Personal Reading: 10x

Mats Stijlaart - University of Amsterdam, The Netherlands

March 13, 2015

What Is 10x?

10x is one of the oldest and most researched topics in software engineering where measurements are applied to identify if a 10-fold productivity holds between different programmers [5, p. 567]. These measurements are in place to show if programmers with the same level of experience have higher productivity then others. The '10x' is more a figure of speech than an actual multiplication of performance. It is mainly used to show a significant difference in the order-of-magnitude between two objects. As expressed by Bossavit, "the 10x claims has been around forever" [2, p. 37]. In this expression, as Bossavit mentions himself, one should read 'forever' as "since the beginning of the software engineering as an academic discipline" [2, pp. 38-39].

In my opinion it is not strange that this topic was one of the first to be studied, while the business always strives to increase productivity and managers want to know where the money goes. DeMarco and Lister describe the measurement of individual productivity as a point of friction [3, p. 268]. They state that "the presumption that there are order-of-magnitude differences in individual performance makes cost projection seem nearly impossible" [3, p. 268]. As a variation on the '10x' research, DeMarco and Lister focused on the performance of workplaces rather than the performance of individuals. More on this in the section *Is It Effective?*

What Do We Know?

In 1968 Sackman and Erikson published an article on the difference in performance for programmers performing different programming tasks [6]. The main findings in this article are [6][5, p. 567]:

- The ratio in coding time between the best and the worst programmer was 1 to 20;
- The ratio in debugging time between the best and the worst programmer was 1 to 25;
- The ratio of the size of the program between the best and the worst programmer was 1 to 5.

This paper triggered a lot in the software engineering community. And as Bossavit shows, there were multiple reproductions on the original experiment [2, pp. 38,41]. Most of these studies could not reproduce Sackman's and Erikson's original results [2, pp. 42-44]. The other studies had inconclusive results.

As McConnel summarizes nicely in his chapter on '10x' in *Making Software* [5]: "The general finding that 'There are order-of-magnitude differences among programmers' has been confirmed by many other studies of professional programmers" [5, p. 568]. The studies that McConnel cites show that there is this difference in magnitude, but these results are different in each study. So there are differences, but the question remains what these differences mean.

Another important finding of these studies is that the results that Sackman and Erikson published were actually incorrect. Dickey showed that Sackman and Erikson did not make a clear classification on the environment of their test subjects [4, p. 844]. For example, they let two subjects program in different programming environments. Dickey showed that when the correct classification was used, the actual debugging was not 1 to 25, but only 1 to 5.

Measurement

Augustine observed that more than 50% of the work is typically done by 20% of the people [1][3, p. 268]. This seems a promising observation, but there is more to it. For example, it is easy to measure performance differences in 100m sprint athletes: we can say that an athlete runs 0.8 seconds faster, or has as velocity that is 5% above average. But Augustine's observation is more doubtful on soccer teams, where a striker scores the goals and a goalkeeper does not. In these kind of settings, where different people play different roles, it is hard to compare performance on easy to define measurements.

The different types of measurements return in the presented studies over the last years. Bossavit classifies the used measurements in a clear overview in his book [2, pp. 42-43] and it is easy to include an argument to show that the measurement might not be sufficient. For example:

- Time to complete: The time to complete measurement should indicate that better programmers will finish their work faster, but better programmers may also address more (error) cases of which a less performing programmer may not think. Due to his experience, a better programmer may thus need more time, but will end with better software.
- LOC per staff per hour: With this measurement the researchers try to show that better performing programmers write more code per hour, although in a working environment the better performing developers may be pulled out of their context more often to answer questions, make decisions or help colleagues. This may affect the amount of code the developers actually write, whilst they actually have high value for the team and/or organization. Another counter argument is that better programmers may come up with better solutions in the end that require less code.

Another arguable difference is that the researchers use test subjects with equal amount of ‘experience‘ usually expressed in their years active in the industry. But given a programming task, a Java database expert might be less likely to solve it than a Java webservice programmer. Classifying people by their years active in the industry leads to comparing apples with oranges. In my opinion a stronger argument can be made when comparing two people with the same years of experience:

A developer married with children, who is inactive in the evenings, might perform differently than a developer living in his parents' basement and programming into the wee hours - even if they have the same amount of years of experience It is not acceptable to say two individuals are equal based on the active years in the industry while they have completely different lives or expertises.

