# Measuring Software Engineering

## Andrew Mc Donald – 18318748

Table of Contents

[Measuring Software Engineering 1](#_Toc57376096)

[Andrew Mc Donald – 18318748 1](#_Toc57376097)

[**Introduction** 1](#_Toc57376098)

[**Measuring Processes and Data** 2](#_Toc57376099)

[Code 2](#_Toc57376100)

[Tests 3](#_Toc57376101)

[Time 3](#_Toc57376102)

[Function and Story Points 4](#_Toc57376103)

[Others 5](#_Toc57376104)

[**Platforms available** 5](#_Toc57376105)

[Plurasight Flow 5](#_Toc57376106)

[Velocity 6](#_Toc57376107)

[Hackystat 7](#_Toc57376108)

[**Algorithmic Approaches** 7](#_Toc57376109)

[Level 1 - Basic 8](#_Toc57376110)

[Level 2 - Intermediate 8](#_Toc57376111)

[Level 3 - Advanced 8](#_Toc57376112)

[Mode 1 – Organic 8](#_Toc57376113)

[Model 2 – Embedded 8](#_Toc57376114)

[Model 3 – Semi-Detached 9](#_Toc57376115)

[**Ethical Concerns** 9](#_Toc57376116)

## **Introduction**

Measuring the productivity of individual engineers is a difficult and complicated task. So much so that some people argue that it can’t be quantified accurately at all. One would generally consider productivity to be one’s ability to produce products but with software is not so simple due to the absence of a production line.[[1]](#footnote-1) There are many intricate processes that are unique and tailormade to the scenario or clients’ wishes. Regardless, it is, of course, important that employers evaluate employees for many reasons.

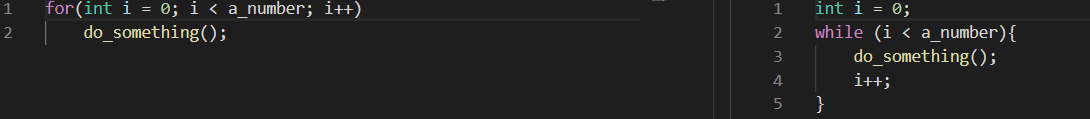
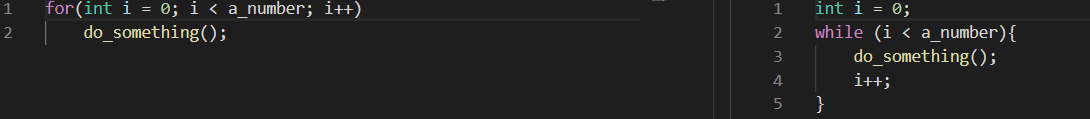
In this report I will explore the processes by which data is collected, the platforms available to employers to gather this data, the algorithmic approaches as well as the ethics surrounding the collection and analysis of such data.

## **Measuring Processes and Data**

While it is argued that the overall productivity of an engineer can be difficult to measure it is clear and obvious that there are many individual metrics that can be, and are simple to be, measured. These include code quantity, tests created, time taken and function points.

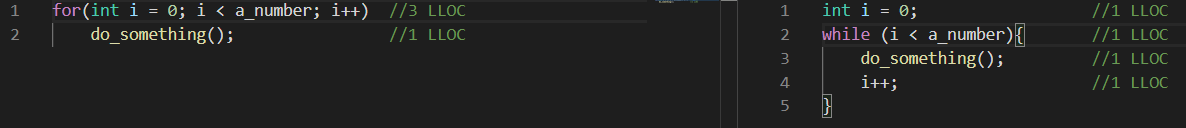
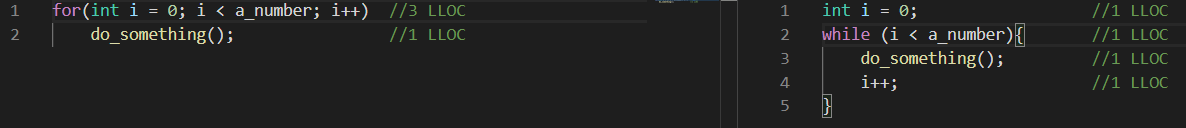
### Code

This is initially the simplest way to measure the amount of work done by an individual software engineer. The concept requires that you count the number of lines of code produced by the engineer[[2]](#footnote-2). I said “initially” because this metric has several issues. For example, the following snap shows two snapshots of code that do exactly the same thing yet have a different number of lines

. 

