CHAPTER 3

Why We Should Not Measure Productivity

Amy J. Ko, University of Washington, USA

Software moves faster every year. Markets shift rapidly, releases are ever more frequent, and languages, APIs, and platforms evolve at a relentless pace. And so the interest in productivity, both by developers who want to keep up with these changes and by managers and organizations that need to compete, appears entirely rational. Moreover, improving software faster holds even greater promise to the rest of humanity: getting more work done with less effort may mean an increased quality of life for everyone.

In pursuit of productivity, however, there can be unintended consequences from trying to measure it. Here are some examples:

- Measuring productivity can warp incentives, especially if not measured well.
- Sloppy inferences from measurements could result in *worse* management decisions rather than better ones.

Are these bad enough that we shouldn't even try to measure it? To find out, let's do a thought experiment. I want you to imagine an organization that you've worked for or are working for now. Let's consider what might happen if it invested seriously in trying to measure productivity. As we go, test the argument against your own experience.

Unintended Consequences

The first unintended consequence comes from trying to use any single concrete measure of productivity. Take, for example, a measure of productivity that focuses on *time to release*. An individual developer committing faster means a team reviewing faster, which ultimately means shipping faster, right? But unless your organization also measures the outcomes of shipping—positive outcomes such as adoption, customer growth, and sales increases, or negative outcomes such as software failures or harm to brand—one risks optimizing for an intermediate outcome at the expense of an organization’s ultimate goal.

For example, in the race to release, a team might ship more defects than it would have otherwise or take on more technical debt than is desirable for longer-term goals. Most other single metrics have the same problems. Counting the number of bugs closed, the number of lines of code written, the number of user stories completed, the number of requirements met, and even the number of customers acquired—if your organization tried to measure these, optimizing any one of them would almost always come at the expense of others.

But this is a bit obvious. I bet it’s even more obvious if you’ve been in an organization that did this because you probably lived those unintended consequences every day, feeling tension between the official measures of productivity and the other concerns that related to that measure. So, let’s take our thought experiment in a more radical direction.

Imagine it was possible for your organization to measure *all* dimensions of productivity. After all, software has a vast array of quality dimensions Redundant, as do software development methodologies. Perhaps measuring all of these dimensions can overcome any overfitting to one metric. Let’s put aside for the moment that we don’t know how to measure most of these dimensions well, imagining a future in which we can accurately observe and measure every dimension of work. Would a holistic, multidimensional metric of productivity be any better?

It would certainly make the activities of a team more *observable*. Developers and managers would know every aspect of every developer’s work, able to observe every dimension of progress or lack thereof. It would provide a perfect model of developer activity.

But this omniscient vision of software development work still comes with significant unintended consequences. First, if this monitoring were done at a team or organization level by managers, how would being monitored change developers’ behavior? The effect of being observed so thoroughly might actually result in developers self-monitoring

their every action, unintentionally *reducing* productivity. Even if this were a net increase in productivity, it might also lead to developers leaving the organization, moving to organizations that were a little less like Big Brother.

Explaining Productivity

For the sake of our thought experiment, let's imagine that you and every developer in your organization fully embraced rich monitoring of productivity of all kinds. What would a manager actually do with this data to improve productivity?

- They could use the data to rank the productivity of individual developers and teams to make promotion or investment decisions.
- If the data were real-time enough, they might use it to intervene in teams that are seeing drops in productivity.
- With enough detail, the data might even reveal which practices and tools are associated with increased productivity, allowing an organization to change practices to increase productivity.

This rich stream of real-time data could empower an organization to fine-tune its activities to more rapidly achieve its goals.

Unfortunately, there's a hidden requirement to achieve this vision. For a manager to actually go from data to intervention, they need to make a creative leap: a manager has to take all of the measures, correlations, and models to ultimately infer a theory for what *explains* the productivity they're observing. Making these inductive leaps can be quite challenging, and coming up with a wrong theory means any intervention based on that theory would likely not be effective and may even be harmful.

Even if we assume that every manager is capable of creatively and rigorously inferring explanations of a team's productivity and effectively testing those theories, the manager would need richer data about causality. Otherwise, they'd be blindly testing interventions, with no sense of whether improvements are because of their intervention or just the particular time and context of the test. Where would this causal data come from?

One source of richer data is *experiments*. But designing experiments requires control groups that are as close to identical as the treatment group or sufficiently randomized to control for individual differences. Imagine trying to create two teams that are identical in nearly every way, except for the process or tools they use, and randomizing everything else.

