



Trinity College Dublin  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

---

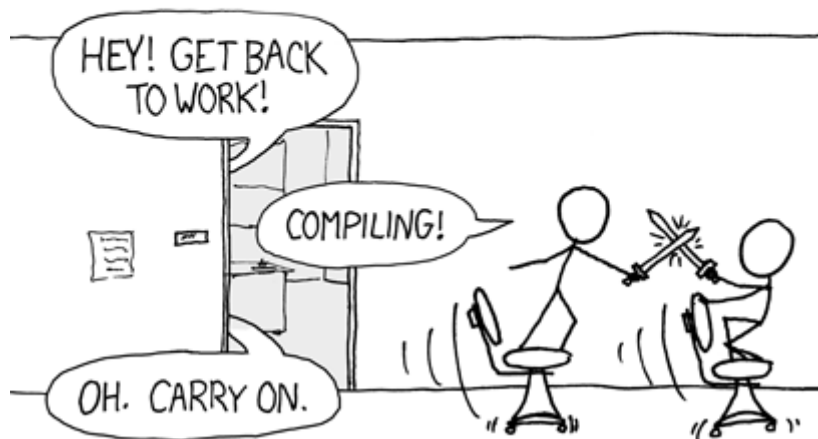
CSU33012: Software Engineering

# Measuring Software Engineering

Bernadine Lao 19333627

---

THE #1 PROGRAMMER EXCUSE  
FOR LEGITIMATELY SLACKING OFF:  
"MY CODE'S COMPILING."



<https://xkcd.com/303/>

# Table of Contents

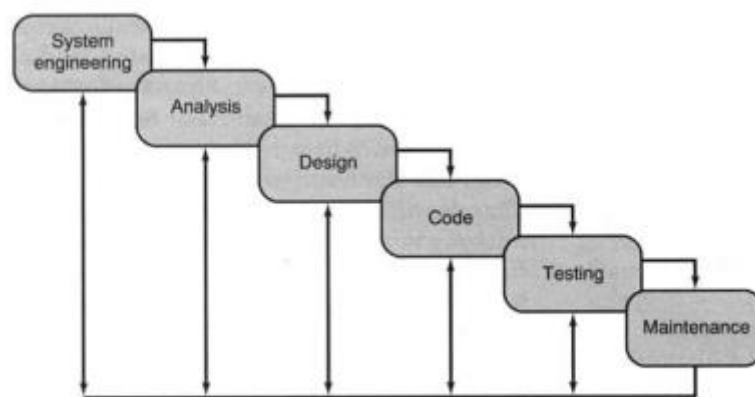
Introduction.....	1
Software Engineering Process.....	1
Measurable Data.....	2
Motivations.....	2
Productivity of a Software Engineer.....	2
Productivity in Teams.....	4
Productivity in Programming Languages.....	5
Software Engineer Stereotype.....	6
Development Analysis Platforms.....	7
PluralSight.....	7
Code Climate.....	8
Timeular.....	9
Algorithm Approaches.....	9
Cyclomatic Complexity.....	9
Halstead Volume/Effort.....	10
Shao and Wang's Cognitive Functional Size.....	11
Ethical Concerns.....	12
Conclusion.....	13

# Introduction

We have become part of an age where big data is generating a lot of information. As a community who constantly obsesses with data, analytics, and testing. It seems only natural that we would want to measure how efficient we are when coding and to track our progress. It is quite a dilemma how we can measure a software engineer's efficiency or productivity.

I am with the notion that software engineering is measurable. Just like how we can survey population annually and extrapolate data from it, despite knowing the general trend of the sample, there are some hidden variabilities and a lot of generalisation in it. For example, some people might have not been surveyed or a survey has been done before an outbreak. Complex data such as measurement of software engineering has some unaccounted variables and hidden errors.

## Software Engineering Process



Also called as the Software Life Cycle, I think it is important to know how much work has been put in each project made by teams of software engineers. Metrics aside, this “waterfall model” that a lot of software engineers in the past used for a systematic and sequential approach to software development, depicts numbers of hours in front of a computer screen constantly writing and rewriting codes to meet the goal and finish the code. As someone who has done just a fraction of what the engineers in the field do, I sympathise and appreciate their work.

