MEASURING SOFTWARE ENGINEERING

CSU33012 Software Engineering

23RD DECEMBER 2021
IMOGEN GREEN
Student Number: 17326096
WORD COUNT: 4,074

Contents

Introduction

Today’s world calls for increased measurability in every area of our lives. From the number of hours that we sleep for every night to the number of steps we take, we are constantly being measured. And software engineering is no exception. However, there exists a long-held misconception that software engineering simply cannot be measured.

I believe that the crux of the issue regarding the measurement of software engineering does not lie solely in whether we are able to gather data; it is a question of ‘should we’? And if we choose to, how do we ensure that the systems of gathering and measuring are fair, unbiased, and add value?

Throughout this report, I will be examining how and why we might want to measure software engineering, the platforms that can facilitate this measurement and data-gathering, as well as the various algorithmic and computational approaches that allow us to profile software engineers based on the data that they generate. I will conclude with an examination of the ethics and legality of capturing this data, and an elaboration on whether I believe doing so is justified.

Measuring Engineering Activity

First, it is important to define what we mean by ‘software engineering’. Ultimately, it is a “process… characterized by a number of distinct phases” (Whitson, 2020). The most important part to keep in mind is that it is a process which can take on a variety of different approaches. For example, agile software development calls for continuous, incremental delivery where each step in the development and deployment of a software system is broken out into phases, starting with establishing the requirements of the system. Once a meeting has been held with the client, prototyping begins rapidly. The idea is that engineers move fast and continually reshape what will eventually become the final deliverable. How productive they are in reaching this point, however, remains elusive.

If we are to measure software engineering, we must be sure that there is sound reasoning behind why we are doing it, as well as ensuring fairness. Metrics in any system are subject to abuse, but there must be ways in which this can be mitigated. In the absence of fairness, distrust festers and serves only to detract from the work of software engineers who find themselves struggling to reach targets simply to appear productive. A good example is that of using lines of code (LOC) as a measurement.

Lines of Code (LOC)

Lines of code (LOC), also referred to as source lines of code (SLOC), remains one of the most commonly used metrics in software engineering, in part due to the ease of computation. It is interesting to note that there are two slightly different versions of LOC: logical LOC and physical LOC. Logical LOC pertains to “the number of executable expressions” (PVS Studio, 2013) in a codebase, whereas physical LOC counts the physical number of lines that exist. LOC is frequently used for purposes of “cost estimation” (GeeksforGeeks, 2021), with it being viewed as a measure of the effort that will go into developing the codebase. However, one of the most crucial drawbacks of LOC is that it does not measure code complexity, and therefore cannot truly be a measure of effort. I have given a very simple example below which shows how counting the lines of code in a program can result in code which takes up more space, simply for the sake of doing so. Please see figures 1a and 1b.

```
for(int a=0; a < 5; a++)
{
        System.out.println(a);
}
```

Figure 1a

```
for(int a=0; a < 5; a++) System.out.println(a);
```

Figure 1b

Figure 1a shows a simple code snippet that has two logical LOC and four physical LOC. However, figure 1b has two logical LOC and one physical LOC. This example highlights how this metric is easy to game. In the words of Bill Gates, “measuring programming progress by lines of code is like measuring aircraft building progress by weight” (Gates). Clearly, using lines of code is a poor metric that is open to abuse.

Take the example of Stripe, payment processing platform that requires taking just “seven lines of code” (Vance, 2017) to handle payments on your website. It is simple, if not ingenious. A formerly long and tedious process for developers to set up websites to take payments online was reformed forever. This highlights that the quality of code lies not in its mass, but rather in what you can do with the space available to you. To use LOC as a metric completely bypasses this and as such is a poor metric to use in evaluating developer productivity.

Why measure at all?

Like any activity we participate in, we want to be able to benchmark ourselves against not only what the people around us are doing, but also against how we performed in the past. As such, monitoring our productivity becomes important. For Chief Technology Officers (CTOs) across organisations, understanding efficiency of current processes, as well as productivity of engineers, becomes critical.

In other industries, the measurement is more straightforward. But with software engineering, the complexity of achieving meaningful measures of productivity skyrockets. And ensuring that it is done fairly is one of the greatest concerns for software engineers.

Perhaps CTOs wish to measure productivity of the engineers working in their company with certain development methodologies, for example more sequential processes such as the waterfall model compared with an agile approach. They want to understand what works best for their team so that they can facilitate "planning, organizing, controlling and improving" (GeeksforGeeks, 2020) current practices.

