CHAPTER 4

Defining Productivity in Software Engineering

Stefan Wagner, University of Stuttgart, Germany

Florian Deissenboeck, CQSE GmbH, Germany

Successful software systems are subject to perpetual change as they need to be continuously improved and adapted to continuously changing requirements. *Software evolution* is the term used in software engineering to refer to this process of developing software initially and then repeatedly updating it. It is an essential goal to minimize the cost and to maximize the benefits of software evolution. In addition to financial savings, for many organizations, the time needed to implement software changes largely determines their ability to adapt their business processes to changing market situations and to implement innovative products and services. With the present yet increasing dependency on large-scale software systems, the ability to develop and change existing software in a timely and economical manner is essential for numerous enterprises and organizations in most domains.

We commonly call this *productivity,* which across disciplines and domains refers to the ratio between output and input. The input side—the cost spent—is relatively easy to measure in software development. The challenge lies in finding a reasonable way to define output as it involves software quantity and quality. The software engineering community has so far been unable to develop a thorough understanding of productivity in software evolution and the significance of the factors influencing it, let alone universally valid methods and tools to analyze, measure, compare, and improve productivity. Perhaps the most difficult issues are the many factors that influence

© The Author(s) 2019
C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering,*
https://doi.org/10.1007/978-1-4842-4221-6_4

productivity—and that they are different in every project, which makes it so hard to compare them. What complicates the situation is the lack of an established, clearly defined terminology that serves as a basis for further discussions.

Hence, we see the disambiguation of the terms that are central to productivity as a first important step toward a more mature management of productivity in software engineering. For that, we make use of the existing work from other research areas with a focus on *knowledge work*. We discuss the terms frequently associated with productivity, namely, *efficiency*, *effectiveness*, *performance*, and *profitability*, and explain their mutual dependencies. As a first constructive step, we propose a clear and integrated terminology.

To better put the terminology in the perspective of software engineering, we start with a description of the history of software productivity.

A Short History of Software Productivity

A wide variety of definitions of software development productivity have been discussed for more than four decades. In the beginning, however, this discussion was usually based on anecdotal evidence presented by renowned researchers and practitioners of the field. For example, Brooks stressed in 1975 the importance of people-related factors for software productivity [3], which was more recently followed up on by DeMarco and Lister [4], as well as Glass [5]. First isolated experiments were carried out to investigate productivity variations and its causes as early as 1968 [7, 11].

The late 1970s and early 1980s brought the first attempts to tackle software development productivity in a more comprehensive manner. As measuring productivity requires a well-defined notion of the size of the generated product, considerable effort was spent on the definition of size metrics that do not suffer from limitations of the classic lines of code (LOC) metric. In 1979, Albrecht introduced *function points* to express the amount of functionality of an information system rather than the size of its code. Based on the specification of a system instead of on its implementation, function points were designed to support early development effort estimation and to overcome limitations inherent to the measurement of LOC, e.g., comparability between different languages. Function points provide a basis for productivity measures such as function points per week or work-hours per function point.

In parallel, Boehm developed his cost estimation model COCOMO—now COCOMO II [1]—which is part of the standard software engineering knowledge today. While

not directly based on function points but on LOC, COCOMO addresses development productivity by explicitly including productivity factors such as *required reliability* or the *capability of the analysts*. Boehm also recognized the importance of reuse, a phenomenon unknown in manufacturing, for software productivity and introduced a separate factor that should cover this influence.

The 1980s deepened the understanding of software productivity by significantly enlarging the then poor empirical knowledge base. Most notably, Jones contributed to this through his systematic provision and integration of a large amount of data relevant for productivity analyses. In his books, he discusses various factors for productivity and presents industrial averages for these factors that potentially form a basis for productivity assessments. Nevertheless, one of his insights [6] is that for each project a different set of factors may be most influential.

In the beginnings of the 2000s, several researchers proposed economic-driven or value-based software engineering as an important paradigm in future software engineering research. For example, Boehm and Huang [2] point out that it is not only important to track the costs in a software project but also the real earned value, i.e., the value for the customer. They explain that it is important to develop the software business case and keep it up-to-date. By doing so, they open up a new perspective on software productivity that reaches beyond development costs and explicitly includes the benefits provided for the customer.