Assume we can eliminate these arguments, and let the programmers participating in the experiment sit in an isolated workplace and get the task to finish a problem as fast as possible. How effective will this be? It may show that some developers perform better under time pressure (write the feasible code within the timespan) or write a short solution (for example using some obscure constructs in some languages), but these kind of setups are far from realistic compared to the industry, when aspects such as quality may play an important role. The point to be made is: programmers may be more ‘productive’ in a research setup, but are they actually as effective in the industry?

Is It Effective?

An important problem is that different programmers must work in different environments, which makes it almost impossible to apply the same measurements on each individual. So if we apply ‘10x’ measurements in the industry, will it be effective?

As McConnel shows in *Making Software*, the measurement must align with the goal [5, pp. 570-571]. In which he shows that this is hard and presents the same kind of arguments as I have presented in the previous section *Measurement*. None of the research I read on this topic provides a conclusive metric that will actually fit the industry.

But assume a manager in a software company wants to measure how his programmers are performing and he applies one of the inconclusive measurements. What may happen? I believe it will do more harm than good. For example, it is really easy to cheat on a LOC metric by splitting statements on different lines, or picking easy tasks of which a programmer knows he can write a lot of code instead of the hard to find one-line bugs. What will happen with the good programmers that can actually fix these one-line bugs? They perform well in their team, but the applied measurement does not project their actual value. These programmers might start to feel under-appreciated and stop performing well or even move to different companies.

DeMarco and Lister state that high and low performers tend to cluster in different organizations [3, p. 268]. But I question if the high performers are not just moving away from the low performant ‘slackers’ as McConel puts it [5, p. 569]. DeMarco and Lister performed a different kind of experiment than the other comparable research papers did. Instead of looking for performance of individuals, they looked at the performance of programmers related to their work environment. One of what I find the most interesting things I have read on this topic are their results on the effects of the environment on performance of individuals [3, p. 271]. They present significant differences in the environment of higher performing (programming speed in combination with amount of defects) programmers. Such as:

- Having more floor space;
- Acceptable quiet/private workplace;
- Not forced to pick up phone calls;
- Less needless interrupts;
- Feeling more appreciated at your workplace.

None of the above points relate to the actual programming. They relate to how people feel or how happy they are in their job. Therefore, I do believe the research to identify programmers with higher performance is beneficial. For example, companies can benefit in their hiring procedure by applying productivity measurements [1]. However, I also believe it will have the opposite effect when applied continuously in an organization.

Is It Real?

Overall I believe there are programmers that have a significant higher contribution for the project or the company for which they work. This is due to a gut feeling, but also from my own experience in the field that developers do not know what to do when one employee is sick or has holidays. This one developer is key and identified as the high performing employee.

[1]Of course companies must use this as an addition on the original process.

McConnel tries to put in a good argument to show that it is real, but I still find his argument too weak. He presents the results of the written lines of code between Microsoft Excel 3.0 and Lotus 123 and highlights that the Microsoft team had a higher velocity [5, p. 572]. But we still do not know the context of these projects, did Microsoft copy big parts of their Excel 2.0 project? Or did they have a really good architect in place that allowed the developers to be more productive? I find it too bold to compare two different companies with each other, even though they developed the same kind of application at the same moment.

Conclusion

After reading the literature on this topic, for me it is still unclear if we can measure performance differences between individuals, and how to express it (lines of code, number of resolved tickets or number of defects). I strongly believe that organizations should not measure the performance of different entities (teams or individuals), but should focus on the performance of these different entities over time. For example, will these entities have a higher velocity when they stop taking phone calls? And just maybe organizations and managers must focus on teams: focus on putting the right knowledge and characters together to create a high performing team.

References

[1] N.R. Augustine. "Augustine's Laws and Major System Development Programs". In: *Defense Systems Management Review* (1979), pp. 50–76.

[2] Laurent Bossavit. *Leprechauns of Software Engineering: How Folk Law Turns Into Fact [Electronic Resource].* Lean Publishing, 2013.

[3] Tom DeMarco and Tim Lister. "Programmer performance and the effects of the workplace". In: *Proceedings of the 8th international conference on Software engineering.* IEEE Computer Society Press. 1985, pp. 268–272.

[4] Thomas E Dickey. "Programmer variability". In: *Proceedings of the IEEE* 69.7 (1981), pp. 844–845.

[5] Steve McConnell. "What Does 10x Mean? Measuring Variations in Programmer Productivity". In: *Making Software: What Really Works, and Why We Believe It.* 1st ed. O'Reilly, 2011. Chap. 30, pp. 567–573.

[6] Harold Sackman, Warren J Erikson, and E Eugene Grant. "Exploratory experimental studies comparing online and offline programming performance". In: *Communications of the ACM* 11.1 (1968), pp. 3–11.