CHAPTER 7

# Software Productivity Through the Lens of Knowledge Work

Emerson Murphy-Hill, Google, USA

Stefan Wagner, University of Stuttgart, Germany

While this book focuses on software developer productivity, other fields have studied productivity more broadly. Such work lends a perspective that can contribute to a solid foundation to what we know about software developer productivity. In this chapter, we provide an overview of related work about perhaps the most relevant allied field outside of software engineering, namely, the productivity of *knowledge workers*.

## A Brief History of Knowledge Work

The term *knowledge work* was coined by the management guru Peter Drucker in 1959 [1]. Unlike manual labor where the main output is largely physical goods, knowledge workers deal primarily with information, where each task is usually different from the last, and the main output of the work is knowledge.

Later, Drucker challenged the field of management research to improve the productivity of knowledge workers in the same way they improved the productivity of manual laborers [2]. Drucker's contrast of knowledge worker productivity against manual worker productivity is insightful. While productivity of the manual worker can be improved by understanding and automating the routine steps involved in creating a physical good, the steps involved in the tasks performed by knowledge workers are so nonroutine that similar kinds of automation cannot be easily employed.

For the past half-century, studies in management and other social sciences have examined how to improve the productivity of the knowledge worker. Because software developers are one kind of knowledge worker, it stands to reason that much of what such studies have learned will be applicable to software developer productivity as well.

Studies about knowledge workers can teach us at least two things about productivity of software developers: techniques for measuring productivity and a set of drivers that have been shown to affect knowledge worker productivity. We next discuss each in turn.

## Techniques for Measuring Productivity

As we discuss elsewhere in this book, measuring software developers' productivity is challenging, and likely no single metric will do (see Chapters 2 and 3). This problem also afflicts researchers in knowledge work, yet they have made progress on the problem by developing a breadth of techniques for measuring productivity. We next describe the techniques used to measure knowledge worker productivity by turning to a taxonomy of techniques from Ramírez and Nembhard [4]. We describe some of those techniques and discuss the trade-offs in using each technique. Further, we group these techniques into four categories, which we call outcome-oriented, process-oriented, people-oriented, and multi-oriented techniques. Software engineering practitioners and researchers can use these categories to choose appropriate productivity measures for their contexts.

## Outcome-Oriented Techniques

In the original literature on improving the productivity of manual workers, it was common to measure productivity by looking primarily at the output of work per unit time. For software developers, this could be realized by measuring the number of lines of code written per day, for instance. This measurement technique has also been extended in knowledge worker research by accounting for inputs to the process—such as resources or salaries used by the workers. Such outcome-oriented techniques have the advantage of being relatively straightforward to measure. However, as Ramírez and Nembhard point out, the knowledge worker research community has largely converged on the opinion that such outcome-oriented techniques are generally inadequate because they fail to take into account output *quality*, which they generally regard as a critical aspect of productivity. See Chapter 5 for an in depth discussion of the importance of quality when measuring productivity. An additional challenge to outcome-oriented metrics for software engineering is that difficult software problems may have similar-appearing output to easy problems.

Another refinement of these outcome-oriented techniques is using organizational economic output as the outcome, such as a company's earnings. The main advantage of this approach is that economic output is arguably the most direct measure of productivity, at least at a large scale—if a developer's work does not produce profit directly or indirectly, are they really being productive? The disadvantages of this approach is that, as Ramírez and Nembhard point out, tracing profits down to individual knowledge workers is difficult and also that present economic output is not necessarily indicative of future potential economic output. In complex software organizations, measuring the economic effect of key but indirect developers—such as open source developers or infrastructure teams—is relatively challenging.

## Process-Oriented Techniques

Rather than looking at the outcomes of work, some studies examine how knowledge workers' tasks are performed. For instance, using the *multiminute measurement* technique, knowledge workers fill out forms at regular intervals, reporting what they have done from a predefined list of tasks. Building on this, productivity measurement techniques can measure the time spent in value-added activities, which looks at what percentage of time knowledge workers spend doing desirable activities compared to the total number of hours worked. In software engineering, we could define desirable activities as activities that add value to the software product. This could include constructive activities, such as writing code, but also analytical, improving activities, such as performing code reviews. The advantage of such techniques is that they are amenable to some amount of automation, such as through experience sampling tools (for example, `www.experiencesampler.com/`) or instrumentation like RescueTime (`https://www.rescuetime.com/`). The primary disadvantages are that simply measuring activities doesn't measure how well knowledge workers conduct those activities and that it doesn't take into account quality. To the latter point, some activity-tracking techniques have also been extended to measure quality-enhancing activities, such as by counting thinking and organizing as activities that enhance quality and thus enhance productivity. This shows, however, that it is difficult to clearly distinguish between value-adding and non-value-adding activities. Potentially, the categorization of *waste* could be useful (see Chapter 19).