As a scientist of software engineering, I've tried, and not only is it extremely time-consuming and therefore expensive, but it's almost always impossible to do, even in the laboratory, let alone in a workplace.

Another source of rich data about causality is qualitative data. For example, developers could report their *subjective* sense of their team's productivity. Every developer could write a narrative each week about what was slowing them down, highlighting all of the personal, team, and organizational factors that they believe are influencing all of those elaborate quantitative metrics being measured in our omniscient vision. This would help support or refute any theories inferred from productivity data and might even surface some recommendations from developers about what to do about the problems they're facing.

This would be ideal, right? If we combine holistic *qualitative* data from developers with holistic *quantitative* data about productivity, then we'll have an amazingly rich and precise view into what is either causing or preventing an organization's desired level of productivity. What could be more valuable for improving developer productivity?

Dealing with Change

As usual, there's another fatal flaw. Such a rich model of productivity would be incredibly powerful if developers, teams, and organizations were a relatively stable phenomena to model. But new developers arrive all the time, changing team dynamics. Teams disband and reform. Organizations decide to enter a new market and leave an old one. All of these changes mean that the phenomena one might model are under constant change, meaning that whatever policy recommendations our rich model might suggest would likely need to change again in response to these external forces. It's even possible that by having such a seamless ability to improve productivity, one would accelerate the pace at which new productivity policies would have to be introduced, only creating more entropy in an ever-accelerating system of work.

One final flaw in this thought experiment is that, ultimately, all productivity changes will come from changes in the behavior of developers and others on a team. Depending on their productivity goals, they'll have to write better code, write less code, write code faster, communicate better, make smarter decisions, and so on. Even with a perfect model of productivity, a perfect understanding of its causes in an organization, and a perfect policy for improving productivity, developers will have to learn new skills, changing how they program, communicate, coordinate, and collaborate to implement

more productive processes. And if you've had any experience changing developer or team behavior, you know how hard it is to change even small things about individual and team behavior. Moreover, once a team changes its behavior, one has to understand the causes of behavior all over again.

This thought experiment suggests that regardless of how accurately or elaborately one can measure productivity, the ultimate bottleneck in realizing productivity improvements is behavior change. And if our productivity utopia relies on developer insight into their own productivity to identify opportunities for individuals to change, why not just focus on developers in the first place, working with them individually and in teams to identify opportunities for increased productivity, whatever the team and organizational goals? This would be a lot cheaper than trying to measure productivity accurately, holistically, and at scale. It would also better recognize the humanity and expertise of the people ultimately responsible for achieving productivity. A focus on developers' experiences with productivity also leaves room for all the *indirect* components of productivity that are far too difficult to observe, including factors such as developers' motivation, engagement, happiness, trust, and attitudes toward the work they are doing. These factors, likely more than anything else, are the higher-order bits in how much work a developer gets one per unit time.

Managers as Measurers

Of course, all these individual and emotional factors about probing developer experience are just fancy ways of talking about *good management*. Great managers, by respecting the humanity of the people they are managing and understanding how their developers are working, are constantly building and refining rich models of their developers' productivity all the time and using them to make identify opportunities for improvements. The best ones already achieve our productivity measurement ideal but through interpersonal communication, interpretation, and mentorship. The whole idea of *measuring* productivity is really just an effort to be more objective about the subjective factors that are actually driving software development work.

So, what does this mean for improving productivity? I argue that instead of *measuring* productivity, we should instead invest in finding, hiring, and growing managers who can *observe* productivity as part of their daily work with developers. If organizations grow good managers and can trust that their great managers will constantly seek ways to improve productivity, developers will be more productive, even if we can't objectively measure it.

Of course, part of growing good management *can* involve measurement. One can think of measurement like a form of self-reflection scaffolding, helping a manager to reflect on process in more structured ways. That structure might help inexperienced managers develop more advanced skills of management observation that do not necessarily involve counting things. More advanced managers can be more intuitive, gathering insights as they work with their team and making changes to team dynamics as the world around the team changes. This vision of management ultimately frames measurement as just one small tool in a much larger toolbox for organizing and coordinating software development work.

Now all we need is a measure of good management.

Key Ideas

The following are the key ideas from the chapter:

- Improving productivity requires explaining the factors that affect it, but that requires qualitative insights into team behavior.
- Teams are always changing, making it even harder to get insights about team behavior through data.
- Managers are best positioned to get these qualitative insights by interacting with their team.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits any noncommercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if you modified the licensed material. You do not have permission under this license to share adapted material derived from this chapter or parts of it.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.