The thought of some machine analysing my work is giving me mixed emotions. How can a machine quantify my productivity? From lines and lines of code? Is this detrimental to how I view my colleagues?

# Measurable Data

## Motivation

Engineering is a process therefore it is measurable. With sufficient reliability, measuring software engineering can be an extremely valuable tool to make judgements to performance, targets, and investments. Measurement is quantifying from the empirical world to the relational, formal world.

Software engineers are some of the most who leaves footprint of data daily. It is impossible not to collect their data. There can be no measurable improvement without measurement or metrics. The recognition for metrics is not just applied to software engineers but to all workforces. A company might allocate funding for an improvement programme if they see a need for it from the result of the measurement. Many studies have been published that improvement programmes have critical factors on its success: purpose, clear and measurable goals, trained resources, and reports.

Despite all the success and improvement measuring software engineering can bring, they are not mandatory and are an exception. Software engineers are not one to be afraid in being measured. The problem lies in ambiguity in the measurement where if they are being measured by the lines of codes, their working methods might be compromised and be forced to lower the quality of the code in exchange for “increase” in the measurement of productivity. Some metrics, just like my previous example is a waste of time and money making it cost-inefficient as engineers will work against their goals and it might promote mediocrity in the quality of their projects.

Using metrics that are insufficiently understood, inadequately validated and are not transparent to what data are being collected, can induce dysfunction within the workplace.

## Productivity of a Software Engineer

Measuring all software engineers individually in a company might be costly and not good long term but a software engineer can measure themselves to see how they are faring. Here are some ways an individual software engineer can measure their productivity.

### *By lines of code*

Bill Gates once famously once said, “Measuring Programming Progress By Lines Of Code Is Like Measuring Aircraft Building Progress By Weight.” Someone outside Software engineering or someone who does not code might think that it makes sense that the more lines of code you have, the better, more productive, and more hardworking you are. However, most, if not, all developers understand that the higher lines of code, there might be a higher chance of potential bugs, which would then need higher maintenance costs.

---

N.E. Fenton and S.L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach", Second 2nd edition revised ed. Boston: PWS Publishing, 1997.

<https://www.bcs.org/articles-opinion-and-research/the-importance-of-measurement-in-software-process-improvement/>

Assessing a software engineer's productivity by the lines of code is like judging a fisherman by the number of fishes they caught. Software engineers have different roles in the industry, some creates new features on an app, some does unit testing and some checks for bugs. Hence, someone who debugs code would not have a lot more codes to write up compared to a developer as it is not their job to do so.

### *Hours of Work*

Again, just like from the previous section, a naïve way of measuring productivity would be how long an engineer works. It does make complete sense to reward people who continue to work for longer periods than those who leave right before 5pm. However, in my opinion, this is a dangerous way to measure a software engineer's productivity. Engineers who used to work efficiently and fast will slow down their productivity to try to "increase" their productivity according to this metrics. It does not reward competency and steadfastness in the workplace and progress will stagnate.

It is important in any workplace to balance quality over quantity. An experienced engineer might be able to complete a task in an our whilst a new engineer might take a day to do so. If hours of work are the basis of productivity, then is the latter more productive? Well, personally, it is both yes and no. No because, whilst the more experienced engineer might have just taken a short time to complete the task, they are more efficient and can continue to work on another. This work ethics is important for the production both in the short and long term. Yes, because a newer engineer might take hours to finish the tasks, constantly learning from their mentors and from the internet, but also building up their skills at the same time. And in the long run, rewarding this type of behavior will encourage progress from those who are inexperienced and will be just like the experienced engineer from the example.

Of course, this example is just one of the scenarios in a workplace, but I think this kind of metric cannot be automated and hence not cost-effective in terms of a large scale like a large corporation.

On the level of personal improvement, personally, I would like to measure my productivity to know how well or how bad things are going. Measurement drives behaviour. To see a concrete proof of my efficiency, I can prioritise my time and attention to improve and learn.

