Measuring Software Engineering

Cian Jinks - 19334781

Introduction

This is my report for the “Measuring Software Engineering” assignment as part of module CSU33012. Alongside this, I have created a visualizer for Github API data as a form of measuring software engineering. It visualizes contributions by user for a number of popular open source projects on Github using a scatter plot and pie chart. A link to it can be found here.

Before getting into details I feel it is important to clarify the definition of what measuring software engineering is as well as why it is an incredibly relevant topic for many companies and individuals.

To set the scene, let us imagine a simple example where we are the project manager for a large software engineering team working on a Project X. Project X has tight deadlines and targets which must be hit and we as the project manager are responsible for hitting them. This is the exact type of environment where it can be incredibly instrumental to implement some form of measuring software engineering. Specifically, we can gather metrics which relate to the performance of individual engineers within the team or even the overall project as a whole. We can then use this data by applying computations to it which allow us to draw inferences about the productivity of the project and/or engineering team such as identifying bottlenecks and various problems. This process is exactly the definition of measuring software engineering.

It is also clear to see from the example above how the need/demand for efficient ways of measuring software engineering is heavily motivated by a desire to increase or maintain a level of productivity on a project. One might point out that there are many other areas beyond productivity which measures of software engineering are designed to improve such as code complexity/readability, code performance, code validity, etc. However, at the end of the day the reason for improving each of these areas ultimately boils down to a likelihood for better productivity in the future.

Measuring of Software Engineering

In this section I discuss various simple strategies for measuring various metrics related to software engineering as well as their benefits and flaws.

Lines of Code Committed (LOC)

One of the most basic metrics one can come up with for measuring software engineering is “Lines of Code Committed” by each engineer as a measure of productivity. At a naive first glance we might infer that if one engineer has committed more lines of code than another then they must be the more productive engineer. However, this measure of software engineering is fatally flawed. Anyone who has programmed even a little bit understands that the number of lines of code you write is not any indicator of engineer productivity (nor code quality or validity). Consider the case where one engineer is writing code in C and another is writing code in Java. C being the much lower level language can often require many more lines of code to achieve the same exact functionality as in Java. Does this make the C programmer the more productive of the two engineers? No, of course not. Not to mention that it is trivial to pad out code with meaningless lines that do not change its functionality.

Does this mean that “Lines of Code Committed” is useless as a measure of any metric relating to software engineering? I would say *mostly*. However, I argue that the concept of LOC should not be thrown out the window entirely, as I think it is statistically likely that more productive coders *will* produce more LOC on average than less productive ones. It is just the impossibility of being certain that makes it not a very useful metric. **[1]**

Frequency of Code Commits

Similar to LOC, we have “Frequency of Code Commits”. Take for example two engineers working over a 30 day period. Naively, we may think the engineer who has the most commits within that period has been the most productive. But alas, this metric is flawed in very similar ways, if not the exact same, as LOC. What if for example one of the engineers had spent all of their 30 days devising an ingenious algorithm to solve a specific problem and had therefore only made one commit implementing this algorithm while the other engineer had committed 30 simple typo corrections? I suspect now many would say the engineer who made the 1 commit was more productive.

Once again however this metric is not entirely useless. Just like with LOC there is likely a tendency for the more productive engineers to be making more overall commits. We just cannot say for sure that this will be the case for any given time period.

Code Structure & Complexity

An ability to write concise and easy to read code is often the sign of a great engineer. Therefore, it can be important to have a way to measure software engineering code quality. One idea would be to analyze the complexity and structure of an engineer's code.

This approach is certainly not easy though. Firstly, it is guaranteed to be much more complex to measure than the two mentioned previously as it is hard to define what exactly constitutes good code structure and less or more complex code. We can see an example of the complexity in measuring this metric in the general structure of a source code analysis program diagram seen in **[2]**.

However, when the analysis tools for this type of software engineering measure are designed well, great quality information can be inferred from the data produced about a given software engineer's code.

Existing Platforms

There exists many platforms today which one can use to gather and perform complex and expensive calculations over data sets which contain measures of software engineering. In this section I discuss a handful of these from general to specialist and present the possibilities with each option.

Cloud Computing

Cloud computing platforms such as Amazon Web Services (AWS) **[3]**, Microsoft Azure **[4]**, and Google Cloud **[5]** are arguably the largest platforms available today which enable the efficient measuring of large amounts of software engineering metrics. Cloud computing is incredibly versatile in this area in that it allows companies to design incredibly detailed and complex algorithms of their own for measuring software engineering metrics and then run these algorithms with virtual "supercomputers" in the cloud. This is enabled by the ability of the aforementioned services to split compute workloads across many high end servers seamlessly and efficiently.

