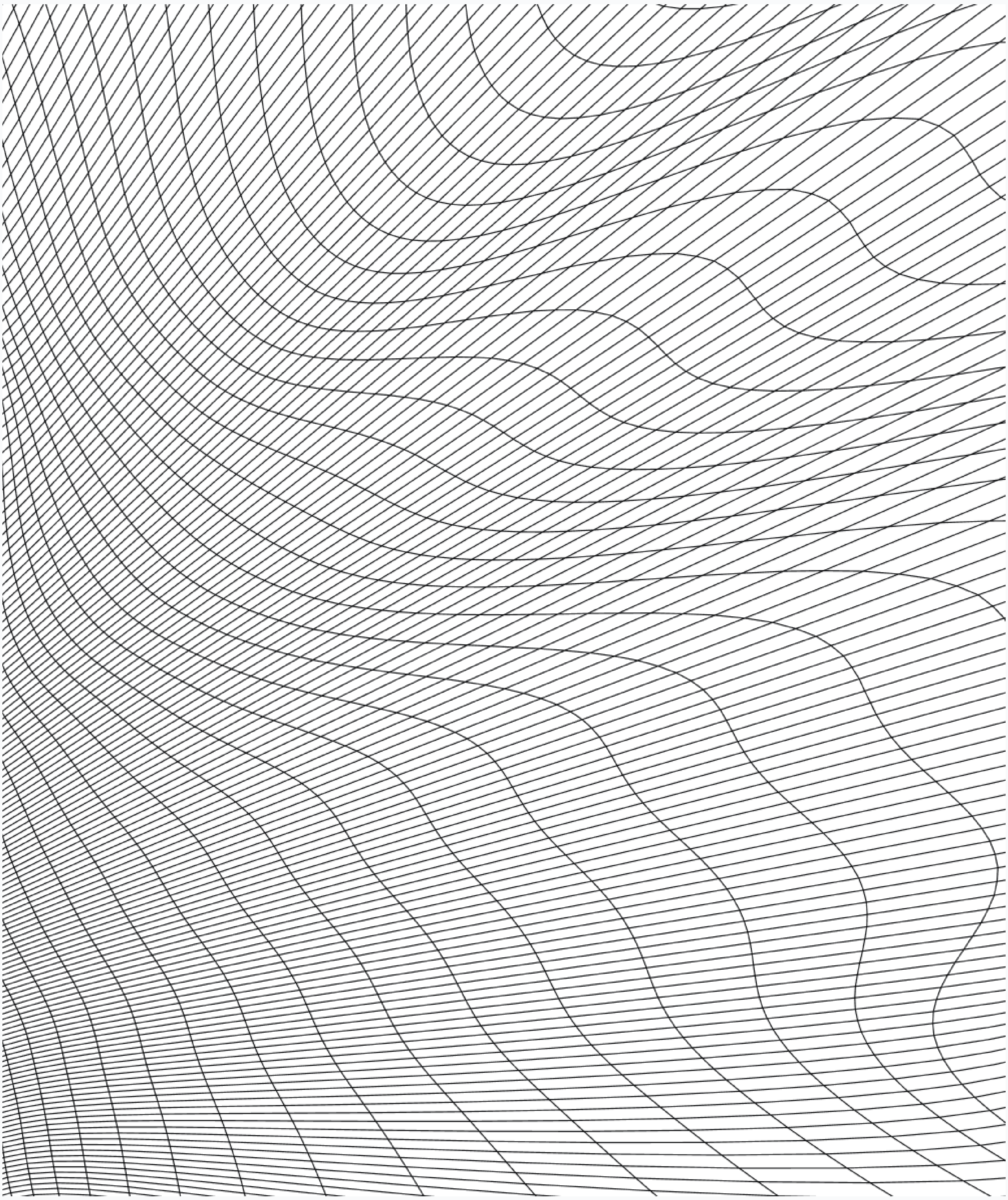
EXPERT GUIDE

Designing Developer Experience Surveys



Leading technology companies like Google, Microsoft, and Spotify have long relied on surveys as a way to measure and understand developer productivity. These surveys, which are typically run quarterly or semi-annually across an organization's developer population, provide leaders and teams with a comprehensive view of the developer experience (DevEx).

