Measure Software Engineering Report

Akinfolarin Fifo

18323551

Junior Sophister

Integrated Computer Science

***How can Software Engineering be measured?***

Before we can begin to understand whether or not software engineering can be measured, we need to properly define what software engineering means and what it entails to be a software engineering. Software engineering can be defined as the systematic application of engineering techniques to the development of software. Software engineering isn’t just about writing code to solve problems, it’s about writing code is usually just a small piece of a much bigger project. Software engineering entails not only the creativity to develop solutions that fit consumer needs but also requires the patience and perseverance to test code millions of times and fix bugs as they as well as maintain and adapt the code to suit the ever changing needs and wants of the consumer. Software engineering is also about working it team to solve these complex issues and being able to clearly recognize one’s role in a software engineering team and fulfilling one’s role to the maximum output so as to benefit the project at hand. But how does one actually measure the output of a software engineer? It isn’t like say, an accounting job where productivity could be measured by how many clients and dealt with and how their overall satisfaction with the firm. It can’t be measured like the work of a criminal defense attorney who can be easily judged by the cases they’ve won. A car salesman’s productivity is measured simply by totaling up the number of cars he sells in a certain time period and comparing that to his peers. Software engineering, however, isn’t as simple.

Software engineers write lines of code, thousands and thousands and thousands of lines of code. As a budding software engineer myself in 3rd year of study with no prior coding experience before university, I estimate that I’ve probably written about ten thousand lines of code already. But if I have written that much code, does that make me more productive than an experienced software engineer who might write half as much code in that same time frame? The answer is a simple no, because analyzing the productivity of a Software engineer is a bit more difficult than simply counting lines of code. Because while software engineers do write lines of codes, they also debug code over and over again as well as going back to fix errors in the code. These tasks are some of the most important undertaken by software engineers and they don’t necessarily add to the amount of code they’ve written. Furthermore, the best software engineers will use a significantly lower number of lines of code to solve a problem as an inexperienced coder, who will have more lines of code and spend longer on the problem but at the end of the day both engineers completed the same problem, but the experienced engineer used less code which is easier to maintain and debug. Therefore the experienced engineer is more productive than the inexperienced one but if you just counted up the number of lines of code and used that as a barometer for productivity, you wouldn’t come to that conclusion.

All this means is that we have to look at a different approach to measuring software engineering. There are some simple methods of methods of measuring software engineers such as counting lines of code as I’ve already discussed. Another one is the counting the number of commits to version control per day, but again this a metric that can easily be gamed. Engineers are smart people and it’s obvious that counting commits would lead to an increase in the amount of commits but these commits will a lot smaller, therefore not really increasing productivity at all. Or maybe you could measure the number of tickets handled in a day but again this would lead to engineers favoring easier tickets so as to portray an increase in productivity. Another reason that none of these single data type metrics will truly work is that in the real world, software engineering isn’t about individuals, it’s about the collective.

The team dynamic is what software engineering is built on. Teams create good software that will satisfy the consumer and its rare, especially in the context of measuring software engineering, a practice likely to be taken up my tech corporations who want to measure the productivity of their software development teams, that individuals are able to achieve this on their own. A team of very experienced and talented engineers who perhaps have poor team dynamics or a project that wasn’t right for them is likely to not perform as well as team of average developers who have the correct synergy in the team as well as a project that suits them. So the second team is more productive but is it because of their own doing or the team dynamic that perhaps a company placed them in. Another important factor in the success or failure of a software engineering team is the manager or lack thereof. A software team of six developers shouldn’t perform as well as team of five developers and one manager because the manager plays a crucial role in the motivation of the team and making sure all the engineers stay on task. They have to be good communicators as well as being results oriented and have good decision making. So yet again you can see that the productivity of a software engineer can’t be easily measured because of caveats like these. If there was a simple and defined way to measure the productivity of an engineer then everyone would employ it but there simply is not.

So how can we actually measure software engineers in a way that encapsulates all the factors that might affect the data. Because yes, even though I’ve explained that single metric won’t get you anywhere, data is the only way to really measure a coders productivity. It’s about looking at a wide array of data and always applying context to this data. For example, if one developer on a team was taking twice as long to merge their commits and they get twice as many code review comments as others in the team then one could perhaps question the productivity. Applying context may reveal however that the engineer in question just recently joined the team and is still getting up to speed with things. These metrics could also be used to identify top performers in the team. Another example could be an engineer whose number of commits and code reviews has dropped drastically, this could be a cause for concern but could also be an experienced engineer helping out a newbie and this helps the overall team productivity but might not reflect directly on that individual’s metrics.

***What platforms can be used to gather and process data?***

Before we look at the platforms that that are used to measure the data and process this data into something actionable for corporations or lone engineers themselves, we need to look into how where this data comes from in the first place. Version control is one of the most important aspects of being a software engineer, so much so that as a rookie coder, you will likely be introduced to some form of version control in your first year of study. Version control is defined as a class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information. I remember in my first year of university, we had a group project and were told to use TortoiseSVN, which is a version control tool based on Subversion. Subversion used to be the major player in the version control world but since the emergence of Git, its influenced has dwindled. So let’s talk about the Git, the most popular and universally used type of version control. Git is an open-source version control system for tracking changes in any set of files, designed specifically for, but not limited to, coordinating work among programmers cooperating on source code during software development.

Two of the major platforms that implement the Git version control are Bitbucket and the world-renowned GitHub. GitHub is a subsidiary of Microsoft which provides hosting for software development and version control using Git. The basic services that GitHub offers are free of charge and the free accounts are often used to host open source projects. They also offer commercial services for large tech firms to use. These firms monitor their employees software development progress using GitHub and there are many other companies solely created to extract the data from GitHub to then use this data to ascertain the productivity of software engineers. Bitbucket is another platform widely used by software engineers for version control. It is written in python and the main difference between the two is that Bitbucket is seen as more private and used by private companies while GitHub retains a huge open source community. A lot of users, worried about keeping code private, switched to Bitbucket after the controversial acquisition of GitHub by Microsoft in 2018. Both of these version control platforms host millions of repositories and are used by millions each day. There is also extensive data about the work software engineers do to be collected and I will now talk about some platforms designed precisely to do that.

**Velocity**

Velocity is a tool designed by the company Code Climate that analyses all the data from GitHub repositories and provides clients with heads-up displays, real-time analytics, and custom reports to give them a clearer perspective on how their engineering team is working. Velocity looks at the data from GitHub such as