As you can see, they are identical yet solely judging by lines of code written the coder on the right has done more than twice the amount of work as the coder on the left.

Another approach one might use could be to count Logical Lines of Code (LLOC). LLOC attempts to measure the number of "statements", but their specific definitions are tied to specific computer languages.[[3]](#footnote-3) Take a look at the same snippets of code with the number of logical lines of code implemented and you will notice they’re the same. LLOC overcomes some of the issues faced by simply counting lines.[[4]](#footnote-4)

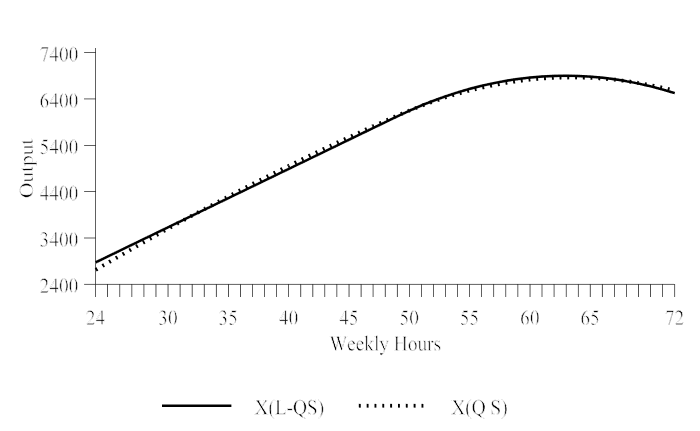
This metric is not perfect, however. As well there being no transparency, there can be much debate as to what constitutes one LLOC.

### Tests

Another metric to test an engineer’s productivity would be to measure code tested. By measuring what percentage of the code written is used in the various tests created by other users. While it’s certainly preferable to keystrokes among other metrics there are a number of problems associated. One would be that it, unsurprisingly, relies on the quality of the tests. Another would be that this metric gives no indication of the efficiency or scalability of the code written.

### Time

The overall time spent on a project by an engineer is problematic as not all time spent is necessarily actively coding. Especially with so much work being done from home if someone wanted to up their time spent on a project, they could simply open the files and walk away from their computer. Often times, even if someone does work significantly longer hours the quality of that work is so poor that one would be better off not working at all. This was proven in 2014 by Professor John Pencavel working out of Stanford.[[5]](#footnote-5)



This graph from that same study shows the quantity of units outputted by men and women turning fuse bodies (unbroken line) and milling screw threads (broken line) per hour. As you can see, at around 55 hours, the rate of efficiency starts to drop[[6]](#footnote-6).

### Function and Story Points

For measuring a code’s value function points and story points are a good metric to use. Function points are a measure of the functionality delivered by the project or application.[[7]](#footnote-7) They are often accepted as the industry standard.

Story Points are a measure unit resting on the perception of the work to be done by the project team. The determination of that size is based on the level of comprehension of the complexity, and thus, the required effort.[[8]](#footnote-8) Story points can be implemented faster than function points. As story points are a collaborative effort by the team, they are unique to each project and each team. This makes them more accurate at evaluating a team’s progress than the boilerplate function plates.

### Others

Some of the most important factors in a team or individual’s efficiency cannot be quantified. Leadership skills and the ability to direct a team, for example, wouldn’t show up by any amount of study of the code yet a good leader can be the make or break of a project not to mention the colossal impact on efficiency a solid direction has.

Helpfulness and ability and willingness to assist other members of the team are another huge part of an engineer’s efficiency to the overall team. What is more is that it is another factor that wouldn’t show up solely by reading the code. A helping hand from a member of the team can quickly solve a problem that the original engineer was stuck on. A fresh set of eyes is sometimes all that is needed especially in large scale projects. There are of course countless more ways to measure a software engineer as well as external influences on their productivity these are simply two measurements that one mightn’t first expect.

