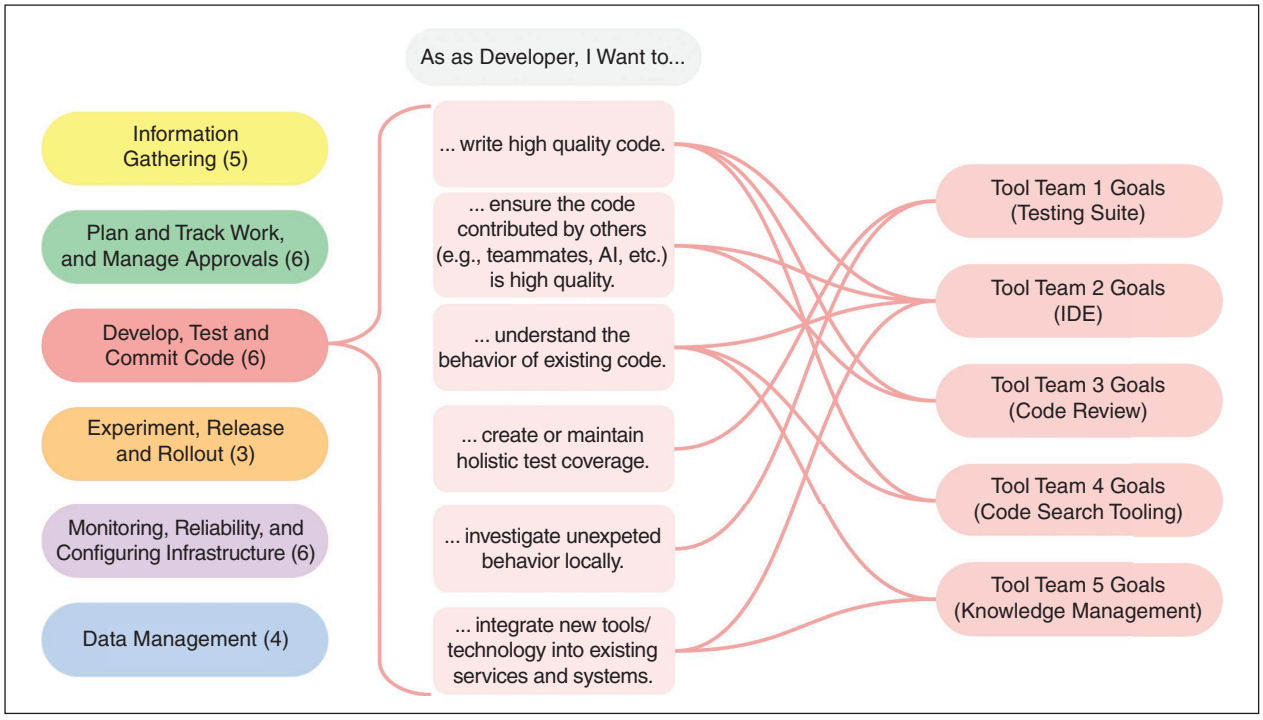

FIGURE 2. An example of how our higher-level developer goals (here, the subset associated with the Develop, Test, and Commit Code phase of the SDLC) maps to lower-level developer tool goals as articulated by developer tools teams. IDE: integrated developer environment; SDLC: software development lifecycle.

complicated and could be achieved in a variety of ways. Our developer tool teams have found value in being able to quantify how often developers take a "golden path" versus other less well-supported paths to complete their goals. If a system was designed to only capture those that take the "golden path," we would miss insights into why developers take those less-desired paths. Additionally, those configuration options let us do things like count how many iterations were required for someone to debug a failing test. In this case, just capturing the time between the most proximate failing test and a passing one would miss all of the work that went into understanding and debugging that test when it was still failing.

The balance of requiring specificity in the end points because you have to know what log points to measure, but allowing flexibility in how developers actually get from a start to an end, enables a rich understanding of how developers navigate development systems.

Deepen Understanding by Combining Attitudinal and Behavioral Data

By triangulating data from EngSat sentiment measurement and logs-based measurement in our Developer Journeys system, we can identify patterns of usage that are shared by our most satisfied or productive developers, enabling an even deeper level of insight into ways we can make their work days even more productive and satisfying or learn from how they navigate a journey to be able to make improvements for developers who feel less well supported.

For example, we might find that developers who express positive goal sentiment in EngSat also exhibit specific usage patterns, such as more frequent code review sessions or sessions with longer duration. Conversely, developers who report negative sentiment might show different behavioral patterns, such as more switches between tools or context switches. By identifying these correlations, we can pinpoint areas that would benefit from integration, improved onboarding, or further investigation. It is the combination of survey and logs data that enables us to make data-informed improvements that enhance the developer experience.

Ultimately, the value of combining sentiment and behavioral data lies in its ability to provide a more holistic and thorough understanding of developer goals. This approach lets us move beyond surface-level metrics and uncover deeper insights that can inform product development, improve developer satisfaction, and drive business success.

