Measuring Software Engineering

By

Oisín Thomas Morrin

18321152

(3428 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 (2020a) 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.

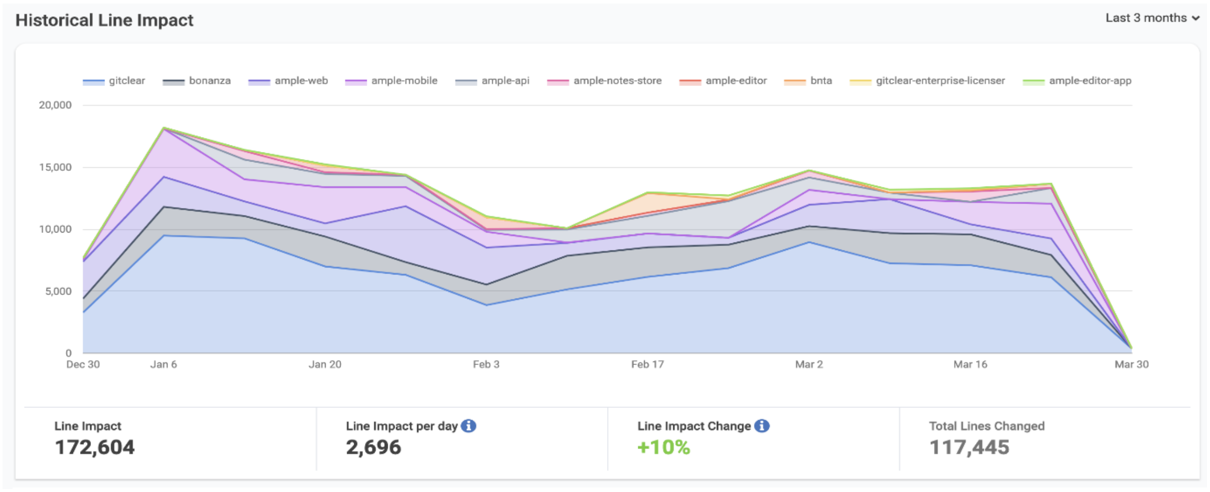
Though one would hope there are few software engineers so cynical as to game the data, when accounting for commits, it is important to measure the size of the commit along with the number of commits as well as the utility of each commit (through the community's appraisal of the value of the commit and 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.

Line Impact

One extremely interesting platform I found when researching for this essay was GitClear. It boasts a metric called 'line impact' that is designed to measure how much cognitive energy is being put into software development which is explained as it “cancels out all of the interstitial activity ("churn") that happens as a feature gets developed, leaving a concentrated embodiment of the work that took place” (Harding, 2020b). It is a cousin of the metric that we discussed ‘number of lines of code’ but instead of measuring the bulk, it measures the changes that matter through a combination of factors: code churn, activity type, file extension, domain, related commits, ignoring branches, merge commits and duplicate sections; file purpose, and custom ignored files.



Line Impact	Line Impact per day	Line Impact Change	Total Lines Changed
172,604	2,696	+10%	117,445

Fig 1: 10 repos graphed by their Line Impact, a metric that quantifies the pace at which source code is evolving, image courtesy GitClear (Harding, 2020b).

What Platforms Can Be Used to Gather and Process Data?

Nowadays, there are a variety of platforms that individuals and organisations one 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. Additionally, there are places that offer a thin array of data processing over their true purpose which is to provide data (metrics).

Some sources of metrics:

- GitHub API
- GitLab API
- Bitbucket API

Free tools/platforms include:

- Code Climate Quality (from free)
- Hackystat (no longer under active development)

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 available (like in GitHub API, GitLab API or Bitbucket API), and one wrangle meaning from the data yourself or by using a third party platform.

One other important point to note in this section is that, in many cases, the data is merely gathered and presented visually, without any real novel processing. It can be argued that much of what one is using these tools for is visualising the data and then having to infer your own insights from the data. How the data is presented is still very much up to the organisation providing the service and indeed the user of the service.