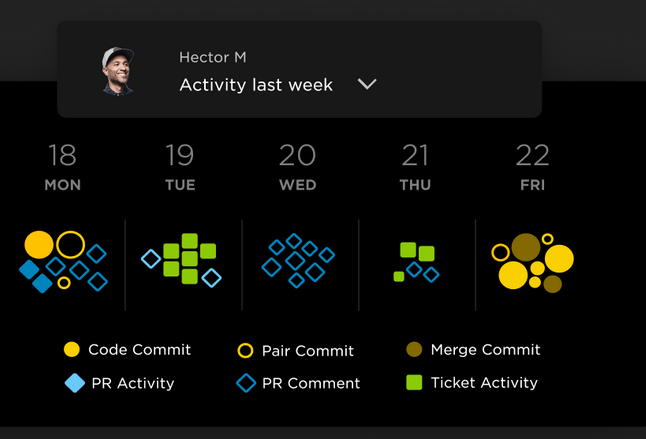
# **Platforms available**

As outlined, there are many metrics that affect a programmer’s efficiency and output. When it comes to measuring certain metrics there are many different computational platforms that aid in the measurement and analysis of software engineers and their work.

### Plurasight Flow

Plurasight’s Flow takes data from different version control systems and analyses it to gather insights on the engineers.[[9]](#footnote-9) They analyse a large number of variables surrounding the code too so not only do they collect the measurable data I discussed earlier, such as logical lines of code and function points, they gather insights into the interactions between the engineers. So, if one engineer tends to edit code written by another engineer a little too frequently Flow might suggest they have conflicting styles.

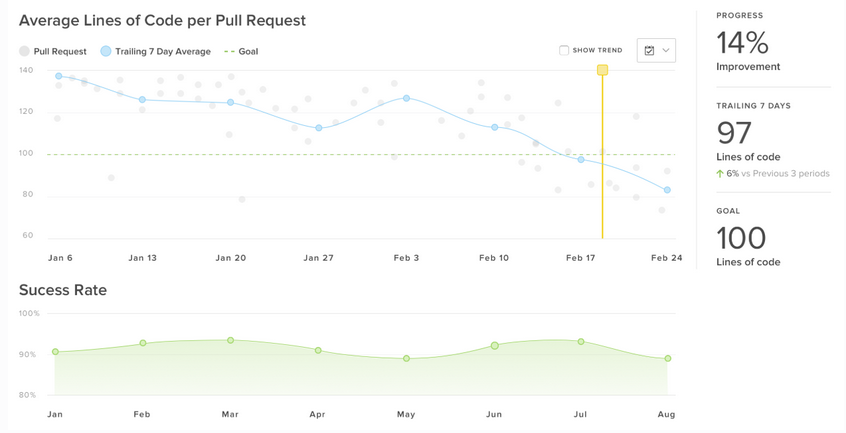
With the measurable data from individuals they can provide data on the performance of a specific engineer within a team. As previously discussed, while the numbers don’t lie, they certainly don’t tell the whole story.

[[10]](#footnote-10)

A view of Flow’s GUI.

### Velocity

Velocity is another analysis service similar to Flow where it reads the commits in a project. Where they differ is in velocity’s goal. Velocity attempts to find the bottleneck in efficiency. Be it a certain engineer in the team or, more often, in a specific department or area of the program. They also focus on visualising the data in a way that’s easier to digest for people with little or no experience in the field. This helps map progress over time.

[[11]](#footnote-11)

### Hackystat

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data.[[12]](#footnote-12) Hackystat users typically attach software ‘sensors’ to their development tools, which unobtrusively collect and send “raw” data about development to a web service called the Hackystat SensorBase for storage. [[13]](#footnote-13) Since the framework is designed to be set up by the user and can gather data from development tools it allows for analysis of documentation and other work done separate from the software.

## **Algorithmic Approaches**

In previous sections I have discussed the idea of measuring an engineer’s performance based on data accumulated. These metrics are easy to gather but what you can extrapolate from that isn’t always the best approach. If a company wants a more in-depth analysis for a project then a more complex algorithmic approach may be best.

While there are many, many models that one can use but for the purpose of this report I will focus on the COCOMO model. The Construction Cost Model is a regression model based on lines of code.[[14]](#footnote-14)

The COCOMO model follows the following hierarchy of levels. Basic, intermediate and advanced. [[15]](#footnote-15) As well as three separate modes, Organic, Embedded and Semi-detached, which allows it to be used in various engineering projects.

### Level 1 - Basic

The Basic COCOMO model is a static, single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC).

### Level 2 - Intermediate

The Intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel and project attributes.