Surveys are essential for understanding developer productivity because basic metrics such as cycle time and throughput do not give us the full picture. By asking developers about their tools, processes, codebases, and team dynamics, organizations can get a more complete understanding of productivity and how to improve. Designing and running DevEx surveys isn't an easy task.

Companies like Google and Shopify staff dedicated teams of researchers, engineers, and data scientists that design and analyze their surveys. Sustaining high participation rates is a persistent challenge that requires careful planning and attention.

This guide provides step-by-step instructions for designing and running developer experience surveys in your own organization. We cover survey design, launch, and analysis, deriving from our experience partnering with hundreds of organizations to design and implement DevEx surveys.

The Landscape

While this guide focuses on "state of" surveys that are run quarterly or semi-annually, it can be helpful to start with the bigger picture of the different types of surveys that exist and the types of data that can be collected.

This micro-workshop from DX's CTO, Laura Tacho, gives the full landscape and offers advice on how to choose the right option for your company.

Designing Your Survey

An effective DevEx survey captures data that helps leaders and teams understand and improve productivity. Determining what to measure and how to capture reliable data can be a lengthy process. In this first section of our guide, we cover the key steps involved in survey design. Since survey design is critical to collecting reliable metrics and insights, we recommend hiring or consulting experts in this area as needed.

Select topics to measure

To get a holistic picture of the developer experience, leaders should include topics that span the full range of activities, tools, and processes encountered by developers. The topics may include areas such as code quality, automated testing, cross-functional collaboration, and psychological safety. Below are three recommended approaches for selecting topics:

Review research literature. There is a substantial body of research that examines factors that drive developer productivity. This research can be helpful in providing initial ideas of topics to focus on. Researchers from DX have published several research papers specific to developer experience: one identifies 25 top factors affecting developer experience, and another provides a practical framework for identifying what to measure.

Interview developers. Interviewing developers can provide a basis for determining which tools and processes should be touched on in your survey. For example, Peloton's Tech Insights team interviewed over 100 developers to understand the pain points they faced in their daily work. "We pulled this information into a journey map, which we call 'a day in the life of a developer at Peloton,'" says Thansha Sadacharam, who leads the Tech Insights team. "This was really helpful for shaping our survey."

Collect ideas from other teams and stakeholders. It's a good idea to reach out to leaders and stakeholders across your organization to gather their suggestions for what to cover in your survey. For example, internal platform teams may be eager to capture feedback about specific internal tools or workflows. Or, front-line managers may be interested in getting data about team morale. By incorporating suggestions from others, your survey becomes a more valuable asset for your organization.

Capture two types of measurements

To fully understand the developer experience, it's important to capture both developers' attitudes and opinions as well as more objective information about engineering systems and processes. A recent paper by DX researchers refers to these distinct types of measures as perceptual measures and workflow measures.