## People-Oriented Techniques

In contrast to the prior techniques, which seek to define productive outcomes and activities up-front, people-oriented techniques empower knowledge workers to define metrics for productivity for themselves. One way to do this is through the *achievement method*, which measures productivity by determining the ratio of completed goals to planned goals. An extension of the achievement method is the *normative productivity measurement* methodology, which works to establish consensus among knowledge workers about the different dimensions of productivity. The advantage of these techniques is that measuring productivity as completion of self-determined goals has good construct validity, as research suggests that task or goal completion is the top reason that software developers report having a productive workday [5].

Using interviews and surveys to measure productivity is "a straightforward and commonly used method" to measure knowledge worker productivity and to determine knowledge worker compensation [4]. Such techniques have the advantage of being relatively easy to administer with existing instruments from the literature and can capture a wide variety of productivity factors. On the other hand, such techniques may have low reliability. To increase the reliability of these techniques, many studies have used *peer evaluations*, where knowledge workers rate their peers' productivity. However, the disadvantage of this technique is the so-called halo effect, where a peer might rate a knowledge worker's past performance as indicative of their current performance, even if past and present productivity are unrelated.

## Multi-oriented Techniques

As we describe in Chapters 5 and 6, productivity can be measured through multiple facets within an organization; likewise, the knowledge worker literature has sought to understand productivity through multiple facets. For example, the *multiple output productivity indicator* can be used to measure productivity when a knowledge worker has more than one output. For instance, a software developer not only produces code but also produces infrastructure tools and trains peers in organizational development practices. A multiple-level productivity measurement technique is the *macro, micro, and mid-knowledge worker productivity models*, which seeks to measure productivity at the factory, individual contributor, and department levels, respectively. This technique measures productivity over time using attributes such as quality, cost, and lost time. The main advantage of these techniques is that they provide a more holistic view of organizational productivity than many other metrics, but at the same time, collecting them can be complex.

These three kinds of techniques—process-, people-, and multi-oriented—provide a variety of options for practitioners and researchers to use. One way these techniques can be used is to enable those who want to measure productivity to use off-the-shelf, validated techniques, rather than creating new techniques with unknown validity. Another way these techniques can be used is as a framework to broaden productivity-measurement efforts; if an organization is already using process-oriented productivity techniques, they could broaden their portfolio by adding people-oriented techniques. Similarly, researchers can choose multiple techniques to increase the validity of their studies through triangulation.

## Drivers That Influence Productivity

The second major contribution of research on knowledge workers that can be applied to software engineers is an understanding of what drivers can change knowledge workers' productivity. Understanding productivity drivers is valuable because it tells organizations what changes they can make to improve knowledge worker productivity. While some productivity drivers are specific to software development, such as code complexity (see also Chapter 8), other drivers probably apply equally well to knowledge workers generally and software developers specifically, such as the need for quiet spaces required for concentration.

We draw on prior research, which we have found personally insightful, that catalogs productivity drivers among knowledge workers. In an attempt to measure knowledge worker productivity, Palvalin created SmartWoW, a survey that captures all the drivers that affect productivity, according to the knowledge work literature [3]; readers who want to know the strength of the scientific evidence for each factor are encouraged to explore the research cited by Palvalin. Palvalin showed that his survey has reasonable validity and reliability by assessing it at nine companies with almost 1,000 knowledge workers. SmartWoW divides productivity drivers into five types, which we describe here:

**Physical environment.** The physical environment refers to the place where the work occurs, whether that's in the office or at home. Studies of knowledge workers have found that a physical environment that increases productivity is one where there is adequate space for solitary work for concentration, official and unofficial meetings, and informal collaboration. A physical environment that enhances productivity also has good ergonomics with low noise and few interruptions. Software developers' frequent complaints about open offices underscore the importance of work environment drivers.

**Virtual environment.** The virtual environment refers to the technology that knowledge workers use. A virtual environment that enhances productivity is one where the technology is easy to use and available wherever the knowledge worker is working. Knowledge work studies have also identified several specific types of technology as productivity-enhancing, including use of instant messaging, video conferencing, access to co-workers' calendars, and other collaborative groupware. This research suggests that usable programming languages and powerful tools, as well as collaboration platforms like GitHub, are important for improving software developer productivity.

