Trinity College Dublin

The University of Dublin

# Measuring Software Engineering

## CS33012 Software Engineering


## Michael Sweeney

**18325015**


## Table of contents

## 1. How can software engineering be measured?

**Introduction**

There are vast quantities of data that can be gathered on individual software engineers. Modern software is becoming more complex at an ever-increasing rate. software engineering has turned into a profession focused on boosting the rate at which work is completed and improving work efficiency and quality. There have been many different approaches on how to best measure the weight of the software engineering process. This report will examine the various methods used to establish whether a software engineer is being productive, what affects their working quality and what motivates them to work harder

### 1.1 Why should we measure software engineering?

Although measuring this data can be difficult, measuring what a software engineer does can be very valuable. Engineers need feedback so they can improve their skills and deepen their knowledge, but it is important to measure accurately and to measure exactly what you need and in the right ways. They need to be personalized as a one size fits all approach could be harmful to the employee.

### 1.2 What can be measured?

To measure a software engineer we will have to look at some raw data involving the specific engineer; however, this data could be very easily misinterpreted so it is very important to evaluate a larger quantity of data to decrease the dependencies on single metrics. So for the majority of cases, we'll be looking at the output data of an engineer

### 1.3 Measurable Data

Below I will go through the individual pieces of data that could be measured to evaluate a software engineer. In my opinion, I would not take any of these metrics on their own and would evaluate my engineers depending based on putting all this data together.

Counting the Lines of Code

Counting the Lines of Code

One approach to measuring a software engineer would be to count the number of lines of code produced. Advantages to this metric are that it is easy to count and keep track of and could be automated easily. This however would encourage the process that "the more lines of code a programmer has written the better the programmer". There are many flaws with this method as it would require measuring how many lines someone would produce, but not examining the format of how the program was written and completely disregarding the performance of the program.

```java
public class Main
{
    public static void main(String[] args) {
        for(int i=0;i<10;i++){
            System.out.println("Hello World"+ i);
        }
    }
}
```

```java

public class Main
{
    public static void main(String[] args) {
        int i=0;
        int n=10;
        while(i<n){
            System.out.println("Hello World"+ i);
            i++;
        }

    }
}

```

In the example above although the code on top might be praised by this measurement, it wouldn't take into account that the shorter one is more efficient and overall better written. It could lead to individuals working in the field working only to trick the system and pad their performance statistics by coding very long inefficient code.

Another method of measuring the amount of code written would be to measure the logical lines of code. This would measure the number of statements in the code, so it would ignore the number of lines written to a point. Doing this would fix some issues with just counting raw lines. But unfortunately, this method can be cheated with.

Hours Worked

This is another basic piece of data that could be measured, despite it being able to give a quick observation as to how long a project must be worked on, to be completed. Although this metric could give a good intuition as to how long a certain problem has taken. It is better to be combined with other data to be more useful. As sometimes an engineer can't be consistent with the amount done in a certain number of hours.

Code Quality

To determine if code quality is good, we must consider the key metrics that many software engineers agree on, those are reliability, maintainability, testability, portability, and reusability. Being reliable enough so that it runs without errors or warnings over a certain amount of time. Being maintainable refers to how hard it is to keep this piece of code maintained and running, to do with the consistency, indentation, and structure of the code. Many metrics would have to be considered for this. Testability refers to how easy it is to test different areas of the code to find certain failures or faults. How much would be needed to test all cases? The easier and shorter the testing is the higher the score. Portability is how independent the software is meaning how easy would it be to transfer to different environments without breaking components. The easiest way to measure this would be to make sure it works in a neutral environment such as docker and to continuously try it out on different platforms. Using this scoring process would be a good way to determine and vastly improve the quality of the software that is being measured. It considers a vast number of aspects and gives a better understanding of the performance of individuals. In my opinion, although this way of measuring could be subjective it could be tailored to companies needs and prove to be very valuable.

Impact

We can measure the impact that code has on a project, the larger the impact a software engineer has on a project the more likely he's a more valuable member of the team. For example, adding 50 new lines to a project could have much less impact than a change that reduced the number of lines in the original project. Making changes to reduce the bugs in a project or deleting redundant lines could be measured as a benefit to the project. We could measure it in a way that the impact score is higher if the number of code changes, the severity of those changes and the number of files affected by the efficiency of the code.

Number of times code was committed

Another piece of data that could be used to measure software engineers would be to count the number of commits made. Measuring this would be measuring the changes made to a project, however, their size nor their frequency correspond with the work needed to commit that change, thus showing that programmers' commits should not be compared.

