Measuring Software Engineering

By

Oisín Thomas Morrin

18321152

(XXXX words)

**Abstract**

In this essay, I will open with identifying the scope of measuring software engineering, before detailing how software engineering can be measured, the platforms that can be used to gather and process the data, the algorithms that can be used to do it and finally answer the personal question of whether it is ethical at all to do any of this - and if it is, to what extent.

**Introduction**

The initial question of whether measurement is required can be easily answered with a resounding yes. Lazlo Bock of Google found that from a hiring perspective, the best indicator of the success of a future employee at the company was taken from a candidate performing a work sample test (29%) (Bock, 2015). But one could make the argument that perhaps basic success on the task is all that matters. Interestingly, the second best indicator is a general cognitive test (26%) (Bock, 2015), which would serve to make the point that, though binary success or failure is important, it is still too general, and that there is more at play. Working inductively from the hiring stage, it is evident that monitoring the work of employees would be of benefit. It is to make the general precise that the use of metrics is becoming pervasive.

As proof of the use of metrics, Harding (2020) uses the example of GitPrime when he reported:

“GitPrime (now Pluralsight), the earliest entrant to today's developer productivity space, did an excellent job of following up with their customers and documenting the impact that performance metrics can have on results. Their case studies include a 137% increase in Impact by Storyblocks, and a 25% increase in measured Impact enjoyed by Adext.”

But to what extent will we be considering these metrics? When considering measuring software engineers, there are a variety of metrics available to the organisation or person in charge of the engineers. The first question is whether the issue can be dealt with by analysing the individual alone. The measure of the productivity of the individual seems to be the keenest way to decide from a management standpoint whether to retain or fire and hire again.

If developer productivity can be measured, how is there not a single standard of measuring this in the industry? It tends to be because of a process called Goodhart’s Law: Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes (Wikipedia, 2020). In other words, all simple metrics will be gamed - be that lines of code written, number of commits, tickets posted etc.

What of the team? As has been seen in recent years, software engineering metrics for the individual can dissolve in the face of an awesome team: Whatsapp was a team of 50 engineers that now serve over 2 billion users (Bucher, 2020), and Instagram started as a team of 13.

So it is clear that a mixture of personal and interpersonal metrics will be needed.

**How can Software Engineering be measured?**

Now we get to the question of how software engineers can be measured. What ‘data’ does a software engineer produce by doing their work? In, truth there are many pieces of data produced daily: number of lines of code written, the number of commits, timestamp of commits, average commit size, time to review, number and frequency of code reviews, time to merge pull requests to the main branch, test coverage, number of contributors to the project. As we can see from above, none of these metrics are helpful by themselves as any one of them can be gamed, but together, they give a fuller image of the software engineer – highlighting their strengths and weaknesses.

When looking at things from a team level, there are even more metrics to be considered, including: team interactions on platforms, the morale of the team, time it takes to complete projects versus the projections for the project, etc. Arguably, the latter data set is harder to map, though there is an abundance of data to extrapolate from.

Now we will have a look at some of these metrics and their level of usability.

**Number of Lines of Code**

This type of data is probably the easiest to obtain, and for that reason, much like counting lines of writing for a writer, it can be the most misleading. Bill Gates made this point abundantly clear when her said: “Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weights.” Measuring the length of code one writes doesn’t “take into account the intelligence, content and layout of the code” (VerifySoft, 2020).

Encouraging the metric of number of lines of code to be used, it can lead to detrimental effects. In the article by VerifySoft (2020) they recommend file lengths to be between 4 and 400 lines, with any longer leading to a decrease in readability. Also with the concise expressions being used across many languages (for example, lambda expressions) using the alterative long form can be seen as not adhering to a modern code style and perhaps even a loss of functionality that newer paradigms bring. In truth, the number of lines of code a software engineer produces is an extremely poor metric. Perhaps a heuristic should be employed in this case: “once its not zero, zero worries.”

