CHAPTER 6

Individual, Team, Organization, and Market: Four Lenses of Productivity

Amy J. Ko, University of Washington, USA

When we think about productivity in software development, it's reasonable to start with a basic concept of work per unit of effort. The more work a developer accomplishes with their efforts, the better.

But when researchers have investigated how developers think about productivity, some surprising nuances surface about what software engineering "work" actually is and at what level this work should be considered [14]. In particular, there are four lenses through which one can reason about productivity, and each of these has different implications for what actions one might take to increase productivity in a company.

The Individual

The first and most obvious lens is the *individual* perspective. For a developer, a tester, or any other contributor to a software team, it's reasonable to think about the tasks they are assigned, how efficiently those tasks can be completed, and what affects how efficiently those tasks are completed. Obviously, a developer's experience—what they've learned in school, online, or in other jobs—can affect how efficiently they accomplish tasks. For example, one study showed that in terms of task completion time, the skill of *comprehending* what a program does explains much of the variance in task completion

C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering*,
https://doi.org/10.1007/978-1-4842-4221-6_6

time [3]. But these skills aren't static. For example, while one might expect inexperienced developers to always be less efficient than experts, teaching novices expert strategies can make them match expert performance quite quickly [17]. As any developer knows, however, there's no such thing as mastery; even senior developers are always engaged in learning new concepts, architectures, platforms, and APIs [5]. This constant learning is even more necessary for new hires, whose instincts are often to hide their lack of expertise from the people they need help from [1].

But experience isn't the only factor that affects individual productivity. For example, we know that tools strongly influence how efficiently a development task can be completed. IDEs, APIs, and programming languages, for example, pose many barriers, including finding relevant APIs, learning to use them correctly, and learning to test and debug them correctly [7]. For example, one study found that simply using rudimentary tools for navigating code (scroll bars, text search, etc.) can account for up to a third of the time spent debugging code [8]. Another study found that tracking the specific structural elements in code that a developer navigates and making those structures and their dependencies visible can nearly reduce this overhead [6].

Having the right documentation with the right information (e.g., Stack Overflow or other sources of information about API usage) can also accelerate program construction [11], but when that documentation is wrong, it can actually have the opposite effect on time to complete tasks [18].

These discoveries have some simple implications for individual developer productivity. For example, teaching developers strategies that have proven to be more effective seems like an unqualified win. Training developers on tools that increase productivity is a potentially cheap way to help developers get more work done in the same amount of time.

The Team

And yet, when we use a *team* lens on productivity, some of these improvements to developer productivity suddenly seem less important. For example, if one developer is twice as efficient as others on a team but is constantly blocked waiting for work from others, is the team really more productive? Research shows that team productivity is actually bounded not by how efficiently individual developers work but by

communication and coordination overhead [5]. This is partly because teams work only as fast as decisions can be made, and many of the most important decisions are not made individually but collaboratively. However, this is also because even for individual decisions, developers often need information from teammates, which studies have shown is always one or two orders of magnitude slower to obtain than referencing documentation, logs, or other automatically retrievable content [10]. These interactions between individual productivity and team work are also affected by changes in team membership: one study found that *slowly* adding people to a team (i.e., waiting for them to successfully onboard) *reduced* defects, but *quickly* adding them *increased* in defects [13].

Other team needs can lower productivity for individuals but increase it for the team. For example, interruptions can be a nuisance for individual developers, but if they have knowledge that others need to be unblocked, it may improve team productivity overall. Similarly, senior developers may need to teach skills or knowledge to junior developers to help junior developers be independently productive. That will reduce the senior developer's productivity for a time but will probably increase the team's long-term productivity.

If we view a team's work as correctly meeting requirements, then the influence of communication and collaboration on a team is clearly just as important as the productivity of individual developers on meeting those requirements. Finding a way to manage teams that streamlines communication, coordination, and decision-making is therefore key and perhaps more impactful than making individual developers faster. All of these responsibilities fall upon an engineering manager, whose notion of productivity isn't about how *efficiently* individual engineers work but rather about how efficiently a team can meet high-value requirements.

The Organization

Even a team lens, however, is a narrow view. An *organizational* lens reveals other important factors. For example, companies often set norms around how projects are managed, and these norms can greatly influence how efficiently work can move at the individual and team levels [4]. Organizations also set policies on whether developers are collocated, work down the hall, work at home, or work in entirely different countries. These policies, and their implications for coordination, can directly affect the speed of decisions proportionally to distance [16]. Organizations can also set formal policies and

informal expectations about work-life balance, which can inadvertently lead to fatigue and defects [9]. Organizations have different norms of code ownership, which affects coordination within and between teams and can lead to defects when no one owns part of an implementation [2]. Organizations also invest infrastructure for maintaining awareness of work in other parts of the organization [12], such as Google, which has a single company-wide repository, versus other companies that have vast numbers of disconnected repositories. Companies also have different norms about how interruptions are handled, which can have organization-wide detrimental effects on productivity [15]. All of these cultural and policy factors can also complicate the recruiting and retention of productive developers, as we observed with Yahoo's decision to require that all engineers work on the main Yahoo campus.