More information on some of these tools can be found in an article written by Bill Harding (2020a). Additionally, there are a variety of platforms available that deal in productivity in the workplace that are not specialised in measuring software engineers but can be useful in some respects. For example, one could use the productivity application Trello for listing OKRs (Objectives and key results). The idea of using OKRs to measure performance of software engineers and managers is known to be used by Google – you can read more about this and OKRs in general in this article by Felipe Castro (2020).

What Algorithms Can We Use?

And much like there are a variety of metrics that one can measure, there are a variety of tools at one's disposal. These include the buzzwords of the day: computational linguistics, artificial intelligence, machine learning and deep learning. There is also the burgeoning area of data visualisation which is the most common process employed still. In this section, we will look at these process and algorithms used to interpret insight from the data.

Data Visualisations

The first process we will look at is data visualisation. There is a plethora of visualisations at our disposal, from bar charts to scatter plots, network maps to density graphs, and many more. As I showed previously – in regards to Line Impact by GitClear - the line plot can be used and is good for visualising changes in data values over time. In many respects, the simplest visualisations – like pie charts, bar charts, histograms, scatter plots – are ones we are all familiar with, so I will therefore focus on highlighting more obscure and abstract graphs that can be employed with great effect.

- **Network Maps**

These network maps are important when trying to understand what software engineers are interacting with a project: in regards to whether they are collaborating on files and whether engineers are working across many files or on a particular group of files.

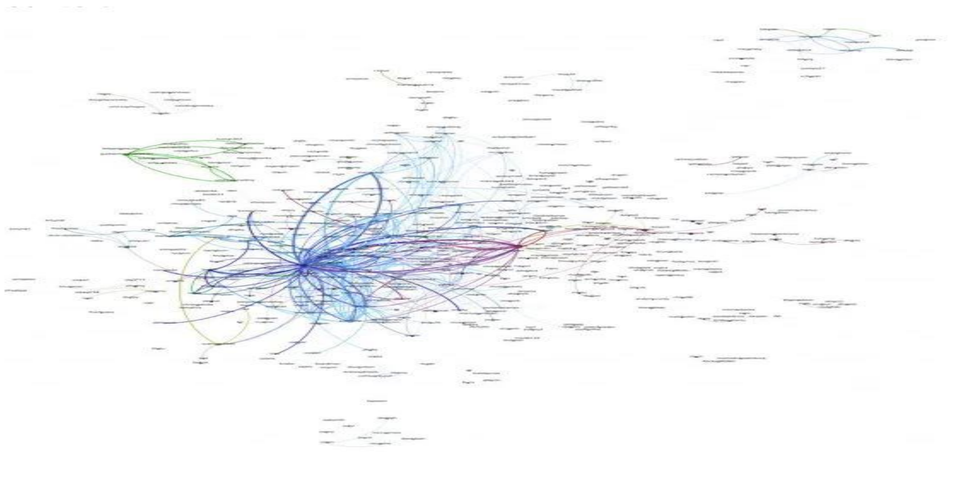
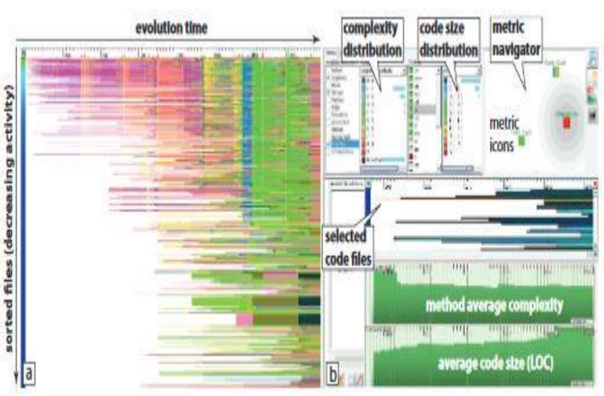


Fig 2: Social Network Graph of Python and its dependencies (FlowingData, 2014).

Complex graphs are important for displaying data and its relation to other data across multiple parameters. They are a double-edged sword. They aggregate data that is visualised in various ways to compute a visualisation that is much richer in content, and hopefully context, but can be hard to interpret due to the increase in content.



