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 number of commits, average commit size, number and frequency of code reviews, time to merge pull requests to the main branch etc. None of these are particularly useful on their own but can paint a good picture of the overall performance of an engineer or a project when proper context is applied usually by good manager who is often a senior software engineer themselves.

**Pluralsight Flow**

Formerly called GitPrime before the company was acquired by Pluralsight in May 2019, Flow is the world’s leading Engineering Logistics Platform. Flow is popular because it analyses a large number of variables from git repositories to measure the productivity of teams and individuals. They also gather information on the interactions between engineers at pull requests and they are able to give insight on the performance of an engineer or team over time.

**Timeular**

Timeular is company that has designed a method used by many professionals in the workforce. Software engineers in particular can use this to engage in the personal software process, which is described as the process of engineers using tools to measure their own productivity so that it can be used to better themselves or be used to help them secure new jobs or promotions or for contractors to satisfy clients. Timeular created a 8 sided die that allows workers to measure the length of time spent on a particular task such checking emails or attending meetings. the user just turns the die onto whatever side represents the task they are currently doing and the die will measure how long they’ve spent on that task and adds up the total time spent on each task a day.

***What Algorithms can we use?***

Machine learning is the study of computer algorithms that improve automatically through experience. It is a subset of artificial intelligence and it can be used to help corporations or individuals understand and measure the productivity of their software engineers. These will help automate the process of collecting data from software engineers and will use the data to infer certain conclusions based on what sort of machine learning is taking place.

Supervised Machine Learning involves learning a function that maps an input to an output based on a given example of input-output pairs. So the machine is given a starting point and a destination and has to map out the journey between the two essentially. A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. So for example if we had the what we believed to be a top level engineer and we gave him or her a task to do, and we compared their code after one hour to their code at the end of the task we could use the time taken and use that as a barometer for how long it would take a mid-level engineer based on the algorithm created.

The next machine learning approach is Unsupervised Machine Learning. It is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels with human supervision at a minimum. It contrasts supervised learning in that it doesn’t take predetermined outputs to map algorithms. A method of unsupervised learning is cluster analysis which involves grouping together data that has not been categorized and identifying similarities between data and then looking for the presence or absence of these attributes in new data sets. This could be applied to measuring software engineering by perhaps looking at the type of debug code used by known productive software engineers, and then using the data of other software engineers to infer whether these engineers have efficient debug code based on the similarities in their code compared with the productive engineers. Another subset of unsupervised machine learning is Anomaly Detection which takes lots of similar datasets and looks for outliers in them and then applies this thinking to the next dataset it receives to see if there are any outliers or anomalies in the dataset. This could also be applied in analysing the work patterns, the commits or code of software engineers and using it to find positive or negative outliers of other software engineers while applying appropriate context.

The final machine learning approach I will discuss is the Reinforcement Machine Learning approach. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. It differs to supervised learning in that it does not take predetermined outputs, its similar to unsupervised learning in that respect but differs from it in that unsupervised learning compares similarities while reinforcement learning involves learning the answer by itself by leaning on its experience and hence the more experience it has the better the solution will be. The possible application of this to measuring software engineering could be perhaps giving a machine a dataset of previous engineers attempts to solve a coding puzzle. The machine learns from all these different approaches and tries ascertain the best way to solve this puzzle. The machine can now be a sort of guide to other coders who attempt the challenge and can help increase their productivity with all the experience it has from the dataset of previous engineers. The overarching theme with these machine learning approaches to measuring software engineering is that a lot of data will need to be gathered and I will discuss the ethical concerns of this as well question whether machines should actually assess humans in this way.

***Is this ethical?***

Is gathering the code of a software engineer unethical? Not in my opinion. Tech firms will likely have access to engineer’s code via clauses in their contracts anyway. What about services like GitHub? Is it unethical to track the every move a developer may make whilst developing a piece of software? Again I think as a way to measure productivity it’s completely fine but what else could this data be used for possibly. Could an engineer’s code be possibly fed into machine learning algorithms that are essentially created to try and replace that very engineer. Moving to Artificial Intelligence isn’t unethical in itself but would engineers agree to be tracked if they knew that the knowledge from the commits and repositories was going to be used to perhaps make them redundant. That would be an example of practice that isn’t bad per se but could be used in an unethical manner.

But what about other types of data gathering? It isn’t only the code and Git data that could be used to measure the productivity, lack thereof or a fall or rise in it. There are so many methods that can be used. For example, in times like this where there isn’t any engineers at work in offices, measuring the productivity of engineers may prove more difficult. To combat this some employers could instruct their employees to wear tools that gather personal data, say perhaps a Fitbit watch that measures their heart rate and tracks their sleep. This information could actually prove very useful to firms who want to know why their assets aren’t performing quite as well but it presents a massive ethical issue as these things such as heart rate, exercise and sleep are seen as private. It would be highly unethical in my opinion to for firms to gather this type of data because I don’t feel like individuals could feel completely secure with that information being in the hands of corporations who are sometimes the victims of large leaks.

The next ethical issue I will discuss whether we should trust machines to analyse and assess humans. All machines are made by man the end of the day and machines can be corrupted to fit man’s will. The ethical question is whether we can be sure that those who create algorithms for the purpose of assessing humans performance are doing so with the best and most fair intentions at all times especially as the conclusions drawn from these algorithms can affect another person’s livelihood. Overall I wouldn’t lean in favour of using this technology for assessment purposes because I personally wouldn’t trust a machine to make a better assessment than a human would, even given humans and all their inherent biases. It also has to be looked at in reverse and be recognised that humans can fool machines and we have many examples of scientists and engineers purposefully fooling artificial intelligence. So another ethical problem arises as now those who have chosen to game or trick the machine will be judged differently to those who haven’t done so. For example if you have a simple algorithm that gives a score based of the number of commits per day, this could easily be gamed by increasing the number of commits while decreasing the amount of code per commit. Someone else could do the same amount of work but receive a lower score because they haven’t done that.

Overall I am largely against the use of machines as a means of tracking the productivity of a software engineer because I feel that the machines are too easily fooled or corrupted. I do however endorse a more human approach involving managers who can apply context to the data gathered.

References

<https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/>

<https://en.wikipedia.org/wiki/Git>

<https://en.wikipedia.org/wiki/GitHub>

<https://www.upguard.com/blog/bitbucket-vs-github>

<https://www.pluralsight.com/product/flow>

<https://timeular.com>

<https://en.wikipedia.org/wiki/Machine_learning>

<https://en.wikipedia.org/wiki/Unsupervised_learning>

<https://en.wikipedia.org/wiki/Supervised_learning>

<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>