During the 2000s and the recent years, agile software development has made a strong impact on many organizations that develop software. One of the core principles of agile development is to create *customer value*. Hence, many aspects of agile development aim to focus on this value generation. One example is the evolution from continuous integration to continuous delivery [13], i.e., to deliver value to customers not at the end of the project or a sprint but continuously. Another aspect related to productivity brought in by agile development was the counting of *story points* and the calculation of *velocity* as the number of story points per sprint. However, many proponents of agile development recommend not to use this measure of velocity as a productivity measure because it can lead to unwanted effects. For example, Jeffreys [15] states, “Velocity is so easy to misuse that one cannot recommend it.” The effects can include that story points are inflated instead of used as a means to identify too large stories and keeping developers from working on stories with a small number of story points. Hence, agile software development has no clear definition of *productivity* or a solution for measuring productivity.

Terminology in the General Literature

Our starting point is Tangen's [12] Triple-P-Model, which is a well-established model in knowledge work research to differentiate *productivity*, *profitability*, and *performance* as well as the programming productivity Wikipedia article (https://en.wikipedia.org/wiki/Programming_productivity). Especially in software engineering, *efficiency* is used instead of *productivity*; we also discuss it and differentiate it from *effectiveness*. Finally, following Drucker [8], we include a short discussion on the influence of quality on productivity. We discuss each of these terms separately in the following sections and will integrate them afterward.

Productivity

While there is no commonly agreed on definition of *productivity*, there appears to be consensus that productivity describes the ratio between output and input.

Productivity = Output / Input

Across the various disciplines, however, different notions and different measurement units for input and output can be found. The manufacturing industry uses a straightforward relation between the number of units produced per time unit and the number of units consumed in production. Nonmanufacturing industries use person-hours or similar units to enable comparison between outputs and inputs.

As long as classical production processes are considered, a metric of productivity is straightforward: how many units of a product of specified quality are produced at which costs? For intellectual work, productivity is much trickier. How do we measure the productivity of authors, scientists, or engineers? Because of the rising importance of "knowledge work" (as opposed to manual work; see also "What We Can Learn from Productivity Research About Knowledge Workers" [8]), many researchers have attempted to develop productivity measurement means that can be applied in a nonmanufacturing context. It is commonly agreed on that the nature of knowledge work fundamentally differs from manual work and, hence, factors besides the simple output/input ratio need to be taken into account, e.g., quality, timeliness, autonomy, project success, customer satisfaction, and innovation. However, the research communities in neither discipline have been able to establish broadly applicable and accepted means for productivity measurement yet [9].

Profitability

Profitability and productivity are closely linked and are, in fact, often confused. However, *profitability* is most often defined as the ratio between revenue and cost.

Profitability = Revenue / Cost

The number of factors that influence profitability is even greater than the number of factors that influence productivity. Particularly, profitability can change without any change to productivity, e.g., due to external conditions such as cost or price inflation.

Performance

The term *performance* is even broader than productivity and profitability and covers a plethora of factors that influence a company's success. Hence, well-known performance control instruments such as the Balanced Scorecard [14] do include productivity as a factor that is central but not unique. Other relevant factors are, for example, the customers' or stakeholders' perception of the company.

Efficiency and Effectiveness

Efficiency and *effectiveness* are terms that provide further confusion as they are often mixed up themselves; additionally, efficiency is often confused with productivity. The difference between efficiency and effectiveness is usually explained informally as "efficiency is doing things right" and "effectiveness is doing the right things." While there are numerous other definitions [12], an agreement prevails that *efficiency* refers to the utilization of resources and mainly influences the required input of the productivity ratio. *Effectiveness* mainly aims at the usefulness and appropriateness of the output as it has direct consequences for the customer.

Influence of Quality

Drucker [8] stresses the importance of quality for the evaluation of knowledge worker productivity. Productivity of knowledge work therefore has to aim first at obtaining quality—and not minimum quality but optimum if not maximum quality. Only then can one ask, "What is the volume, the quantity of work?" However, most of the literature in nonsoftware disciplines does not explicitly discuss the role of quality in the output of the productivity ratio [8]. More recent work from nonmanufacturing disciplines have

a stronger focus on knowledge, office, or white-collar work and hence increasingly discuss the role of quality with respect to productivity [4, 9, 10]. Still, it appears that these efforts to include quality in the determination of productivity have not yet led to an operationalizable concept.

