

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.

Firstly we must define what a software metric is, "A software metric stands for a potential area where measurement can be effectively applied to a certain software module or its specifications" (Info Pulse, 2018). In other words, a metric is taking some data from your product development lifecycle and using that data to measure productivity.

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

challenging 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

Another way to measure software engineering is by counting the number of tickets closed by a developer over a period of time. Tickets are like all the tasks that need to be done for a project. If a developer closes 6 tickets in a week then he has completed 6 tasks in that week. Could a CEO effectively measure productivity through closed tickets though? Again without context, this metric could easily be gamed. Once the CEO would introduce this metric. Software engineers would just cleverly split tasks into smaller

more straight forward tasks which on paper still make sense when phrased correctly and the CEO could potentially see an artificial increase in productivity.

By building on this method though, we could potentially see a reasonably efficient method. “If the tasks are written properly, and they are assigned in terms of business priority, then measuring closed tickets may be a metric for you to consider when measuring developer productivity” (Asahara, A, 2020). This is the first step in improving the tickets closed metric. If every ticket now has a valuable meaning, we know that valuable work is being done. This still doesn’t address the issue of someone doing 1 long and difficult task and someone doing all the handy straight forward ones. “Because of this, some teams score tickets by the amount of effort required. In this way, you may be able to get a better gauge of developer productivity.” This is a more refined method of judging productivity but has been abstracted out to a point where it isn’t a one-dimensional metric but a glimpse into the topic of measuring with context.

Measuring with Context

As outlined by Lowe (2016), “metrics don't matter in software development” “unless you pair them with business goals”. This is what I mean by adding context. In the software engineering industry, no developer works by themselves on a project, (bar the odd project). Most people work in a team and that needs to be taken into consideration when measuring the productivity of an individual. “Metrics like closed tickets, completed code reviews, and even conversations with other employees can give you a much fuller look at the developer productivity on your team” (Asahara, A, 2020). So how do we measure the productivity of a team as a whole? It depends on the workflow process the team is operating under, but in this day and age, it is safe to assume that most teams are operating under the agile structure. “For agile and lean processes, the basic metrics are lead time, cycle time, team velocity, and open/close rates” (Lowe, S, 2016).

Lead Time

The time it takes a team to go from an idea to a finished product is an important metric when it comes to measuring software development. In the short term, it may be a hard metric to measure as it implies the team has completed a project before. Nonetheless, it is a vital metric when looked at in the scope of achieving a business goal. In today's environment, it doesn't matter how much code you can write and commit in a period of time if you can't effectively bring a product to completion with a team. "If you want to be more responsive to your customers, work to reduce your lead time, typically by simplifying decision-making and reducing wait time." (Lowe, S, 2016).

Cycle Time

With agile development, team meetings are held often, this helps to accommodate clients who may want to integrate feedback into the previous deliverable. Cycle time is how long it takes to make a change to a software system and also how long it takes for those changes to make it to production.

Team Velocity

Team velocity is one of the most important metrics when tackling how to measure software engineering. In simple terms, "It is a measure of the team's progress rate" (Coelho, E. and Basu, A., 2012). It's known as velocity as it follows the formula: (Ziauddin, S.K.T. and Zia, S., 2012.)

$$Vi = \text{Units of Effort Completed} / \text{Sprint Time}$$

Team velocity should mainly be used for estimating and planning future sprints and iterations.

Open/close rates

We have covered the creation of valuable tickets and assigning each ticket a score based on the amount of effort it is estimated to take. Tracking those tickets is a great way to measure software engineering. Aside from tracking the tickets that are being closed, we can also measure the productivity of a team based on the rate at which they both open and close tickets/issues. Analyzing these trends can help discover where productivity might fall in the future by analyzing the rates of the past

To summarise this section on measuring with context, the best way to measure how efficient your software development is to analyse how quickly your software reaches business goals and improves business results.

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

Asahara, A. (2020). *"10 Tips to Measure Developer Productivity"* Sider Team Insights,
<https://www.sleeeek.io/blog/10-tips-to-measure-developer-productivity#:~:text=1.,productivity%20is%20through%20closed%20tickets.&text=But%2C%20if%20the>

%20tasks%20are,consider%20when%20measuring%20developer%20productivity
.
[Accessed 13 November.]

Coelho, E. and Basu, A., (2012). "*Effort estimation in agile software development using story points*". International Journal of Applied Information Systems (IJ AIS), 3(7).

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.]

Info Pulse. (2018). "*Top 10 Software Development Metrics To Measure Productivity*" Info
Pulse,
<https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/> [Accessed 13 November.]

Lowe, S. (2016). "*9 metrics that can make a difference to today's software development teams*", Tech Beacon,
<https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams> [Accessed 13 November.]

Lowe, S. (2016). "*Why metrics don't matter in software development (unless you pair them with business goals)*", Tech Beacon,
<https://techbeacon.com/app-dev-testing/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals> [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.]

Ziauddin, S.K.T. and Zia, S., (2012). "*An effort estimation model for agile software development*". Advances in computer science and its applications (ACSA), 2(1), pp.314-324.