Measuring Software Engineering

Measuring Software Engineering

Varjak Wolfe – 18325326

Measuring software engineering is an important aspect of employee management. If you could measure software engineering, you could identify the effective employees on your workforce as well as the ineffective ones. Unfortunately, measuring engineering is not as straightforward as it might seem. It is complex to the point that there isn't really any industry standard for it. In this report, I'm going to examine several ways we can measure an engineer and their performance.

We will first look at the concept of a "software metric". "A software metric is a measure of software characteristics that are quantifiable or countable."(Stackify, 2017). This will be examined closely in my explanation of the measure of software engineering.

"There are no universally accepted methods to measure knowledge worker productivity, or even generally accepted categories"(Ramírez, Y.W., 2004). The "knowledge worker" is said to think for a living. This includes doctors, scientists, lawyers and programmers. We would typically associate manual labour with measurable outputs but there are some ways that we can examine the "output" of a software engineer.

I think software metrics can often be a little unclear in their measurement, given there are often multiple different ways of measuring them.

Lines of Code

For example, the lines of code (LOC) characteristic is a commonly used metric. One method is to count every single line, but many don't accept this because it can include lines of so-called "dead code" or comments. Another arguably better method is to only count each logical statement as a line of code. Due to this vagueness, its difficult to use this metric to measure a developer's output because there isn't a standard definition for it. It can thus be useful to pre-establish a method of measurement that is used consistently throughout the whole project.

Even so, it's a dangerous metric to use. If the first method of LOC is openly used as a measure of employee's work, you will find that the programmer's revert to using code that uses as many lines as possible. You can do a for loop in one line or four. The second method is much harder to dupe, and could be further strengthened by enforcing strict standards for writing and formatting code, as many companies do. Counting logical statements will mean a for loop, no matter how it is written, will have the same value in the measurement. This method isn't susceptible to strategic formatting. However, it is still a contentious method because it has no standard definition for what counts as a "logical statement" and this is further contrived by abstraction in different programming languages. As languages further develop and abstract, its becoming more difficult to create a tool with universally-accepted standards for measuring software engineering metrics.

Commits

Committing code means adding your work to an existing codebase. It can be seen as a log of every contribution a developer has made to the project. “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, 2016). Similarly to LOC, as outlined above, this metric could be artificially inflated by engineers, as instead of committing larger chunks of code, they would simply commit every small change. Another very important point to make about measuring commits, is that the content and frequency of a commit is often down to personal preference of the developer. Such a metric would not be effective for measuring their performance.

Tickets

Tickets are essentially tasks. Its yet another form of one-dimensional measurement that is ineffective. It would look at the number of tasks an engineer has completed in a period of time, such as a week. This would simply lead to engineers opting for smaller or easier tasks or breaking larger tasks into smaller ones that add up to more. However, it could be improved. If the tasks were written properly and assigned according to business priority, it may be a more effective measure of productivity. This means controlling the meanings of the tasks, to make sure that each task ticked off the list is a meaningful one and thus we can measure meaningful work being done. Another useful modification is scoring tickets based on effort or time required, in order to balance one engineer taking a long, complex task with an engineer taking multiple smaller tasks. This is where we break out of the first dimension of measurement, and begin to look at productivity within a context, instead of simple metrics.

Adding Context to Your Measurement

One-dimensional software metrics, as we have seen, are largely useless without a proper context. It is rare for a developer in the engineering speciality to work on their own. The productivity of an individual engineer must be measured within the context of the performance of their team and its goals. How do we measure the productivity of a whole team? The industry standard for work structure in modern teams is the “agile structure”. This is a non-hierarchical flat structure “where people are given the autonomy to work independently and organise themselves. Each team member has a defined role and responsibility, but unnecessary layers of management are removed, enabling people to self-manage effectively.” (Wrike). The relevant metrics for such a team structure are “lead time, cycle time, team velocity and open/close rates”(Lowe, 2016), but we’ll also look at some others too.

Lead Time vs Cycle Time

“Lead time and cycle time are often confused with one another. Both cycle and lead time are important time metrics in manufacturing, but they’re also important strategic tools for project management” (Waida, 2021).

Cycle time is the time it takes for a team to finish a project. Specifically, it is the time from the moment the work is in progress to the time it has been completed. It does not include the planning stage or the time it takes for the idea to be formulated.

The lead time is the time it takes for a product to be created, developed and released to the relevant platform.

Measuring cycle time allows you to identify any problem areas in terms of the team's efficiency. Measuring lead time however looks at the wider picture of how many products the team is taking on and how long it takes for them to fully form and execute the idea and release the final product.

Lead time is the vital metric when considering business goals. With business, the amount of code you can write within a period of time is pointless if you can't completely develop a product within a team and release it.

Team Velocity

This may be one of the most important metrics in measuring software engineering. "Velocity is a term used in agile software development to illustrate the 'rate of progress' for a team."(Software Development Today, 2008). It looks at the number of units completed within the specified time period. This can be very useful for estimating future sprints. It allows a team to estimate how much work they can complete in a given time, based on how quickly they have in the past. You must first decide on the unit of work, (e.g. engineer hours or backlog items) and also the interval of time within which to measure. This however has its own issues, like 'point inflation' where teams will estimate more conservatively, such as claiming that a 2 hour task will take 4 hours, and then they receive more credit for getting it done in half the time they said it would. It also doesn't take into account the quality of the work or the alignment with user goals. Team velocity can be increased by neglecting quality of design and coding standards. Similarly, since velocity measures work done regardless of the benefit of that work, you can have cases where unnecessary features are completed in order to add another unit of work to the calculation.

Due to its issues, team velocity should only be used for estimation and planning, not for measuring performance in your company.

Open/Close Rates

This metric measures how many production issues are raised and subsequently closed within a specific time period. This is a further modification to the tickets metric we already looked at. Here we are also looking at the productivity of a team based on the rate at which they both open and close tickets.

Measurement Technologies

There are many such technologies to choose from that allow companies to measure performance within their workplace. This allows for tracking of much more complex codebases in which it might not be possible to monitor everyone properly.

Github offers a service called ‘Insights’ where it is possible to track the amount of pull requests, whether merged or open, as well as closed or open tickets. It shows each repository member’s contributions, including lines and commits, as well as a timeline of recent commits and branches/merges.

Sealights offers service software focused around test code and test coverage, because this is a solid of measure of the quality of code being implemented into a project. They developed their own maturity model to help companies benchmark the quality of their software production. It provides real-time insights across the entire delivery process of a product.

Another popular choice among agile teams is Jira. Originally designed as a bug tracker, it is now a powerful management tool. It provides scrum and Kanban boards for ease of tracking tasks and completions, as well as time-tracking and real-time performance reports including burn charts, sprint reports and velocity charts. The creators understood that much development is iterative on existing code.

Ethical Considerations

You can't discuss any form of data collection without touching off of the ethical dilemmas this presents. Data, in sufficient size, can be used to predict pretty much anything, which is beneficial in its own right, but also brings about its ethical downfall. Its obviously beneficial for improving productivity and efficiency within software development companies but does it allow for too much monitoring over employees? Can a stakeholder viewing data trends find a drop in an engineer's work and determine that this employee is perhaps thinking of moving to a different company, and thus pre-emptively fire this employee? There must be, as with anything, a balance (and government regulation). C:\PythonProjects\GithubVisualisation

Links

Commits Do Not Equal Productivity | GitLab

Defining an Agile Team Structure | Wrike Agile Guide

Cycle Time vs. Lead Time: Everything You Should Know | Wrike