**Number of Commits, Size of Commits & Commit Messages**

Much like the number of lines of code being used as a metric, using the number of commits that a person makes at face-value can be misleading. Additionally, commits require context: they can only truly be evaluated based on their utility to the project. Commits speak of the attitude of the software engineer. Unlike the number of lines of code metric, the number of commits a software engineer can be useful in seeing how open a software engineer is to sharing the code they write for others to evaluate, and more generally, how much time it takes for them to reach a level of satisfaction that allows them to share (hopefully) a working piece of code.

The size of the commits is also tied to the number of commits. Like the idea of gaming the metric on code length, if one spreads a piece of code out across multiple commits, the value of the commit drops dramatically.

That brings us to the third facet of this data: the commit message. The commit message signals the utility of the commit to the community of software engineers on the project. That is why the idea of making good commit messages is so important.

Though one would hope there are few software engineers so cynical as to game the data, when accounting for commits, it is important to measure the size of the commit along with the number of commits as well as the utility of each commit (through the community’s appraisal of the value of the commit and commit messages) as they depend all on each other to make one useful piece of data.

**Number of Contributors to A Project**

The number of contributors to a project is another important piece of data in the puzzle to understand how one measures software engineers. This data point is a collection of data taken from the absolute number of contributors, the size of the contribution of each and must take into context the time it takes to complete projects versus the projections for the project (if there were any deadlines). This is a measure of the competency of the individuals themselves (by measuring their input) and also of the team (by measuring how far they came together). It is clear from so many examples, like Instagram and Whatsapp, that the number of contributors and quality of the contributions can make or break a project. By looking at this data, one can see glimpse the morale of the team and the quality of the interplay between the software engineers.

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

Nowadays, there are a variety of platforms that individuals and organisations can make use of in order to measure software engineering through the data I’ve spoken about above and many others. Like most aspects of business, there are both free and paid services available.

Free tools/platforms include:

* GitHub API
* GitLab API
* Bitbucket API
* Code Climate Quality (from free)

Paid tools/platforms include:

* Pluralsight
* Code Climate Velocity (from free)
* GitClear
* Pinpoint

It must be noted that there are differences in how the data is processed and gathered even among these tools in order to measure a software engineer. Many of the paid tools and platforms will collect the data and funnel them through an algorithm before providing the result – the fully automated approach. In other circumstances, the raw data is provided and one must follow other lines of enquiry (sub routes) within the platforms in order to wrangle meaning from the data.

More information on some of these tools can be found in an article written by Bill Harding (2020). Additionally, there are a variety of platforms available that deal in productivity in the workplace that aren’t specialised in measuring software engineers but can be useful in some respects.

**What algorithms can we use?**

<https://semmle.com/assets/papers/measuring-software-development.pdf>

**Is this ethical?**

**Conclusions**

**References**

Bock, L. (2015). Here's Google's Secret to Hiring the Best People. Wired. Retrieved from  <https://www.wired.com/2015/04/hire-like-google/> on the 13/11/2020.

Bucher, B. (2020) WhatsApp, WeChat and Facebook Messenger Apps – *Global usage of Messaging Apps, Penetration and Statistics*. Messenger People. Retrieved from <https://www.messengerpeople.com/global-messenger-usage-statistics/> on the 13/11/20.

Harding, B. (2020). Measuring developer productivity in 2020 for data-driven decision makers. Retrieved from <https://www.gitclear.com/measuring_developer_productivity_a_comprehensive_guide_for_the_data_driven> on 14/11/20.

VerifySoft. (2020), Measurement of Lines of Code – *Metrics with Testwell CMT++ and Testwell CMT Java*. Retrieved from <https://www.verifysoft.com/en_linesofcode_metrics.html> on the 14/11/2020.

Wikipedia. (2020). Goodhart’s Law. Retrieved from <https://en.wikipedia.org/wiki/Goodhart%27s_law> on the 13/11/2020.