### Level 3 - Advanced

The Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

### Mode 1 – Organic

This mode is most suitable for small scale projects where not many snags are expected.

### Model 2 – Embedded

A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. For Example: ATM, Air Traffic control.[[16]](#footnote-16)

### Model 3 – Semi-Detached

A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed.[[17]](#footnote-17)

## **Ethical Concerns**

The final aspect of measuring software engineering I will explore is ethics. In an increasing digital world privacy and other ethical concerns can be overlooked in the endless pursuit of efficiency.

Firstly, we have the concerns regarding data gathering. Initially with the original metrics discussed. Lines of code and tests written are really of no ethical concern as they are made public by the engineer. The other metrics mentioned such as leadership and helpfulness are a different matter. These are overall personality traits which can bring into question how ethically sound the collection of that data can be. Employers might be able to influence their life outside the job. Take the Fitbit or other smart watches, for example. While data collected on those would certainly influence an engineer’s ability to do their job, an employer could notice that an employee had not been exercising recently. Surely, an employer has no right to that data.

Secondly, would be the concerns regarding the analysis of this data. Entrusting the analysis of an engineer’s work to a machine is unsavoury for a lot of people, specifically outside of the IT industry. One major issue with the algorithmic analysis is that you aren’t guaranteed the correct result. Recently, scientists at MIT's LabSix, an artificial intelligence research group, tricked Google's image-recognition AI called InceptionV3 into thinking that a baseball was an espresso, a 3D-printed turtle was a firearm, and a cat was guacamole.[[18]](#footnote-18) While these algorithms are of course constantly improving they aren’t at the stage where they can be trusted wholly.

## **Conclusion**

Overall, measuring software and software engineers can be a complex task. First of all, while individual metrics themselves aren’t difficult to measure, there are some factors that simply cannot be quantified such as leadership and helpfulness qualities. Also as discussed, there are some relevant metrics that should stay private.

With enough quantifiable factors and the correct tools, it is possible to build a profile of an engineer’s productivity, efficiency and overall performance but a profile is only a snapshot into their ability. There is no one tool or metric to determine the overall quality of a software engineer.

1. https://www.devteam.space/blog/how-to-measure-developer-productivity/ [↑](#footnote-ref-1)
2. https://en.wikipedia.org/wiki/Source\_lines\_of\_code [↑](#footnote-ref-2)
3. http://www.projectcodemeter.com/cost\_estimation/help/GL\_sloc.htm [↑](#footnote-ref-3)
4. https://www.aivosto.com/project/help/pm-loc.html [↑](#footnote-ref-4)
5. https://www.cnbc.com/2019/03/20/stanford-study-longer-hours-doesnt-make-you-more-productive-heres-how-to-get-more-done-by-doing-less.html [↑](#footnote-ref-5)
6. http://ftp.iza.org/dp8129.pdf [↑](#footnote-ref-6)
7. https://www.tutorialspoint.com/estimation\_techniques/estimation\_techniques\_function\_points.htm [↑](#footnote-ref-7)
8. https://www.estimancy.com/en/2018/09/25/story-points-vs-function-points/ [↑](#footnote-ref-8)
9. https://www.pluralsight.com/blog/platform/pluralsight-flow [↑](#footnote-ref-9)
10. https://www.pluralsight.com/product/flow/engineers [↑](#footnote-ref-10)
11. https://codeclimate.com/blog/velocity-2-is-here/ [↑](#footnote-ref-11)
12. https://hackystat.github.io/ [↑](#footnote-ref-12)
13. http://freshmeat.sourceforge.net/projects/hackystat [↑](#footnote-ref-13)
14. https://www.geeksforgeeks.org/software-engineering-cocomo-model/ [↑](#footnote-ref-14)
15. “Optimizing Basic COCOMO Model Using Simplified Genetic Algorithm” by Rohit Kumar Sachan, Ayush Nigam, Avinash Singh, Sharad Singh, Manjeet Choudhary, Avinash Tiwari, Dharmender Singh Kushwaha [↑](#footnote-ref-15)
16. https://www.javatpoint.com/cocomo-model [↑](#footnote-ref-16)
17. https://www.javatpoint.com/cocomo-model [↑](#footnote-ref-17)
18. https://mashable.com/2017/11/02/mit-researchers-fool-google-ai-program/?europe=true [↑](#footnote-ref-18)