It is important to note that software engineering takes teams of people, each with their own "strengths and weaknesses that complement each other" (Danger, 2021). To isolate engineers and seek to measure them individually is not a true measure of the strength of the team itself, and this must be taken into consideration in the design of productivity metrics. While measuring and evaluating performance has a place, ensuring that these measurements are meaningful is paramount. Clearly, this is a complex issue, but fortunately there exist some platforms that strive to make measuring software engineering more achievable and straightforward.

Developmental Analysis Platforms

We have established that measuring software engineering is possible. In this age of 'Big Data', information is being constantly generated about all human activities, including software engineering. So how do we use of this data and use it to understand the efficiencies and productivity of the software engineering process?

We must start thinking about the platforms we can use to gather data, and the kinds of computations that can be carried out over this data. Data analytics provides great power and insight that would be otherwise near impossible to obtain. There are various platforms such as Pluralsight Flow that attempt to derive meaningful information from the data that they have, with the hope to identifying trends and important metrics. This, however, all hinges on the emergence of utility computing.

Utility computing

Utility computing provides a framework by which "computing resources are provided to the customer based on specific demand" (HCL Technologies, 2021). This idea of having platforms as a service (PaaS) and applications as a service (AaaS) means that companies can pay for data gathering and analytics so that they can better understand the workings of their software engineering teams. Several platforms providing such services are explored in greater detail below.

Case study #1: Pluralsight Flow

Pluralsight Flow, a product available from Pluralsight and formerly known as GitPrime, provides "deep insights into your engineering workflow" (Pluralsight, 2021a). It takes data from git to create visualisations that are fed back to clients in reports and insights that are easy to comprehend, with the aim of increasing the productivity of developers. As a platform, it provides multiple tools such as visualising pull requests and facilitating data-driven software development.

The aim of Pluralsight Flow is to help developers produce cleaner code with greater impact, reduced bugs, and to accelerate the process as a whole. It places great importance on establishing a strong, positive engineering culture within organisations, one whereby senior engineers provide mentorship to younger colleagues. Pluralsight Flow evidently seeks to enhance every facet of the software engineering process by using a variety of metrics and asking questions such as "what percentage of the team is involved in feedback?" (Pluralsight, 2021a), and facilitating analysis at granular levels to provide an understanding of, for example, monthly commit efficiency for each different programming language.

So… what is impact?

By measuring impact, we can attempt to determine the amount of "cognitive load" (Pluralsight, 2021b) that a software engineer was under while implementing code changes. This goes far beyond the more simplistic LOC measure outlined earlier. Instead, this metric considers:

1. How much code is contained in this change.
2. How much of the code consists of edits to previously written code.

3. Number of locations where changes were made.
4. The number of affected files.
5. How severe the changes are when modifying old code.
6. Comparison of this change to others that have occurred previously within this project.

This is all combined to produce an impact score, with a higher number indicating a higher impact. With all these considerations, impact is a much more difficult metric to game compared to LOC, and as a result can be a source of enhanced value for those who are trying to evaluate software engineer productivity.

Case study #2: Waydev

Waydev is another platform that provides analysis on developer performance, with a strong emphasis on producing "real-time data and automatic reports" (Waydev, 2021).

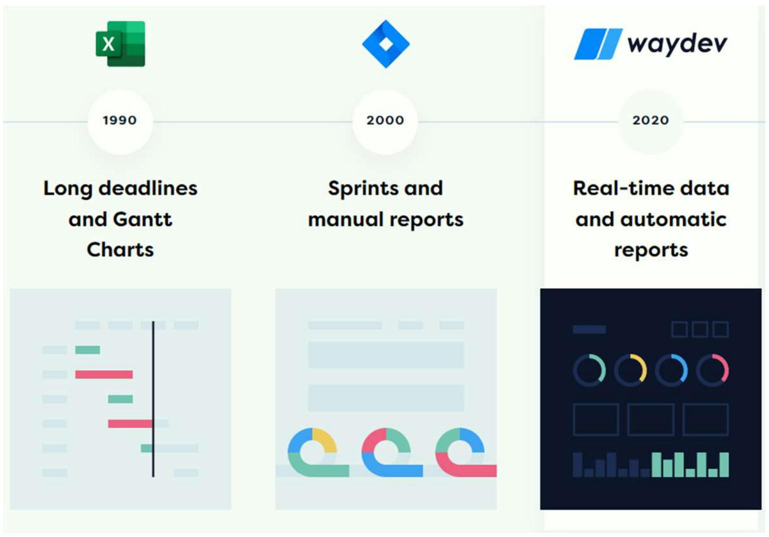
waydev
1990
2000
2020
Long deadlines and Gantt Charts
Sprints and manual reports
Real-time data and automatic reports

Figure 2: the evolution of software engineering productivity measurement (Waydev, 2021).