Bug Fixes

A method to analyse the quality of the code would be to check the number of bugs/errors in each line of code. When developing a product from square one, it can be difficult to avoid mistakes and bugs. These mistakes could lead to major delays and end up costing the company a lot of money. The more errors an engineer could prevent the more productive they can be leading to less time looking for fixes.

Eradicating errors in the initial time of building a project will lead to smoother project implementation. This metric will reward engineers willing to double-check their work and fix their mistakes. Evaluating these defects would be key if defects arise. Using this metric could lead to engineers being more precise and careful with their projects. Although it could also lead to less creative methods. If they are being solely measured on this metric, they could spend way more time fixing other peoples work rather than adding to the project.

Health and Office Climate

Keeping track of variables such as the engineer's health and wellbeing and what climate produced the most benefit to the employer could be useful. This could allow for a more comprehensive breakdown of their performance.

Keeping track of an engineer's health could be beneficial, as it would prevent and diagnose health problems, while also getting the best work out of an employee. Observing this data could benefit the employee as the employer could recommend ways to improve posture, stress rates, breathing techniques.

Improving an office climate could have a major impact on the employee's work ethic and life outside of work. Measuring how the working environment is impacting software engineers could be useful to assess. Variables such as the office chairs and height of the desks, the amount of natural light in the office, the food given to employees for lunch and what temperature the office climate should be kept at. Any way to get the best out of an individual employee. Giving sufficient breaks and having conversations between co-workers could be analysed to see how focused colleagues are in their work. This interaction could help improve team dynamic as well as make working collaboratively easier

## 2. Computational Platforms to Collect and Analyse Data

In recent years the number of platforms in which one can gather metric data on software has grown exponentially. A quick search on Google and many websites can gather and perform the calculation on metric data sets. The likes of GitHub can be used as an example with most software engineers work revolved around committing their work to GitHub. Useful information can be extracted to learn more about their workflow and the quality of work being committed. With this vast amount of data being at our fingertips, it has enabled us to perform complex and computationally expensive calculations. In this section I will describe these useful platforms and what information they give; I will also go through the benefits and disadvantages of each platform.

### 2.1 GitHub

Lots of companies in the software industry have their codebase on website repositories such as GitHub. With more than 100 million repositories on GitHub, GitHub can give

us valuable insights into many projects and how they are being worked on. If observed correctly they could provide some useful metrics about individuals and how they work on projects.

If we investigate commits, we can examine the output and look at data such as the amount of code written/deleted. Sometimes though these statistics could give a false insight into individuals. For example, mistaking someone who writes the most lines as being a valuable developer, even though his commits have been full of errors. Though situations like this may arise it's important to note that having these records would make it quite easy to track down that said developer and improve the code with feedback or resolve an issue with their contribution.
**(GitHub)**

### 2.2 Pluralsight

Pluralsight describes itself on its website as "aggregates historical Git data into easy-to-understand insights and reports". Companies can easily receive topical data that they can use to show how progress on a specific project is going. Pluralsight advertises itself as a company that wants to save time identifying bottlenecks and comparing releases over time. They focus on historical data as past performances predict future performances.

Pluralsight proves a very useful tool for many companies as the majority of engineers uses git to build projects. The tools collect a plethora of data and can combine and compare historical data on any engineer in a non-bias way. They will try to match data patterns and extract the engineer's strengths and weaknesses. Pluralsight provides companies visibility into workflow patterns to accelerate the production of projects.
**(Pluralsight)**

### 2.3 Trello

From a software engineers' point of view, it has become much easier to keep track of their work. One such tool that has proved useful for software engineers is Trello. Its popularity with software engineers has recently heightened to new levels thanks to its

visual list creation tool that allows them to keep track of tasks and TODOs straightforwardly. By creating cards with tasks to do and tasks that are completed it can be easy to track progress, store information and projects and organise and assign work.

It's a simple Kanban-based task management tool that helps collaboration between teams become more seamless than ever. For companies on a budget, project managers can use it to organize and monitor progress at a glance. Many might pass it up because of its basic layout but its simplicity is its strength. It is perfect for all parts of development from planning and setup to the final implementation.
**(Trello)**

## 3. Algorithmic Approaches

In recent times, we have accelerated the kinds of computation that can be done with software engineering data, to get a good idea of how a software engineer is performing. There are various techniques for computing this data such as basic counting, various software engineering algorithms concepts and many more. In this section, I will attempt to explain two examples of these algorithmic approaches.

### 3.1 AI/ML Analysis

Artificial Intelligence (AI) and Machine Learning (ML) are increasingly being used to analyse large quantities of data, especially looking at patterns in software engineers' work. This data is being coupled with statistical analysis to recognise these statistical patterns so that companies can better predict how long projects will take from historical records.