## Productivity in Teams

“There’s no I in team.” A common phrase to motivate people to work together rather than by themselves to achieve a collective goal. Although this phrase is usually used for sports where teamwork is extremely crucial, it applies to everyone and software engineers too.

Technological companies invest on hiring, managing, and organizing engineering teams yearly. A productive team is capable of innovating and delivering what the company needs on time. As technology constantly evolves at a fast rate, software engineers are crucial to keep up with the change which means that they have the largest cost in the budget. Hence, there is an unavoidable scrutiny on the productivity and resource allocation of software engineering teams. How do we measure a team’s productivity and success?

### *Business Success*

To base a team’s productivity on the business’ success is more than just a black and white situation. This might arguably be the ultimate test, but it is proven difficult to quantify, takes the longest to evaluate and the success might be from more than one team.

If a team delivers a project on time and satisfies the business requirements at a sustainable cost, one will think that this team of software engineers is successful. However, one must not forget that there are many variables not considered with this type of measurement. A team might be productive and working at their utmost best producing great results but the business might not have a great way of marketing the product which would then render the team’s effort futile. In this scenario, the team are in an unfavorable situation where they do not get recognition for their hard work as it is overshadowed by the unfavorable outcome made by other teams. On the other hand, a team might not be productive and produced an output inferior to the team mentioned in the first scenario but receives recognition because of the business’ overall success.

This metric is incapable of showing a single team’s productivity rate and is therefore not a good data for finding a solution or getting an insight on each team’s capabilities.

In an ideal engineering team, software engineers are asked to set goals for themselves on the project they will undertake and fulfil these aims. Tracking their progress using scrums can give us a clear metrics of their output over the time and compare it to their goals initially. As a result, over the course of time, data on their commitments or goals and their actual delivered work start to show trends. This metric informs the business if the software engineering team is productive, functional and if they are generating a sustainable output.

Commitment is the act of following a certain set of goals or obligations. Not only does this metric show quite an accurate result but it also enforces software engineers to work at their capabilities and to not promise what they are not sure they are able to deliver.

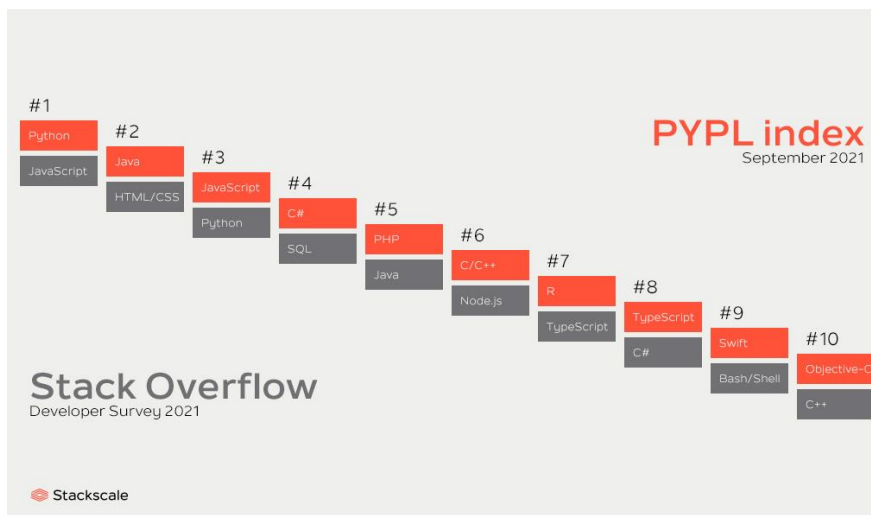
## *Commitment vs. Deliverables*

Despite its promising measurable data compared to other metrics, it is not one without a fault. New software engineers who are inexperienced with this metric might prioritise fulfilling the commitments or scrums and take it as a replacement of their objectives. Last year, I had first encountered scrums and agile development. I found myself constantly sticking close to the weekly tasks and whilst it was not a bad idea to do so, in the long run, the product quality might suffer from the commitment concept.

The software engineering team is under the wrong kind of pressure, wherein the engineers find themselves struggling to deliver tasks on time by means of taking shortcuts, rushing just to fulfil their scrums.