An Integrated Definition of Software Productivity

As discussed, for measuring software productivity we need a measurement of input and output of a software project. The input is the effort dedicated to its development and evolution. The output is the value of the software for its users or customers. The value cannot always be defined by the market value of the software as it is often developed and used internally by organizations and as such does not have a market value. Furthermore, the market value may be influenced by factors that we put to the level of profitability or performance, such as currency valuations or competition on the market.

Hence, we suggest a purpose-based definition of software value. Given a purpose (a business goal or an application vision), we ask, how well does the software address its purpose in terms of functional and nonfunctional requirements? The answer to this question is determined by the functionality as well as the nonfunctional quality of the software.

On the basis of the purpose-based view, we build a consolidated summary of the productivity-related terms. As shown in Figure 4-1, from the purpose, we derive an ideal functionality and quality as well as the ideal effort to serve the purpose correctly. The ideal functionality means the optimal set of features (nothing missing, nothing too much) to fulfil the purpose. Similarly, the ideal quality is the level of the various quality attributes that fit to the purpose in an optimal way. For example, the application scales easily to the needed number of parallel users but not beyond. The ideal effort denotes the number of person-hours if people trained well for the problems to be solved (i.e., the ideal functionality and quality) would have worked in a supportive environment on the software. Comparing the ideal with the actually produced functionality and quality shows the effectiveness of the software development activities; the relation of the ideal to the actual effort gives the efficiency. Both have an influence on productivity.

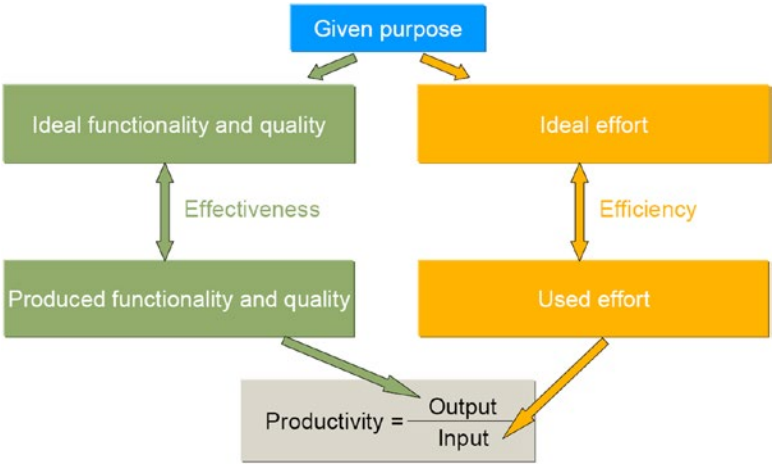

Figure 4-1. *Purpose-based effectiveness and efficiency*

We embed this in the Triple-P-Model from Tangen [12] so that it results in the *PE Model* that illustrates how purpose, functionality, quality, and effort relate to effectiveness, efficiency, productivity, profitability, and performance (Figure 4-2). The original Triple-P-Model already provided the idea that profitability contains productivity but adds further factors such as inflation and pricing. In turn, performance contains profitability and adds factors such as customer perception.