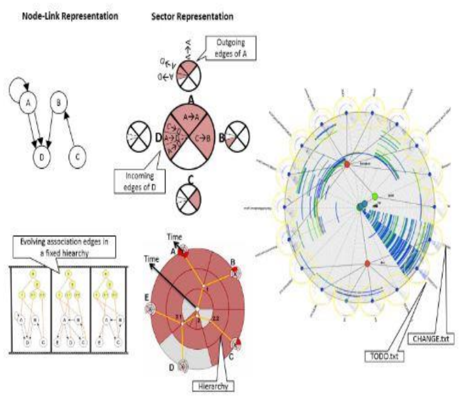


Fig 3 & 4: Visualization of software activity and code quality trends in a repository and of file co-change across multiple software versions, respectively (Diehl & Telea, 2014).

Machine Learning

The use of machine learning techniques is extremely useful in gaining insight into data whether it be a classification problem or a regression problem. There is the humble linear regression that can help predict performance outputs for given parameters of time and experience etc., and logistic regression that be used to understand whether a software engineer will likely quit. K-Means Clustering can be used to see groups within the data obtained from software engineers that can be investigated further. These are but a few possible algorithms, with an example of their application.

Arguably the most used algorithm at the moment are neural networks. In a paper by Hélie, Wright and Ziegler (2018), they investigated how one could use machine learning (in particular, neural networks with hidden Markov models) on version control data to measure software development productivity.

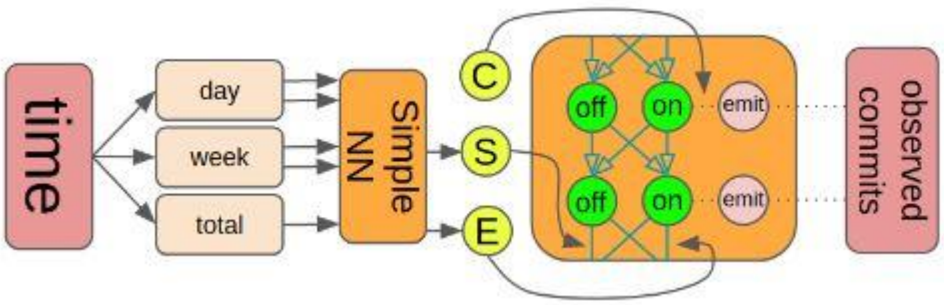


Fig 5: A graph of the neural network used by Hélie, Wright and Ziegler (2014).

Artificial Intelligence, which machine learning is a subset, and Deep Learning are also starting to be used to be used with the data.

➢ Word of Caution

Though, one must be cautious when using these algorithms as they contain biases that must be understood when using them. For example, it is very difficult to understand exactly what is being done to the data (in other words, what affect a piece of data is having on the output) in deep neural networks. There is also the finer point that the algorithms are mathematical constructs that are ignorant to the meaning of the data and are in truth finding patterns across data that has lost much of its contextual meaning. Also in regards to these algorithms, one must be careful of overfitting or underfitting the data (the former case being harder to detect). Additionally, many of these models require large amounts of data in the first place in order to used. All of these issues lead us to question the ethics of processing the data and ultimately leads us nicely to the final point for discussion.

Is this ethical?

Ethics may be said to be subjective, but that does not mean there aren't very important and obvious considerations when deciding whether something is ethical or not. And, in regards to using data from people, one particularly set of issues is how it is collected, stored and distributed. As we are quite aware of nowadays – especially through the advent of GDPR in Europe – is the danger of data being collected unbeknownst to the person and then being sold to third parties who use this data to derive profit through sales, advertising and other means; or perhaps the company themselves for some purpose that was not stated. GDPR in Europe is in place to provide a framework for businesses to follow in order to process data legally, and also giving the person more control over their data.