Waydev uses several metrics in establishing an understanding of software engineer productivity. For example, they use a metric called "help others" (Circei, 2021) which identifies how much recently written code is being replaced by a different engineer. Much like Pluralsight Flow, there is a strong emphasis on the speed of the software development lifecycle and how this can be accelerated using data analytics. In the interests of company executives, Waydev generates reports that answers their most burning questions. Particularly in the time of Covid-19, the effect of remote working on employee productivity is a core focus for managers and executives. Waydev promises to provide answers to questions such as

“’how did our coding efforts change with remote work?’” (Waydev, 2021) so that teams of engineers can work collaboratively, more efficiently, and at pace.

Case study #3: Code Climate

Code Climate dives into the codebase to generate an understanding of where “bottlenecked work” (Code Climate, 2021b) is occurring, while providing objective metrics of code quality that empowers software engineers. This platform takes data from Jira, a software development management tool, and DevOps tools to create these insights.

Code Climate places great importance on tracking progress while maintaining visibility across development pipelines, all while enabling development that is driven by data. This in turn aids software engineers in speeding up the software development lifecycle.

There are a number of concepts that Code Climate uses in understanding code quality and complexity (Code Climate, 2021a). These are:

1. Churn: an approximation of the number of times that a file has been changed in the past 90 days.
2. Cognitive complexity: how difficult it would be to read and intuitively understand the code written.
3. Cyclomatic complexity: in simple terms, this refers to the number of decisions made in a block of code; in other words, it counts the number of execution paths.
4. Duplication: this identifies code repetition and in turn helps to trim down codebases.
5. Maintainability: this produces an estimate of the “technical debt” (Code Climate, 2021a) in a repository by assessing duplication, cognitive complexity and cyclomatic complexity, along with structural issues.

Code Climate also provides reports of test coverage “for each file alongside quality metrics like complexity, duplication and churn” (Code Climate, 2018). This platform provides a robust and holistic analysis of code output impact, testing, and complexity that strives to enhance productivity of software engineering teams.

The value of developmental analysis platforms

While each of the platforms above can provide us with data and insights, it is important to realise that data can be subject to manipulation, and that the inferences it makes do not always tell the full story. It is important to recognise that these platforms are tools or services that can guide managers and organisations, but to use them in isolation would be a regrettable mistake.

Such data is useful only in the right hands, and when the person using the data has the appropriate industry experience and expertise to know when to accept what the data and insights tell us, and when to press for more. Data literacy and "knowing how to use data" (Ellen et al., 2015) is paramount. Ultimately, there is a fine line that must not be crossed; data should be used to guide human judgement, not offered as a replacement. This is where the true value of such platforms can be found.

Algorithmic Approaches

In an attempt to profile software engineering performance, various algorithmic approaches have been deployed. Some, such as measures of complexity, have been previously touched on in earlier sections of the report, but there are others that require deeper examination to understand how they can be implemented.

Before delving into the applications of algorithms to evaluate software engineering performance and productivity, it is worth noting that artificial intelligence (AI) is subject to bias. Training machine learning (ML) algorithms to think as a human would requires data. But due to the inherent biases that exist within us as human beings, the data reflects this bias. This is then fed into ML algorithms and can result in further perpetuating these biases.

ML and the concept of bias

A good example we can take here is that of Google Translate. When translating between certain languages, for example German to Spanish, gender can often be switched to the male of female equivalent depending on the noun that follows. This can be seen in "the phrase "vier Historikerinnen und Historiker" (four male and female historians) [which] is rendered as "cuatro historiadores" (four male historians) in Spanish" (Kayser-Bril, 2020).

Logically, and particularly in modern society, gender roles and norms are close to abolition. As conscious human beings living in the 21st century, we are aware of this. But all that an algorithm knows is what it has been told, and this is where things can become somewhat sinister. These systems can only ever be as intelligent as the data that trains them. As such, smaller training data sets will result in systems of lesser intelligence, while larger training sets will see intelligence levels that converge with levels of human intelligence.

However, a note of caution. Removing bias from data in its entirety is close to impossible at present, and until this can be achieved, using the decisions made by machines must remain an aid to human decision making, not become a substitute for it.

What about Computational Intelligence?