## Productivity in Programming Languages

“The PYPL Popularity of Programming Language Index is created by analyzing how often language tutorials are searched on Google.” Here is the recent PYPL index.



Programs coded in high-level languages are intended to solve large scale problems on parallel machines whilst it may not produce the same result if a program is written on a lower level programming language.

A language’s productivity can be measured by its runtime, or compiler output and might just be easier to measure it. However, there are still underlying factors on in such as how fast it is to wait on memory or how long we have to wait for another enduser to get the message through the network etc.

Ranking programming languages by their expressiveness where the size of source control commits are being analysed with the assumption that one commit corresponds to one functionality.

Another way to measure a programming language’s productivity would be on how big its library availability is. A language with a good array of library support is extremely useful to

make programmers more effective as well as the ease of accessing the libraries' documentation.

## Software Engineer Stereotype

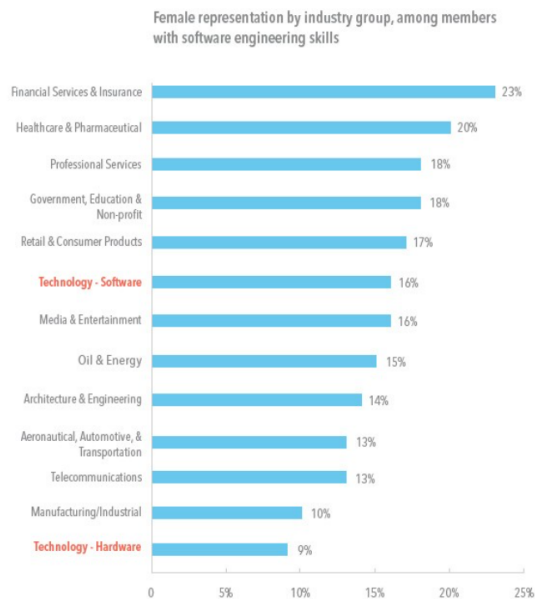
Going slightly off topic, whilst researching for measurable data of Software Engineers, I found a data on Software Engineer stereotypes quite interesting, so I decided to add a section in here for it. Although the statistics on the survey falls on measurable data.

The idea that engineers, especially those in technological industries, avoid human contact as possible is quite silly. An example of exacerbating this stereotype is BBC's IT Crowd starring Chris O'Dowd and Richard Ayoade. The show is absolutely hysterical and a show I just finished quite recently. Despite its comedy and quirk, people in tech are not how they are depicted on the show.

From a survey conducted by job search site Hired on 1600 software engineers, stereotypes on them are just plain wrong. A common misconception is that software engineers do not have a healthy sleep cycle making them unkept and unready for the next day. However, according to Hired, 66% of software engineers prefer to get up early and finish work early than working late. Another stereotype is that tech people are lanky and unhealthy however, high tech companies are making healthy foods and beverages made available to their employees. I went to Google headquarters in Dublin back in 2019 and all the employees I have talked to are very enthusiastic about their canteen that offers a variety of food and accommodates people with allergies and people who are vegan. Hence, debunking this myth.

Whilst some stereotypes are atrocious and inaccurate, we cannot deny the fact that some of them have truths in them too. An example of this would be women in software engineering. Despite the fast-growing dynamic changes in tech industries, from these statistics, women are still scarce in the industry. A lot of films and shows still have little to no women acting as a software engineer or someone technically gifted. However, it is fortunate that a lot of companies are now starting to invest in diversifying their sectors and I hope it will continue, succeed and ultimately break the ceiling.





## Development Analysis Platforms

This section will dive more into platforms or frameworks to capture and gather data as well as the techniques in processing it. These platforms were mentioned on one of the lectures and I took this opportunity to research on it and reflect on them.

These technologies emphasize software engineers' data footprint on a day-to-day basis and quantifies them to present a meaningful and accurate information about the engineers themselves.

### PluralSight Flow

“Work better together with deep insights into your engineering workflow” along with this statement, Pluralsight Flow is an engineering analytics platform that provides easy -to-understand insights on your workflow to make it more efficient. PluralSight also boasts their case study findings which shows increased software development process of teams that can help build healthier development patterns easily and overcome roadblocks.