Getting Specific: Anchoring on Ensuring Quality as the Goal of Code Review

Let's see what this looks like in practice. Prior to switching to a developer journey-focused set of questions, we had asked engineers for six years about code review. The actual question on the survey was "How well do the developer tools you currently use at Google support you in the following developer tasks" with "code review" listed as one among 70 other tasks. In the evolution of this list from tasks to goals, this turned into the question "How well did the developer tools you currently use at Google support you in the following developer activities" with the option text being "Ensure the code contributed by others (e.g., teammates, AI, etc.) is high quality." This language change made the survey item much broader and refocused on the goal—ensuring code that gets

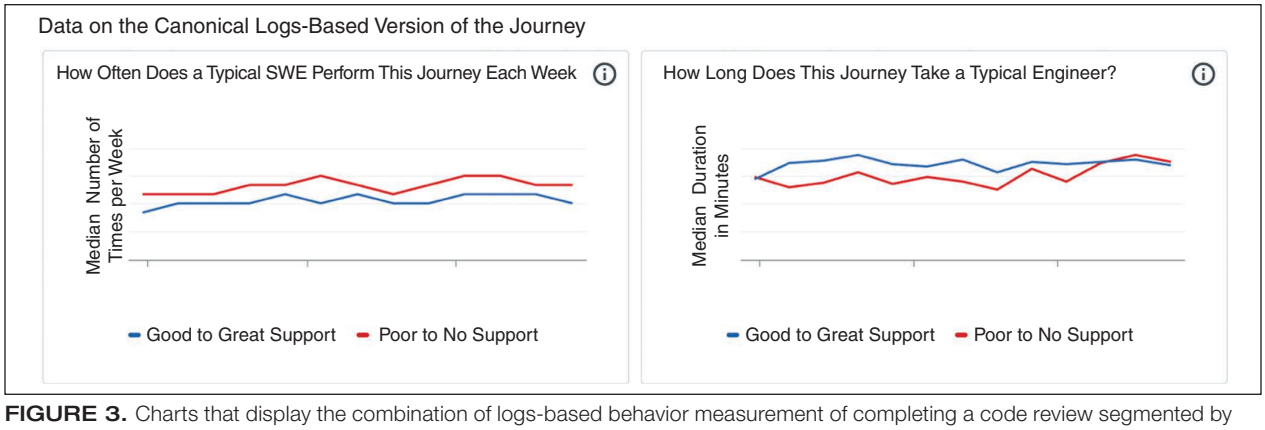

FIGURE 3. Charts that display the combination of logs-based behavior measurement of completing a code review segmented by survey-based attitudinal measures with support for completing that goal. SWE: software engineer.

submitted is of high quality—rather than the mechanics around the task of executing a code review. We saw survey scores shift downward modestly with the language change, opening up new questions for the developer tool team around how they can ensure our internal coding tools are best supporting developers in writing and reviewing high-quality code and opposed to just completing a "code review."

In addition to the insights from broadening language in the survey question, we have data on what completing a code review looks like from tooling logs. We can now bring these data sources together to see differences in the behavior of developers who feel well supported in completing code reviews versus those who feel less well supported. Figure 3 shows a snapshot of our internal dashboard displaying how often the typical developer completes the goal of "ensure the code written by others is high quality" and how long that takes for the most and least well-supported groups. In this case, we see that developers who feel less supported actually review code more often and do so in less time than developers who feel well supported. These data prompted new studies around understanding the review loads and practices of these engineers who don't feel well supported; with the aim of providing new features to support these engineers.

Understanding and effectively measuring developer goals is critical for enhancing developer experience and productivity. By focusing on durable, consistent, relatable, sensical, and observable goals, we create a more robust view into our developers' days. By combining sentiment and behavioral data to measure these goals, we enable data-driven and human-centered insights: insights like the differences in frequency and duration of journeys taken by our most- versus least-well supported developers as they work to "ensure the code written by others is high quality."

As developer tools and infrastructure evolve, especially with large changes like the integration of AI tools, the ability to center the developer through goal-based and technology-agnostic frameworks will become an even more critical piece of our decision-making infrastructure. Our list of 30 goals, grounded in extensive research and designed to be adaptable to future technological shifts, provides a solid foundation for this type of ongoing work. Ultimately, this approach empowers us to create tools and infrastructure that truly support developers in achieving their goals, driving innovation, and delivering value.

ABOUT THE AUTHORS

BENJAMIN FERRARI-CHURCH is a user experience research manager on the Developer team at Google in Sunnyvale, CA 94089 USA. Contact him at ferrarichurch@google.com.

CAROLYN EGELMAN is a quantitative user experience researcher on the Engineering Productivity Research team at Google in Sunnyvale, CA 94089 USA. Contact her at cegelman@google.com.

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. J. Tan, "User 'Journeys': A tool to align cross-functional teams," in *Proc. Int. Commun. Comput. Inf. Sci. (HCI)*, C. Stephanidis, M. Antona, S. Ntoa, and G. Salvendy, Eds., Cham, Switzerland: Springer-Verlag, 2023, vol 1834, pp. 163–167.
3. C. Jaspan and C. Green, "Developer productivity for humans, part 1: Hybrid productivity," *IEEE Softw.*, vol. 40, no. 2, pp. 13–18, Mar./Apr. 2023, doi: 10.1109/MS.2022.3229418.
4. S. D'Angelo et al., "Measuring developer experience with a longitudinal survey," *IEEE Softw.*, vol. 41, no. 4, pp. 19–24, Jul./Aug. 2024, doi: 10.1109/MS.2024.3386027.
5. 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.