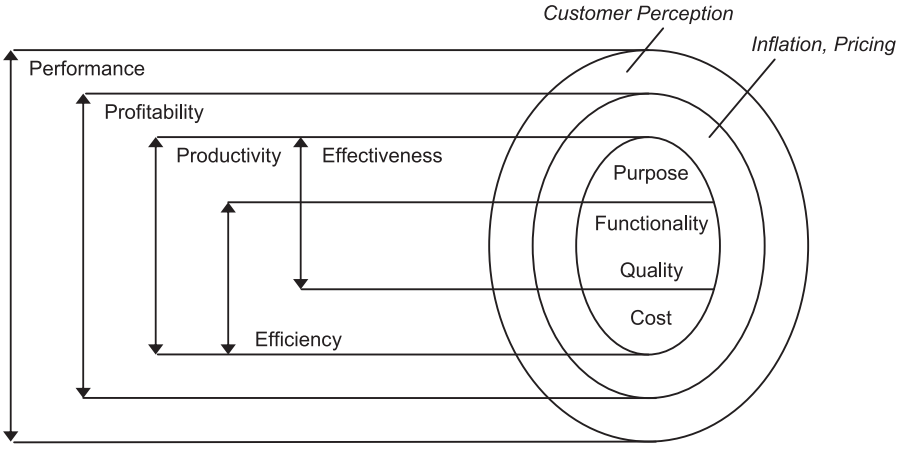

Figure 4-2. *PE Model for software evolution productivity*

We add in the PE Model that productivity is expressed as the combination of effectiveness and efficiency: a team can be productive only if it is effective and efficient! We would neither consider a software team productive if it was not building the features needed by the customers nor if it spent an unnecessary amount of effort on building the software. For effectiveness, we need to consider the purpose, functionality, and quality of the software. For efficiency, we further consider costs. Hence, the PE Model allows us to set all terms discussed earlier in this chapter into relation with each other.

Summary

There is still a lot of work to do until we can have a clear understanding of productivity in software engineering. The complexity of capturing *good* knowledge work is an obstacle in general to unambiguously measuring the productivity of such work. We hope that at least our classification of the relevant terms and the resulting PE Model can help to avoid confusion and to focus further efforts.

Our discussion of the related terms complements the productivity framework in Chapter 5. The framework focuses on the three dimensions of *velocity*, *quality*, and *satisfaction*. While quality is covered in both chapters, we have not incorporated velocity. Velocity can be different from effort as it concentrates on how fast features are delivered to customers. Being faster might actually need more effort. We also have not integrated work satisfaction explicitly as it was not part of the Triple-P-Model. This is surprising as—in hindsight—we would expect that to play a big role in knowledge work in general. Therefore, we believe that a combination of our PE Model and the productivity framework in Chapter 5 will clarify terms and cover the most important dimensions.

In Chapter 7, you can read about research on knowledge work as well as how (not) to measure productivity.

Key Ideas

This chapter covers the following key ideas:

- A clear terminology is important for further discussions on productivity factors and productivity measurement.
- We should reflect on the history of productivity research in software engineering.

- We need to learn from research on knowledge work productivity and use compatible terms.
- The purpose of the software is the necessary basis for all definitions of productivity and related terms.

Acknowledgements

We are grateful to Manfred Broy for fruitful discussions on definitions of productivity in software engineering.

References

[1] Boehm, B. et al. Software Cost Estimation with COCOMO II, 2000

[2] Boehm, B. and Huang, L. Value-Based Software Engineering: A Case Study. IEEE Software, 2003

[3] Brooks, F. P. The mythical man-month. Addison-Wesley, 1975

[4] DeMarco, T. and Lister, T. Peopleware: Productive Projects and Teams. B&T, 1987

[5] Glass, R. L. Facts and Fallacies of Software Engineering. Addison-Wesley, 2002

[6] Jones, C. Software Assessments, Benchmarks, and Best Practices. Addison-Wesley, 2000

[7] Sackman, H.; Erikson, W. J. and Grant, E. E. Exploratory experimental studies comparing online and offline programming performance, Commun. ACM, ACM, 1968, 11, 3–11

[8] Drucker, P. F. Knowledge-Worker Productivity: The Biggest Challenge. California Management Review, 1999, 41, 79-94

[9] Ramírez, Y. W. and Nembhard, D. A. Measuring knowledge worker productivity: A taxonomy. Journal of Intellectual Capital, 2004, 5, 602–628

[10] Ray, P. and Sahu, S. The Measurement and Evaluation of White-collar Productivity.

International Journal of Operations & Production Management, 1989, 9, 28–47

[11] Sackman, H.; Erikson, W. J. and Grant, E. E. Exploratory experimental studies comparing online and offline programming performance, Commun. ACM, ACM, 1968, 11, 3–11

[12] Tangen, S.; Demystifying productivity and performance. International Journal of Productivity and Performance, 2005, 54, 34–36

[13] Jez Humble, David Farley. Continuous Delivery. Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley, 2010.

[14] Robert S. Kaplan, David P. Norton: The Balanced Scorecard – Measures that Drive Performance. In: Harvard Business Review. (January–February), 1992, S. 71–79.

[15] Ron Jeffries. Should Scrum die in a fire? https://ronjeffries.com/articles/2015-02-20-giles/

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.