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 verify if programmers have an higher productivity when having the same level of experience. The '10x' is more a figure of speech than an actual multiplication of performance. It is mainly used to show a significant difference in 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, you should read 'forever' as since the beginning of the software engineering as an academic discipline [2, pp. 38-39].

In my opinion this is not strange, while in business always strives to increase productivity, managers want to know where the money goes. DeMarco and Lister state 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 Real?*.

What Do We Know?

In 1968 Sackman and Erikson published an article on the difference in performance 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 worst programmer was 1 to 20;
- The ratio in debugging time between the best and the worst programmer was 1 to 25;
- And 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]. The main part of these studies could not reproduce the original results from Sackman and Erikson [2, pp. 42-44]. The other part of the studies had inconclusive results.

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

An 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, letting two subjects program in different programming environment. Dickey showed that when the correct classifications were used, the actual debugging time for example 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 none. 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 arguments 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 that a less 'performing' programmer may not think about. Due to his experience, better programmer may thus need more time, but will end with better software.
- LOC per staff per hour: With this measurement the researchers tend to show that better performing programmers write more code per hour, although in a working environment the high performing developers may be pulled out of their context more often to answer question, make decision 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 arguable difference is that the researchers use test subjects with equal amount of 'experience' usually expressed as years active in the industry. But giving a programming task, maybe a Java database expert in Java will be less likely to solve it as a Java webservice programmer. Years active in the industry still makes us compare apples and oranges. In my opinion a stronger argument is comparing two persons with the same years of experience: One developer that is married and has two children doing nothing in the evening, or another developer living in his/her parents basement programming until the wee hours. It is not representable to compare these two persons when the point of view is experience of programmers.

Assume we can eliminate these arguments, and let the programmers that do the experiment sit on a 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 the 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 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?

One problem that exists is that different programmers must work in different environments, which makes it almost impossible to apply the same measurements on each individual.

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*. All the research I have read on this topic, does not present an conclusive metric that will actually fit the industry.

But assume a manager in a software company does wants to measure how his programmers are performing and he applies one of the inconclusive measurements. What may happen? I believe this will kick the manager in the face. 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 states 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 performants 'slackers' as McConel puts it [5, p. 569]. DeMarco and Lister performed a different kind of experiment than others did. Instead of looking for performance of individuals, they looked at the performance of programmers related to their working environment. One of what I find 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 moor floor space;
- Acceptable quiet/private workplace;
- Not forced to pick up phone calls;
- Less needlessly 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. Therefor I do not believe the research to identify programmers with higher performance is beneficial and companies can benefit this by using it in hiring policies. Although I believe it will have a counter effect when applied continuously in an organization.

Is it Real?

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

McConnel tries to put in a good argument to show that it is real, but I still find his argument is too weak. He presents the results in 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 for this topic, for me it is still unclear what performance difference between individuals is, such as 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.