Measuring Software Engineering

Franklin Ume Obiekwe: 18320505

Introduction

If you were the CEO of a software development firm, how would you go about measuring the amount of software engineering being carried out? Would you first ask yourself why would it be beneficial to measure the amount of productivity from your engineers? You don't need a business degree to answer this question. If you had the means to measure the productivity of your engineers, you could identify the unproductive and find a replacement. This would mean more profits in the long run as increased productivity in labour means shorter turnaround times for projects and higher quality projects. Now back to the original question. How does one identify the productive from the unproductive? If I asked you to do a quick google search to tell me what the industry-standard method is, you would return to me with conflicting results. This is due to the fact that measuring software engineering is a way more complex task then the question would lead you to believe.

In this report, I will discuss why measuring software engineering is a complex task, and explore the different methods that have been used in the past and how they have improved and evolved into the techniques and technologies we see used today.

Can You Really Measure Individual Developer Productivity

“Currently there are no universally accepted methods to measure knowledge worker productivity, or even generally accepted categories”(Ramírez, Y.W., 2004). For example, how do you measure the productivity of a doctor or a lawyer? If we try to simplify our problem statement we get a little closer to knowing if it’s possible to measure software engineering. If we were to measure the productivity of a bricklayer, how would we go about it? Well, a bricklayer has a fixed output. We can count the number of bricks he lays on a specific day or project. If we try to adapt this to a software engineer we can indeed identify different outputs. Some of these one-dimensional outputs that can be measured as outlined by Orosz, G (2020), include: lines of code, commits, and tickets closed.

Source lines of code

Source lines of code (SLOC), also known as lines of code (LOC), is one way of measuring the output of a software engineer. It is a software metric used to measure the number of lines in the text of the program's source code. As any software engineer would tell you, SLOC is a flawed system as it is one that can easily be gamed. If we go back to our bricklayer example, how many different ways are there to build a 6-foot by 6-foot wall? There are probably 2 or 3 ways but the wall will nearly always have the exact same number of bricks no matter what way they are positioned. Now if I ask 3 software engineers to write a program that prints the numbers from 1 to 5, they may provide the following results:

Programmer 1:

```
for (let i = 0; i < 5; i++)
{
    console.log(i)
}
```

Programmer 2:

```
for (let i = 0; i < 5; i++) console.log(i)
```

Programmer 3:

```
console.log(1)
console.log(2)
console.log(3)
console.log(4)
console.log(5)
```

Using SLOC, we can see that programmer 3 is five times more productive than programmer 2 and Programmer 1 is four times as productive as Programmer 2. This is the drawback of using SLOC. If a CEO were to implement SLOC as their method of measuring software engineering we would see more people developing code like programmers 1 and 3 in the environment and the CEO could see an artificial increase in productivity of 300% in the company. One way to limit the drawbacks of SLOC would be to remove the one-dimensional aspect of analyzing it in a vacuum. To do this the CEO could implement better-reinforced coding standards. This would eliminate people like Programmer 3 from polluting the code base with terrible code. This still shows a disparity in Programmer 1 and 2. We could upgrade from using standard SLOC and use logical SLOC as outlined by Nguyen, V.(2007). Logical SLOC is basically counting logical code statements as lines rather than every line. In our short example, if we use Logical SLOC, both programmer 1 and 2 would have a SLOC of 2. As Nguyen, V. (2007) says, “This method is less sensitive to format and programming styles, but its imprecise definition has been the source of contention among tool developers. ” As programming languages get more advanced and abstracted, and the development of new modules, frameworks and packages are ever-increasing (DeNisco, A, 2018), it’s becoming more difficult to develop a tool with universal standards that can be defined as an accurate measure of software engineering.

Commits

Committing code is how a developer adds their work to a codebase. With that understanding, using commits may be a reasonable way of measuring productivity. If programmer A commits 12 times a day and Programmer B commits twice a day, then you may be lead to believe that programmer B may be the more productive worker. This

is not necessarily the case though. “Commits are arbitrary changes captured in a single moment in time. Their size and frequency do not correlate with the work needed to achieve that change. At best, commits can be viewed as an indication of activity” (Van der Voort, J. 2016). If a CEO were to go down the route of using commits a day as a metric in which to measure productivity, he would see an artificial productivity increase tenfold to what it was before the metric was introduced. Software engineers would just be committing smaller chunks of code. If anything, productivity would decrease in reality as more time is being dedicated to committing code and documenting those commits than usual. This means less time programming or thinking.

Tickets Closed

Committing code is how a developer adds their work to a codebase. With that understanding, using commits may be a reasonable way of measuring productivity. If programmer A

Team Velocity and Agile Process Metrics

https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams

Trello boards. The tasks to do this week compared to last. The difficulty of each task

Third-Party Technology

Chair or that dice thing. Screen capture, tab watcher etc

Ethics

Chair or that dice thing. Screen capture, tab watcher etc

Bibliography

DeNisco, A. (2018). "*How programming will change over the next 10 years: 5 predictions*" Tech Republic, https://www.techrepublic.com/article/how-programming-will-change-over-the-next-10-years-5-predictions/ [Accessed 13 November.]

Nguyen, V., Deeds-Rubin, S., Tan, T. and Boehm, B., (2007). "*A SLOC counting standard*". In Cocomo ii forum (Vol. 2007, pp. 1-16). Citeseer.

Orosz, G. (2020). *"Can You Really Measure Individual Developer Productivity? - Ask the EM."* Pragmatic Engineer https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/. [Accessed 13 November.]

Ramírez, Y.W. and Nembhard, D.A. (2004), "*Measuring knowledge worker productivity: A taxonomy"*, Journal of Intellectual Capital, Vol. 5 No. 4, pp. 602-628. https://doi.org/10.1108/14691930410567040

Van der Voort, J. (2016). "*Commits Do Not Equal Productivity*" GitLab, https://about.gitlab.com/blog/2016/03/08/commits-do-not-equal-productivity/ [Accessed 13 November.]