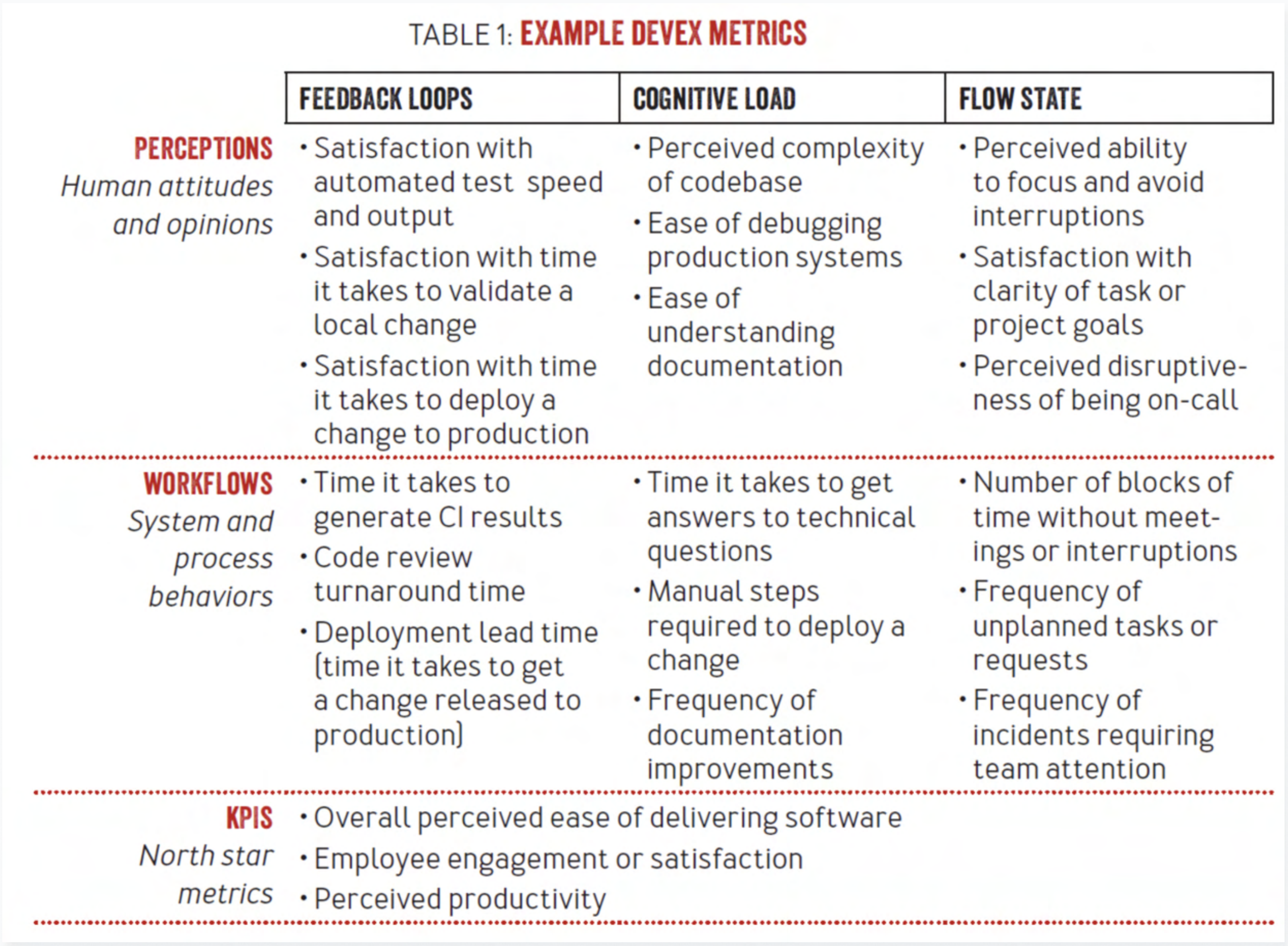


TABLE 1: **EXAMPLE DEVEX METRICS**

	FEEDBACK LOOPS	COGNITIVE LOAD	FLOW STATE
PERCEPTIONS *Human attitudes and opinions*	• Satisfaction with automated test speed and output • Satisfaction with time it takes to validate a local change • Satisfaction with time it takes to deploy a change to production	• Perceived complexity of codebase • Ease of debugging production systems • Ease of understanding documentation	• Perceived ability to focus and avoid interruptions • Satisfaction with clarity of task or project goals • Perceived disruptive-ness of being on-call
WORKFLOWS *System and process behaviors*	• Time it takes to generate CI results • Code review turnaround time • Deployment lead time (time it takes to get a change released to production)	• Time it takes to get answers to technical questions • Manual steps required to deploy a change • Frequency of documentation improvements	• Number of blocks of time without meet-ings or interruptions • Frequency of unplanned tasks or requests • Frequency of incidents requiring team attention
KPIS *North star metrics*	• Overall perceived ease of delivering software • Employee engagement or satisfaction • Perceived productivity		

Perceptual measures capture developers' perceptions, attitudes, and feelings, which provide insights into areas of friction in the software delivery process. In contrast, workflow measures capture objective information about system and process performance, giving us clarity into what's happening.