Machine Learning studies computers algorithms to better improve automatically over time. With the millions of commits made every year these systems have mountains of data to learn and improve from. The ever-increasing intelligence of computational systems has made it even easier for companies to assess individual engineers, and as this kind of analysis gains popularity, it will be interesting to see how it will change our workflow and how projects are completed in the future.

### 3.2 Halstead's Software Metrics

Maurice Halstead in the 1970's introduced metrics to measure software programs and their complexity. In these metrics a program is a collection of tokens, which can be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token. The basic measures are. Below I will give a basic overview of the algorithm.

n1 = count of unique operators.
n2 = count of unique operands.
N1 = count of total occurrences of operators.
N2 = count of total occurrence of operands.

In terms of tokens used, a program size can be expressed as:
N = N1 + N2.

Halstead metrics are:
Program Volume (V): The size of the program.
V=N*log2n

Program Level (L): The level of complexity ranging from 0 to 1.
L=V*/V

Program Difficulty (D): The error-proneness of a program.
D= (n1/2) * (N2/n2)

Program Length: The number of unique operators and operands.
N=N1+N2

Estimated Program Length: Alternative expression
$\hat{N}$ = n1log2n1 + n2log2n2

Potential Minimum Volume: the shortest program in which a problem can be coded.

$V^* = (2 + n2^*) * \log2 (2 + n2^*)$

$n2^*$ = the count of unique input and output parameters

Size of Vocabulary (n): the number of unique tokens used

$n=n1+n2$

n=vocabulary of a program

n1=number of unique operators

n2=number of unique operands

Language Level – The program language level.

$L' = V / D / D$

$lambda = L * V^* = L2 * V$

Programming Effort (E): The unit of measurement of E is elementary mental discriminations.

$E=V/L=D*V$

To get the final coding time used to measure a program we can use the formula

$T=E/18$

Although Halstead metrics were used frequently in the late 20th century and were instrumental in calculation the complexity of software programs. They are more often used for maintenance metrics nowadays.

It still proves to be valuable during development when combined with other algorithm approaches to give a more accurate understanding of a software's complexity (JavaTPoint "Halstead's Software Metrics").

## 4. Ethical Concerns

Whilst some of these methods can be exceptionally effective for a company focused on software development, they also bear some severe ethical and legal concerns as well as moral issues surrounding the handling of this kind of personal data.

With more and more devices involved in our day-to-day life's, many people are becoming increasingly worried about their digital footprint, knowing the data harvesting that is occurring outside of work. Introducing a system or method that would cause developers to feel like they are being constantly watched would almost change how developers acts. It would burden them with even more challenges on an already stressful, deadline heavy job. Developers could become consumed and channel all their work towards meeting certain quotas instead of focusing on a quality product in which they respect.

With many large companies making big decisions off data analysis, there are monumental concerns about ethical considerations. These range from gathering data unbeknownst to the software developer, making them feel like they are being constantly being monitored. Although gathering data on the code being written is of little concern to the developer, it's the analysis of how that code was written that is worrisome, this data could be manipulated by the software engineer to appear more productive or could be unfair to the way the engineer works.

On data relating to personal and health wellbeing could be dangerous. It could allow employers to gain control in an employee's life outside of their workplace. Should an employee have the right to control an employees life choice?

Some of the data (being as sensitive as it is), should be stored as securely as safely as possible. Given its sensitive nature if there was a data breach it would lead to massive data about their lives being on the internet for the whole world to see. Proper steps should be taken to safeguard this information.

## 5. Conclusion

In this report, I have gone over the main areas to do with measuring software engineering. I gave assessed the ways and processes in which data can be measured, I gave an overview on how these metrics can be used with computational platforms and the algorithmic approaches used to give accurate representations of the data and finally we ran through the concern of ethics and legality of the measuring process of these kinds of analytics.

## References


https://www.youtube.com/watch?v=cRJZldsHS3c

https://www.tutorialspoint.com/management_concepts/management_concepts_quick_guide.htm

https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality

https://www.pluralsight.com/product/flow

https://trello.com/

https://github.com/

https://www.javatpoint.com/software-engineering-halsteads-software-metrics

Halstead complexity measures - Wikipedia

https://codeql.github.com/publications/measuring-software-development.pdf

https://digital.ai/catalyst-blog/4-devops-metrics-to-improve-delivery-performance

https://stackify.com/track-software-metrics/

https://www.youtube.com/watch?v=icR4_qbyVt8