Given all of these complex factors of organizational culture, one might imagine that a fruitful way to think about productivity from an organizational perspective is to reason about the unintended consequences of norms and policies on individual and team productivity. An organization's executives might be charged with monitoring for these problems and developing new policies, norms, and processes with fewer impacts on productivity.

The Market

Finally, the organizational lens has its own limitations. Viewing productivity from a *market* lens acknowledges that the whole purpose of an organization that creates software is to provide *value* to customers and other stakeholders. When Google says its mission is to "organize the world's information," it's stating the goal by which the entire organization's performance is judged. Google is therefore more effective when its users are more productive at finding information and answering questions relative to other organizations with similar goals. To measure productivity in terms of value, a company has to define *value propositions* for its product, which is some hypothesis about what value a product is offering to people relative to competing solutions. Some research has framed the refinement and measurement of value propositions as an organization's primary goal [9]. These ever-evolving understandings of an organization's goal then filter down to new organizational policies, new team-level project management strategies, and new developer work strategies targeted at improving this top-level notion of productivity.

Full-Spectrum Productivity

While it's easy to assume that each individual in an organization might have to concern themselves with only one of these lenses, studies of software engineering expertise show that great developers are capable of reasoning about code through *all* of these lenses [5]. After all, when a developer writes or repairs a line of code, not only are they getting an engineering task done, they're also meeting a team's goals, achieving an organization's strategic objectives, and ultimately enabling an organization to test its product's value proposition in a market. And the code they write can be seen as a different thing through each of these lenses, including not just code but also systems, software, platforms, and services, and products.

What does all of this mean for *measuring* productivity? It means you're not going to find one measure for everything. Individuals, teams, organizations, and markets need their own metrics because the factors that affect performance at each of these levels are too complex to reduce to a single measure. I actually believe that individual developers, teams, organizations, and markets are so idiosyncratic that each may need its own unique measures of performance that capture a valid notion of their work output (productivity, speed, product quality, actual versus plan, etc.). That might mean a core competency of everyone in an organization needs to be finding valid ways of conceiving of performance so one can measure and improve it.

Key Ideas

The following are the key ideas from this chapter:

- Individuals, teams, organizations, and markets need different productivity metrics.
- Productivities for these different lenses are often in tension.

References

[1] Begel, A., & Simon, B. (2008). Novice software developers, all over again. ICER.

[2] Bird, C., Nagappan, N., et al. (2011). Don't touch my code! Examining the effects of ownership on software quality. ESEC/FSE.

[3] Dagenais, B., Ossher, H., et al. (2010). Moving into a new software project landscape. ICSE.

[4] DeMarco, T. & Lister, R. (1985). Programmer performance and the effects of the workplace. ICSE.

[5] Li, P.L., Ko, A.J., & Zhu, J. (2015). What makes a great software engineer? ICSE.

[6] Kersten, M., & Murphy, G. C. (2006). Using task context to improve programmer productivity. FSE.

[7] Ko, A. J., Myers, B. A., & Aung, H.H. (2004). Six learning barriers in end-user programming systems. VL/HCC.

[8] Ko, A.J., Aung, H.H., & Myers, B.A. (2005). Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks. ICSE.

[9] Ko, A.J. (2017). A Three-Year Participant Observation of Software Startup Software Evolution. ICSE SEIP.

[10] LaToza, T.D., Venolia, G., & DeLine, R. (2006). Maintaining mental models: a study of developer work habits. ICSE SEIP.

[11] Mamykina, L., Manoim, B., et al. (2011). Design lessons from the fastest Q&A site in the west. CHI.

[12] Milewski, A. E. (2007). Global and task effects in information-seeking among software engineers. ESE, 12(3).

[13] Meneely, A., Rotella, P., & Williams, L. (2011). Does adding manpower also affect quality? An empirical, longitudinal analysis. ESEC/FSE.

[14] Meyer, A.N., Fritz, T., et al. (2014). Software developers' perceptions of productivity. FSE.

[15] Perlow, L. A. (1999). The time famine: Toward a sociology of work time. Administrative science quarterly, 44(1).

[16] Smite, D., Wohlin, C., et al. (2010). Empirical evidence in global software engineering: a systematic review. ESE, 15(1).

[17] Benjamin Xie, Greg Nelson, and Amy J. Ko (2018). An Explicit Strategy to Scaffold Novice Program Tracing. ACM Technical Symposium on Computer Science Education (SIGCSE).

[18] Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., & Fahl, S. (2017). Stack overflow considered harmful? The impact of copy&paste on android application security. IEEE Symposium on Security and Privacy (SP).