Both sentiment and workflow measures are necessary because neither alone can tell the full story. For instance, data on code review workflows may indicate a fast process, but in reality, developers may feel frustrated from being regularly interrupted by the process. Reliably measuring perceptions and workflows requires careful survey design methods, which we cover next.

Develop reliable survey items

It’s easy to develop survey questions that yield misleading or unreliable results. There are two key qualities that make for good survey questions: validity and reliability.

Validity refers to how well the results of a study measure what they are intended to measure. Validity includes three components:

- **Content validity:** Do the questions within the survey adequately evaluate the target topic? This requires the questions to come from research or an otherwise reliable framework which defines the topic. If we want to measure developer experience, the survey questions should come from a reliable framework defining what developer experience is.
- **Criterion validity:** Do measures correlate with other criteria in a way that fits theory? For example, code quality has predictive ability with perceived productivity, therefore a question related to code quality has criterion validity.
- **Construct validity:** Does the item correlate with other measures about the same topic? If we measure happiness, an abstract concept, we must have confidence that developers with higher ratings on the survey question are in fact happier.

Reliability refers to the consistency of a measure, across time and people:

- **Inter-rater reliability** refers to whether different respondents give the same response for the same question. If you ask two developers on the same team about build times and you get very different responses, the question is not reliable.
- **Test-retest reliability** refers to whether the same respondent will give you the same responses when nothing about their experience or attitude has changed. If you ask a developer about the on-call experience on Monday, and ask them the same question on Wednesday, their response should be the same assuming nothing has changed.

Be mindful of survey length

Long surveys can hamper participation and response quality as respondents experience fatigue. For reference, developer surveys at companies like Google and Microsoft typically require around 30 minutes to complete. Our research, however, shows that keeping surveys under 15 minutes leads to optimal participation rates over time.

To reduce your survey length, review questions and ask yourself the following:

- **What decision or action will this inform?** If this is unclear or no action will be taken, remove the question.
- **Is this question already addressed by another item?** It is common to end up with multiple questions that are similar.
- **Can this question be merged with another and turned into a broader question?** If the level of detail gained by asking multiple questions about a single topic is not necessary, remove the question.

Running Your Survey

Leaders should carefully plan the survey's rollout: even perfectly designed surveys can fall flat if not well-communicated. With proper planning and communication, our research shows that developer surveys can achieve and sustain 80%+ participation rates. In this section, we will cover the key elements that are part of successful survey rollout.

Utilize sampling

Sampling can improve participation rates by reducing how often individual developers take surveys. Sampling is the process of using a subset of the population to represent the whole population. For example, Google splits their developer population into three cohorts. They send their developer survey quarterly, so developers will either take the survey once or twice per year.

When utilizing sampling, it's important to collect sample sizes that are large enough to provide results that leaders can use to inform decisions. It is also important to make sure that sampled populations are representative of your full population by utilizing random sampling methods.

Optimize survey timing

The timing and cadence of surveys are important considerations. Surveys should ideally be conducted at times where the results can be meaningfully acted upon, such as during annual or quarterly planning processes. It is also important to be steer clear of overlaps with other organizational surveys being conducted, such as those run by HR.

Your survey cadence should stay consistent and be optimized to the needs of your organization. Quarterly and semi-annual cadences are most common. For example, companies like Google and Dropbox send out surveys quarterly, whereas companies like Atlassian and Shopify send surveys twice per year.