Computational Intelligence (CI) is defined by the Institute of Electrical and Electronic Engineers (IEEE) as being "the theory, design, application and development of biologically and linguistically motivated computational paradigms" (IEEE, 2021). In other words, this area is concerned with developing systems that can mimic human thought processes, and encompasses three main areas:

1. Neural networks
2. Fuzzy systems
3. Evolutionary computation

Neural Networks

A neural network (NN) attempts to emulate the workings of a human brain. These networks are "massively parallel distributed networks that have the ability to learn and generalise from examples" (IEEE, 2021). However, one of the risks in doing so lies in the word 'generalise'. One of the shortcomings of ML is that it smooths out data and, as a result, neglects outliers. In establishing patterns, a general behaviour or set of attributes is determined.

Fuzzy Systems

Based on what are known as fuzzy techniques, "fuzzy systems (FS) model linguistic imprecision and solve uncertain problems based on a generalization of traditional logic, which enables us to perform approximate reasoning" (IEEE, 2021).

FS are a powerful tool that can be deployed in situations where large volumes of data require processing to produce a developed understanding. Much like with NNs, there is an emphasis on generalisation and approximation, which creates its own set of problems.

Evolutionary Computation

Evolutionary Computation (EC) takes inspiration from natural evolution. In other words, it is a "class of global optimization algorithms influenced by natural growth" (Chanal et al., 2021). It encompasses "genetic algorithms, evolutionary programming, evolution strategies" (IEEE, 2021), and is used to solve optimisation problems and train NNs. Just thinking about how human beings evolved over time illustrates this strive for optimality, and how algorithms based on natural evolution would work similarly.

The approximation problem

Herein lies the exact same problem that exists with using the mathematical mean of a data set in statistics. This ‘average’ does not tell you everything that you need to know, and you cannot really paint the full picture without knowing the standard distribution, or the spread, of your data. While NNs, FS and EC are extremely powerful, this further illustrates the importance of using the patterns and generalisations made by such networks in conjunction with human judgement.

Where does this leave us?

There is still huge work to be done in developing algorithms that can accurately measure software engineering performance. The intelligence of these systems depends hugely on the data that they are trained with which has a profound impact on the conclusions and decisions that these systems will make.

In my opinion, we must consider the relativity of the data we capture. To improve, we must seek to do better than we have in the past, and we can only do this by comparing to historical data. In the context of software engineering performance, this could be comparing the impact of code commits to those made previously.

I believe that we are not yet at a stage where systems can fully emulate human intelligence. As mentioned previously, the inferences and conclusions that intelligent systems produce can hold great value, but only when used by the right people with appropriate levels of expertise to enhance decision-making.

Ethics and Legality

After examining the various ways in which software engineering can be measured, there remains a burning question: should we?

This is a hugely complex issue, and there are arguments to be made for and against. In my own opinion, measuring software engineering has its place, but only when strictly used as an evaluation tool to aid human understanding and judgement. Software engineers are human beings. They have lives outside of the work environment. As human beings, they are subject to good times and bad, times in which they may fall ill or suffer the loss of a family member or close friend. They are not machines and cannot be treated as such.

To measure the efficiency of a machine used to manufacture car parts, you might look at how many exhaust pipes are being produced per hour. The requirements for the machine to work might be electricity, routine maintenance, and raw materials as input. For a software engineer to work, they need a workstation, power supply, and internet access. But it also goes far beyond that. Software engineering is a cognitive task, and anything that impedes the ability to think clearly will affect performance that day, that week, that month. Be it an illness or a personal issue, there will be an impact.

If you were to explain the situation to a manager, they would empathise, understand, and be supportive. A system that measures performance will not be capable of making such allowances, and as such the true value of using development analysis platforms, AI and ML to evaluate software engineering performance lies in enhancing human decision-making.

I believe that value can be derived from collecting data about the productivity of software engineers as a way of supporting their work processes. However, someone's job should not hang in the balance as a result of the decisions and inferences made by an intelligent system, and as such, human intelligence must be used in conjunction with it.

The GDPR Perspective

General Data Protection Regulation (GDPR) came "into effect on May 25, 2018" (Wolford, 2021) with the aim of protecting personal information by introducing the "toughest privacy and security law in the world" (Wolford, 2021). The right to privacy is a pillar of the European Convention on Human Rights, and is protected under GDPR.

One of the principles of data protection that is covered under GDPR is data minimisation, in accordance with which "you should collect and process only as much data as absolutely necessary for the purposes specified" (Wolford, 2021). So, are companies collecting too much data?

Tracking your time

In my own view, one of the interesting points to be made pertains not only to the work that is being produced by engineers and the speed at which it is being produced, but more specifically to the work patterns of individuals. Time trackers such as Timeular are used by individuals as well as employees, garnering popularity among software engineers. They are being marketed as a way of boosting productivity, empowering users to "improve time management" (Timeular, 2021).