This type of platform is perfect for algorithms attempting to tackle the problem of measuring code quality and structure as mentioned above as these types of algorithms are often very intensive to compute whether it be machine learning or other.

Flow

As opposed to general cloud computing platforms as mentioned above, PluralSight's Flow **[6]** is a much more specialised platform with a *focus* on measuring software engineering specifically. As explained on their website, Flow measures software engineering by gathering data from git version control systems for a given project and running their own custom algorithms and analysis on that data before presenting it in an easy to understand format.
Flow enables many possibilities for viewing software engineering performance and productivity within a team. For example they advertise the ability to "See how much time is spent refactoring legacy code vs. new work", "Recognise project bottlenecks and remove them", "Know if code reviews are productive and positive" and much much more.

Notably it is highly likely that Flow makes use of the very same Cloud Computing platforms as mentioned above (or has their own) to apply their intensive computational algorithms on the data they gather. Therefore, this also gives us a great insight into the possibilities that this type of infrastructure enables within the area of measuring software engineering.

<u>Algorithms</u>

Machine Learning

When it comes to tackling complex subjective problems such as determining code quality, readability and complexity, machine learning is one of the most popular algorithmic styles one can choose. Current machine learning algorithms offer multiple ways to tackle a given problem such as “Reinforcement Learning” **[7]** vs “Supervised Learning” **[8]**.

In Reinforcement Learning the program makes guesses at what the correct answer to a problem is and then adjusts its parameters based on how accurate its guess was. On the other hand, Supervised Learning is where inputs are given to the program with the correct solution included and it builds an internal model based on this data to make informed guesses about unknown inputs.

We can easily imagine how Supervised Learning can be used to teach a machine learning model to determine code readability given hand-crafted input data. This does not solve the issue of determining what constitutes quality and readable code mentioned above; however it allows us to train a model to automate a subjective task efficiently.

<u>Ethics</u>

When it comes to measuring software engineering today, the most talked about point of all is definitely the problem of ethics, morality and legality. Many of the methods and topics discussed previously have specifically emphasised their applicability to measuring a *single* software engineer’s performance, skill, productivity, etc. When discussing the theory behind measuring software engineering this is all fine and good, however when we decide to actually go and apply our techniques in the real world we must absolutely consider how the information we gather and inferences we make from it, may be affecting the mental health of the individual engineers and the legality surrounding said data about their engineering habits and work.

Take for example a totally imaginary situation where a big tech company is making use of their own techniques to gather software engineering metrics on the performance of their engineers. At the end of each month, the engineers receive a report outlining their performance, productivity, code quality, etc for that month. Implementing this type of system may seem perfectly logical at a first glance as it encourages engineers to improve upon areas that they are worse at. However, what you may not consider is that this would actually be adding significantly increased stress into their daily lives if they are consistently underperforming and consistently being told that they are. One can only imagine the mental health impact this can have. Therefore, it becomes an ethical dilemma of wanting your engineers to improve vs wanting to ensure their mental health. I for one would choose the mental health option.

Aside from mental health, there is also the problem of discrimination in the work place. It has been shown in the past that attempting to measure metrics about a person's performance, skill, etc using numerical data such as mentioned before leads to biases against minority groups. This is something that I would expect any company would want to *avoid*.

Luckily, there are many legal protections in place these days to attempt to stop problems like this from arising. Take GDPR in Europe for example **[9]**. GDPR laws protect a person's personal data from being identifiable to them. This does not stop the collection of data for measuring software engineering purposes, but instead it ensures that any data collected can only be used to measure general information about a team or larger group and not about a single individual. However, this is not perfect. There is a gray area in which all data gathering and computation is done by a computer without any external person viewing it. In this case a report could still be returned to an engineer about their performance, which can obviously still lead to mental health problems as discussed above.

To conclude, I believe the area of measuring software engineering has two paths ahead. One where algorithms are further refined and developed and it becomes standard in every workplace to boil a person down to numerical metrics and data which imply performance or productivity and another where performance and productivity is measured subjectively by managers and colleagues. Regardless of which path is chosen ethics must always be considered in either case to avoid mental health issues in the workplace.

References / Reading Material

- [1] - https://www.pluralsight.com/blog/teams/lines-of-code-is-a-worthless-metric--except-when-it-isn-t-
- [2] - https://www.researchgate.net/publication/228655465_Source_Code_Analysis-An_Overview
- [3] - https://aws.amazon.com/
- [4] - https://azure.microsoft.com/en-us/
- [5] - https://cloud.google.com/
- [6] - https://www.pluralsight.com/product/flow
- [7] - https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- [8] - https://www.sciencedirect.com/topics/computer-science/supervised-learning
- [9] - https://gdpr.eu/