Create a communication plan

Strong communication plans ensure that people across your organization understand what's happening with your survey and feel encouraged to engage. Communication is beneficial at all points along the lifecycle of your survey, but the following should be included, at minimum:

1. **One week prior to the survey.** Give your organization a heads-up about the survey with an explanation of its purpose, how long the survey will take to complete, and what individuals should expect after.
2. **At the onset of the survey.** Announce the survey along with instructions on how to complete it. Reiterate the purpose of the survey in order to encourage participation.
3. **After the survey.** Thank developers for their participation and report on the results. Highlight notable findings, surprises, and trends, especially those that relate to recent initiatives.
4. **Before the subsequent survey.** Share wins and progress made since the previous survey in order to encourage participation in the upcoming survey.

Encourage participation

Companies typically see survey response rates of anywhere from 30 to 90%. Several strategies can help boost participation:

- **Send regular reminders.** Friendly reminders can help nudge people to complete the survey who may have otherwise forgotten. It's important to be considerate and not overdo reminders, however.
- **Ask executives to vocalize the importance of participating.** Developers are more likely to participate in surveys when they feel their feedback matters to senior leadership.
- **Offer incentives or rewards.** Some organizations, such as Microsoft, motivate developers to participate by offering small incentives such as gift cards or company swag.

Taking Action on the Results

Once your survey closes, the next step is to analyze the responses to unlock useful insights and learnings from your data. Here, we share strategies for analyzing and reporting results.

Analyze the results

To identify where to invest, leaders should analyze survey data with a focus on answering the following key questions:

- **What are the biggest priorities?** Score and rank the different areas you've assessed in order to surface clear suggested priorities.
- **How do various personas differ?** Break down your results by dimensions such as role, technical skills, tenure, and location. This can yield valuable insights into the unique challenges and needs of different personas.
- **How do perceptions and workflows match up?** Look for discrepancies between perceptual measures and workflow measures. For instance, if developers are satisfied with certain processes yet workflow measures show these processes as underperforming, this would prompt us to dig in and understand why.
- **Which issues cross-cut the organization?** Which issues are local team problems? What issues are uniquely painful for specific teams? What issues are felt more broadly across teams? This information will guide investments that can be addressed by individual teams, the platform team, or with a temporary tiger team.
- **How does your organization compare to others?** Comparing your results against industry benchmarks can help contextualize your data. For example, developer sentiment toward technical debt is near-universally negative. Industry benchmarks can help you understand whether sentiment is more or less negative than the typical organization.
- **What has changed since the previous survey?** By including the same survey items across surveys over time, you can understand how developer experience is changing over time. These trends can be invaluable for tracking the impact of organizational growth and changes.

Present results to leadership

Presenting valuable insights to executive leaders is one of the key ways to increase support and investment for future surveys. Executive briefings should provide a quick overview of highlighted findings, along with recommendations or proposals on initiatives that can drive improvement.

"You have to translate what you're saying into a language that executives can understand, which is typically finance," says Mike Fisher, former CTO at Etsy. At Etsy, leaders use "back of the envelope" calculations to explain the potential impact of projects. For example, they would highlight the cost of time wasted by inefficient tools or processes, or the cost of reducing attrition.

Jim Wang, VP of Internal Developer Experience at GitHub, recommends organizing projects into categories that leaders can easily grasp. GitHub groups into three categories: developer friction, cutting costs, and improvements to the GitHub platform. "Being able to advocate for your developer experience initiatives from these perspectives is very powerful."

Distribute team-specific results

Executive leaders aren't the only ones interested in taking action on the results. Managers and teams on the front lines can also benefit from pinpointing local issues and determining steps they can take to improve.

To enable this, team-specific survey results should be distributed across your organization along with guidance on how to utilize their data effectively. Instructing teams to choose focus areas and establish concrete goals or OKRs can help spark improvement. Leaders can further encourage teams by treating developer experience as a priority, and celebrating progress that teams are able to make.

Ultimately, developer experience surveys are a critical tool for organizations seeking to improve developer productivity. With careful design, rollout, and analysis, organizations can gain comprehensive insights into areas of friction in order to prioritize projects that drive meaningful change.

Developer experience surveys are a critical tool for organizations seeking to improve developer productivity. With careful design, rollout, and analysis, organizations can gain comprehensive insights into areas of friction in order to prioritize projects that drive meaningful change.