Using an eight-sided die, users can track the time spent working on various objectives by turning the die as they switch between tasks. My own view on this is that time trackers of this type are too invasive, that they provide a near microscopic view of how people split their working day. There is a fine line between ensuring that engineers are working productively and monitoring their actions during the working day. For me, this crosses that line and goes beyond what is necessary data for the purposes of measuring software engineering.

Conclusion

In my opinion, we cannot ignore that there is value to be found in measuring and evaluating software engineering performance and efficiency, but its application must be constrained. Perhaps in the next decade AI will be able to exercise human judgement and will replace the need for decisions made by humans in its entirety.

But we still have a long way to go. Until then, AI and human intelligence must continue to work together to evaluate the performance of software engineers in ways that create value and enhance the working lives of those engineers, not detract from them. Ultimately, we should be aiming for an approach that fosters diversity within teams, supports software engineers, and allows them to reach their full potential. There is still ground to cover, but in terms of what can still be achieved, the sky is the limit.

References

CHANAL, P. M., KAKKASAGERI, M. S. & MANVI, S. K. S. 2021. Recent Trends in Computational Research (Chapter 7: Security and privacy in the internet of things: computational intelligent techniques-based approaches).

CIRCEI, A. 2021. Waydev: Metrics glossary [Online]. Available: https://docs.waydev.co/en/articles/3971218-metrics-glossary [Accessed 17th December 2021].

CODE CLIMATE. 2018. Setting Up Test Coverage [Online]. Available: https://docs.codeclimate.com/docs/getting-started-test-coverage [Accessed 16th December 2021].

CODE CLIMATE. 2021a. Code Climate Documentation (Quality) [Online]. Available: https://docs.codeclimate.com/docs/getting-started-with-code-climate [Accessed 16th December 2021].

CODE CLIMATE. 2021b. Code Climate: Velocity [Online]. Available: https://codeclimate.com/velocity/ [Accessed 16th December 2021].

DANGER, J. 2021. You must not measure individual software engineer performance. Available: https://medium.com/dangerous-engineering/you-must-not-measure-individual-software-engineer-performance-3b68cc45cacb.

ELLEN, B. M., BRENNAN, M. P., EDITH, S. G. & RACHEL, A. 2015. Ethical and appropriate data use requires data literacy. The Phi Delta Kappan, 96**,** 25-28.

GATES, B. n.d. Measuring Software Engineering (Quote) [Online]. Available: https://www.goodreads.com/quotes/536587-measuring-programming-progress-by-lines-of-code-is-like-measuring [Accessed 17th December 2021].

GEEKSFORGEEKS. 2020. Software Measurement and Metrics [Online]. Available: https://www.geeksforgeeks.org/software-measurement-and-metrics/ [Accessed 17th December 2021].

GEEKSFORGEEKS. 2021. Lines of Code (LOC) in Software Engineering. Available: https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/.

HCL TECHNOLOGIES. 2021. What is Utility Computing? [Online]. Available: https://www.hcltech.com/technology-qa/what-is-utility-computing [Accessed 17th December 2021].

IEEE. 2021. What is Computational Intelligence? [Online]. Available: https://cis.ieee.org/about/what-is-ci [Accessed 20th December 2021].

KAYSER-BRIL, N. 2020. Female historians and male nurses do not exist, Google Translate tells its European users [Online]. Algorithm Watch. Available: https://algorithmwatch.org/en/google-translate-gender-bias/ [Accessed 20th December 2021].

PLURALSIGHT. 2021a. Flow: Work better together with deep insights into your engineering workflow [Online]. [Accessed 17th December 2021].

PLURALSIGHT. 2021b. What is Impact? [Online]. Available: https://help.pluralsight.com/help/what-is-impact [Accessed 20th December 2021].

PVS STUDIO. 2013. Source Lines of Code. Available: https://pvs-studio.com/en/blog/terms/0086/.

TIMEULAR. 2021. Unlocking time [Online]. Available: https://timeular.com/about-us/ [Accessed 20th December 2021].

VANCE, A. 2017. How Two Brothers Turned Seven Lines of Code Into a $9.2 Billion Startup. Bloomberg Businessweek.

WAYDEV. 2021. Waydev: Modern Data-Driven Engineering Management [Online]. Available: https://waydev.co/ [Accessed 17th December 2021].

WHITSON, G. 2020. Software Engineering. Salem Press.

WOLFORD, B. 2021. What is GDPR, the EU's new data protection law? [Online]. Available: https://gdpr.eu/what-is-gdpr/ [Accessed 20th December 2021].