**Social environment.** The social environment refers to the attitudes, routines, policies, and habits performed by workers in an organization. Productive social environments are those where knowledge workers are given freedom to choose their work methods, work times, and work locations; information flows freely among workers; meetings are efficient; clear technology usage and communication policies exist; goals are cohesive and clearly defined; work is assessed in terms of outcomes, not just in terms of activities; and experimentation with new work methods is encouraged. A social environment for software development that enhances productivity is one where, for example, developers are given freedom to try new tools and methodologies. The importance of the social environment is underscored by Google's finding that psychological safety—that members of a team should be able to take risks without fear—is the most important predictor of effective teams.

**Individual work practices.** While the prior environmental drivers *enable* productive work through organizational practices, individual work practices measure to what extent knowledge workers will actually implement these practices. Productive individual work practices include knowledge workers using technology to reduce unnecessary travel, using mobile devices when waiting (e.g., during travel), prioritizing important tasks, using quiet spaces and shutting down disruptive software during tasks that require concentration, preparing for meetings, taking care of their well-being, using the organizations' official communication channels, planning out their workday, and experimenting with new tools and work methods. This suggests that developers are productive when, for example, they can code, test, and push while commuting to work on shared transit.

**Well-being at work.** Finally, Palvalin includes a knowledge worker's well-being at work both as a driver of productivity at work and as an outcome of productivity. A productive knowledge worker is one who enjoys and is enthusiastic about their work, finds meaning and purpose in their work, is not continuously stressed, is appreciated, has a work-life balance, finds the work atmosphere pleasant, and resolves conflicts with co-workers quickly. This suggests that the famous 80-hour workweek developer is not a productive developer.

## Software Developers vs. Knowledge Workers: Similar or Different?

In this chapter, we've drawn parallels between software developer and knowledge worker productivity, so it's natural to ask whether one should consider their productivity the same or different. Our opinion is that each extreme is a cop-out; considering software developer productivity the same as knowledge worker productivity would abdicate our responsibility to study the productivity of software developers, while considering them as entirely different would allow us to reinvent the wheel by ignoring prior studies about knowledge worker productivity.

The reality is that knowledge workers and software developers are similar in some ways and different in others, both in kind and in degree. In kind, arguably everything that could possibly affect software developer productivity can be pigeonholed into one the five types of productivity drivers described in the prior section, but doing so elides some drivers that software developers may be uniquely positioned to measure and change, such as software complexity. In degree, software developers' productivity is similar in some ways and different in others. For instance, while surveying Google's employees, the first author found that job enthusiasm affects productivity to a nearly identical degree for both Google's knowledge workers and its software developers; on the other hand, he also found that time management autonomy affected knowledge workers' productivity substantially *more* than it affected software developers' productivity.

In sum, those who want to understand the productivity of software developers should also understand the productivity of knowledge workers, not because the latter can replace the former but instead so they can make informed choices about when existing measures and factors ought to be used and when new measures and factors ought to be invented.

## Summary

While software development has its specific characteristics, there is a lot to learn from studies of general knowledge work. First, it is not sufficient to look at quantity of output but to include the quality of the work as well (see Chapters 4 and 5). Second, it provides approaches to measure productivity besides outcome. Still, knowledge work research has not found a suitable way to capture all important aspects of productivity. Third, it provides a set of drivers for productivity that are directly applicable to software development, such as enough space for solitary work and a pleasant work atmosphere.

## Key Ideas

The following are the key ideas from the chapter:

- Software developers are a specific kind of knowledge worker. Knowledge worker productivity has been studied in a variety of contexts, and those studies can be used to understand software developers.
- There are four main techniques for measuring knowledge worker productivity: outcome-, process-, people-, and multi-oriented productivity measurement techniques.
- There are five categories of drivers that knowledge worker research suggests influence productivity: the physical environment, the virtual environment, the social environment, individual work practices, and well-being at work.

## References

[1] Drucker, P. F. (1959). Landmarks of tomorrow. Harper & Brothers.

[2] Drucker, P. F. (1999). Knowledge-worker productivity: The biggest challenge. California management review, 41(2), 79-94.

[3] Palvalin, M. (2017). How to measure impacts of work environment changes on knowledge work productivity–validation and improvement of the SmartWoW tool. Measuring Business Excellence, 21(2).

[4] Ramírez, Y. W., & Nembhard, D. A. (2004). Measuring knowledge worker productivity: A taxonomy. Journal of intellectual capital, 5(4), 602–628.

[5] Meyer A. N., Fritz T., Murphy G. C., Zimmermann T. (2014). Software developers' perceptions of productivity. SIGSOFT FSE 2014: 19–29.