Measuring software engineering processes

Introduction:

The term software engineering was first defined by Fritz Bauer1 in the 1960s “as the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines”. In todays economy, efficiency is one of the key factors to success in companies and organisations. Especially in IT industries, the efficiency and productivity of software engineers are vital to the company’s output. It becomes very important for managers etc. to acknowledge and monitor worker’s productivity. Although quantitative measurement of one’s development can be premising to determine whether the worker is well engaged in the workload. It can be quite difficult to define a perfect metric for software engineering process. This is due to the nature of software engineering to be non-binary production like some other professions. Software production is very dynamic, many aspects of the end product can change before and after release. There is no completely bug free softwares. Thus there are various numbers of metrics that can be used to measure software engineering process out there that will be discussed in this report.

Measurable data:

In the field of software engineering, there are many measurable data to determine the performance of software engineers. A common and simple one we see is simply counting the number of lines of code written by the software engineers. This is very unreliable in relation to measuring a software engineer’s performance. The reason being the number of code lines has no direct relation to the quality and importance of the issue being addressed by the code written. Different software engineers undertakes different approaches to implementing a functionality and fixings issues. They may require few lines of code or more lines of code but offers same functionality. In other cases, few lines of code might have been a fix to a major bug in the software that the programmer was working on. Fixing a major bug in a software should not really be considered as ‘slacking off’ if the measurement was purely done through lines of code. However it would seem that the programmer is not doing enough work as he did not write much code although the bug may have taken much more work in debugging and researching. Many lines of code also might not add meaning to the project it-self. Abusing this metric approach might lead to programmers writing unnecessarily long codes to appear as if they are doing a lot of work such as turning a short loop into many conditional statements. As we can see that this metric has many flaws and should never be used along to measure a software engineer’s performance. Although it might be useful to be coupled with other metrics to increase reliability of performance measurement.

An other common metric we see related to the previous one is the time spent coding. This approach, like measuring lines of code, is not very reliable. The reason being the time spent coding does not automatically reflect the amount of work done by the software engineer. It also does not provide and information on the amount of meaningful work done by the programmer. The programmer may be solving a major issue that requires much research and analysis, although the actual coding part may only require few minutes. The actual workload of this task is not accounted through this approach. Like before this metric system can be easily abused where a programmer can simply spend useless time on writing pointless code instead of doing actual meaningful work.

Number of bug fixes can also be measured to add to performance of a software engineer. This approach is very inconsistent in many scenarios. A software engineer may have different tasks in a development team and thus some may fix more bugs than others. Is that to say a programmer is not doing his work? Not really as that programmer might be working on other things than bugs. Also some bugs are much more difficult fix than others therefore there is no measurement model to take the importance of a bug into account when measuring through this approach. This metric can also be easily manipulated by a sly engineer.

Lastly, number of issues closed is also a measurable data that may be contributing to a software engineer's performance. As of the above simple metrics this one shares the similar flaws where the number of issues does not mirror the magnitude of the workload. Some issues are more important and more difficult to solve than others thus measuring the number of issues gives no meaningful measurement to the software engineering process. Like all of the previous metrics, this can be easily manipulated and abused by a programmer to appear productive and efficient. Up until now, simple metrics has been discussed with no real benefits to analysing the software engineering process.

Tech debt is Perhaps the most elusive measurement, tech debt can be described as extra development work that arises when code is implemented. A quick solution today might solve an immediate problem, but create other problems that need to be solved in the future. Code duplication and complexity, lack of documentation, and programming rule violations all contribute to it.There is no single formula to calculate tech debt. However, identifying the debt sources and estimating the amount of added development time is a good place to start. This can be a metric to software development as tech debt Essentially adds extra work in the future which in turns mean the less tech debt the better. However on the other hand tech debt may be good if some current bugs are difficultly to solve while causing a tech debt may be creating issues latter, the future problem may not be as obnoxious as the current one.

Measuring it:

As for actual measurement, there are vide variety of softwares and apps out on the internet for tracking workload. The most common being work time tracking. Most of them essentially track the time one spend working from when the worker decides to press a ‘start’ button to a ‘end’ button. This easily be manipulated and false data will be generated. Other more sophisticated softwares like Timecamp and Desktime not only provides time tracking, they also allow tracking of the time spent on individual app or websites. This can keep manager updated with what you are doing during your working hours. Individual reports can be generated and visualised of ones productivity. In some cases a recent screen shot of the computer can also be obtained to further keep track of ones work. Unfortunately this rather not very useful as event though the worker might be spending time on certain websites or applications, they might not be doing meaningful work.

For measurements specific to software engineers, data on GitHub can provide information on lines of code, issues closed, commits, updates, time information, engagements etc. Those are however not very appropriate and accurate measurement to the software engineering process as discussed above. With things like Trello, a development team might be able to track tasks that individuals fulfils and measure a software engineer from there. In many companies Jira Software is used to keep track of team development. While it can be used to measure individual contributions, team progression is rather the main goal of this software. Using it a software development team’s progression can be monitored through reports and boards similar to Trello.

Ethics:

Privacy is the main concern when dealing with measuring software engineers. For the likes of tracking softwares, users application usage can be easily seen and even a screen shot of the screen can be obtained. This can easily compromise one’s privacy. Even when privacy is not compromised, the fairness of these measurement should also be considered. If someone is getting promoted because he appears to be doing lots of work according to the metrics used while there may be others who are more deserving than him. Especially in the current situation where a fairly reliable metric model has not yet being developed for a software engineer, the use of it to fully evaluate a software engineers is highly unethical.

Conclusion:

As we can deduce, software engineering measurement can be highly challenging, controversial and open to discussion. There are absolutely no set metrics that can work to measure software development process. They can serve reference purposes. Quantitative analysis on a software engineer appear to be more or less unfair as the true depth of ones work cannot be calculated generally. More effective evaluation of a software engineer may be of the likes of peer review and self-assessment reports etc.