By giving employers access to the data of software engineers who work under them – both data I have mentioned and those like, posture and heart rate etc., that I haven't – there has to be trust that the data is indeed necessary and will only be used for the purposes of increasing the effectiveness of the software engineer. And is that trust there? Can we say that these organisations will treat the data ethically? I'd like to argue no. History says we can't, and can be confident in continuing not to trust them based on how often new litigation is brought against companies for the very reason of exploting data. At the end of the day, its about profit, and not people in the majoritty of organisations and businesses around the world; and, I expect that, without a clear delineating between work and private life, soon there will only be the monitored life. Using the metrics I spoke of will ultimately lead to the intrusion into the private life of the software engineer because we are human, and that means that so much more impacts our work than is simply measurable by the work itself – like our psychological state and home-life. And employers will want more context for their algorithms; and more context will lead to more data having to be collected; and more data will mean the erosion of a private life. Frankly, whether one has kids, or sleeps well, or has a row with their spouse, will indeed impact on work . So what’s to say that ‘for the sake of accuracy’ that employers won’t strap Fitbits onto the software engineers and ask for access to their phone’s microphone? In some cases, the former has already happened.

This monitored life will be very hard to resist. There is a growing push that the answer can be gotten from having simply more data to make up for the ignorance of the data in regards to reducing the context of each data point through normalisation etc. As I mentioned previously, large amounts of data are required for many of the algorithms, and we have already seen how a one billion parameter BERT model for natural language processing has shown such great results off the back of having so much data to work with.

I’d be of the opinion that workers should be encouraged to monitor themselves as in line with the idea of the Personal Software Process (PCP) framework: systems being built for the individual to use for themselves – not to find some global maxima of productivity by including how often they breathe and what tone they spoke to their colleague, but to find a local maxima for themselves, where they are in control of the data being collected and how its used.

Ultimately, the only person you can trust with your privacy is yourself. Indeed, that is why GDPR was brought in: to stem the tide that was Big Data harvesting people’s data in an unrestrained way for unstated purposes. Therefore, though the methods perhaps being employed at the moment are ethical (by virtue of how simplistic they are presently in regards to the metrics being used), I expect them to become more and more unethical as the usual temptation of greed is succumbed to.

Conclusions

In this essay, I have looked at software engineering metrics in terms of why they are needed, what metrics one can obtain from the software engineer, what platforms one can use to gather and process the data, as well as the processes and algorithms used to gain insight from the data, before finally looking at the ethics of the idea of measuring software engineers. From this essay it has become clear that metrics can indeed be useful and that there exists a variety of techniques and platforms available to monitor these metrics and gain insights into them. As for ethics, it remains a question of how far one can go and ultimately for what purpose the data is being used. One must be aware of the short-comings of the processes used to measure data, and that more data will be seen as the answer by many, but in truth may just lead to a plateauing in the increase in efficiency without solving any of the problems that the algorithms and processes had in the first place like ignorance and biases among training data etc. In the capital hungry world that we live in, arguably the only way to be use metrics is by yourself and for yourself in order to avoid the eventual exploitation.

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.

Castro, F. (2020). The Beginner's Guide to OKRs. Retrieved from https://felipecastro.com/en/okr/what-is-okr/ on the 15/11/2020.

Diehl, S. & Telea, A.C. (2014). Multivariate Graphsin Software Engineering. Retrieved from https://pdfs.semanticscholar.org/e5bc/335c8da66367620986ab7cd461bad8e13f42.pdf on the 15/11/2020.

FlowingData. (2010). Mapping GitHub – a network of collaborative coders . Retrieved from https://flowingdata.com/2010/03/31/mapping-the-github-community/ on the 15/11/2020.

Harding, B. (2020a). 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 the 14/11/2020.

Harding, B. (2020b). Popular Software Engineering metrics, and How They're Gamed. Retrieved from https://www.gitclear.com/popular_software_engineering_metrics_and_how_they_are_gamed on the 15/11/2020.

Hélie, J. Wright, I. & Ziegler, A. (2018). Measuring Software Development Productivity: *A Machine Learning Approach*. Retrieved from https://semmle.com/assets/papers/measuring-software-development.pdf on the 15/11/2020.

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.