- 67 percent of business account respondents significantly increased productivity
- 40 percent of business account respondents significantly increased employee retention, and 85 percent of respondents indicated that they were more likely to stay at their companies long-term due to its investment in their training
- 83 percent of respondents significantly increased the quality of their work
- 89 percent of respondents developed skills needed for their job and were given more confidence to learn new technologies
- 62 percent of respondents advanced their careers faster and a quarter of respondents obtained a promotion

---

<https://www.linkedin.com/business/talent/blog/talent-strategy/women-in-engineering-the-sobering-stats>

<https://spectrum.ieee.org/time-to-update-the-software-engineer-stereotype>

Flow follows code metrics such to *Efficiency* track efficiency of code being added to the project. The code metric Efficiency measures the amount of code that are productive. A productive code is a code that has not been modified by other engineers after submitting it. This metric is interesting to discover as it does make sense that a code is efficient and useful if it is not rewritten by others. Although this metric is great, efficiency drops as time goes by. Since the important pieces of code have been established earlier on the project, finishing touches in the end might not be deemed efficient.

This platform also has a Review Metric which overviews the efficiency of your process and can help guide you to areas one can improve on. Review Coverage is one of the metrics that fall into this category. It measures the amount of code that is reviewed in each pull request. It provides a picture or better understanding on how much of the code committed to the project has been reviewed before being pushed to its master branch. Using this information, an engineer can discover how good their code is and it creates an awareness to how other people view their code.

Overall, I think that PluralSight Flow is a great platform in gathering data that represents software engineers' tasks and activities. It provides a picture on the team's dynamics and this constantly updating information help the team find their strengths and weaknesses to ultimately finish their project.

## Code climate

Code Climate is another tool that provides objective analysis on the code's quality. It also provides information and tools needed to make it more efficient. They offer two products:

*Velocity*- identifies flaws and patterns within a code. It also provides a visualization of code quality and helps in assisting the team to find a solution to these flaws. Velocity mainly focuses on improving the code's functional quality.

*Quality*-An automated tool that processes all pull requests automatically. It aids on improving the code's quality in terms of unit coverage, unused ports, and formatting. Quality's task is to ensure quality before the merge.

Code Climate's target is on Engineering teams and supports more than ten programming languages whilst PluralSight on the other hand, has a wider audience to accommodate. Code Climate is able to analyze any branch from any repository and notifies quality changes, it can also objectively enforce the idea that commits are only pushed if it will improve the code's quality through detecting code duplication and high churn.

---

<https://www.businesswire.com/news/home/20160302005126/en/Study-Shows-Pluralsight-Increases-Employee-Productivity-and-Retention>

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

<https://iqunlock.com/pluralsight-flow-review-is-it-worth-it/#pluralsight-flow-and8211-team-collaboration-metrics>

## Timeular

Timeular is a tool that although not marketed towards software engineers, they are used by them, along with people who want to be in control of their time, track their productivity and to lead a more rewarding work-life. I have first heard of it from the lecture and was intrigued about it.

Unlike any tools on this section, this is a physical device that one can hold to help stay focus and productive. I personally think that it is an ingenious idea to use a real-life object and gamify productivity in such a way that it is fun, intuitive and does what it is supposed to do.

This time tracker might not help increase efficiency of code and it also does not help the team find out tools that might be of help to them. However, it is extremely helpful managing schedules, and time spent on working. Seeing the hours spent on projects will let you know if you are working efficiently. It also gives you an idea on how much longer you take a break and your time interval working.

Whilst it seems like the perfect time tracking tool, I found that since the Timeular's 8-sided dice only tracks time based on which side is facing up, it is not possible to track two tasks at once. Although that might be a good thing as you only must focus on one thing at a time, it might not be the case for some software engineers who are multitasking.

## Algorithm Approaches

Software complexity is one of the most important concerns in software lifetime development. As the project gets bigger and the code becomes more complex, a software engineering team will most likely face problems to debug and maintain it. The lower the complexity of the code, the easier is it to measure its various other factors. Bugs and defects are easier to find and fix, and the cost of many factors will be reduced.

### Cyclomatic Complexity

Cyclomatic complexity refers to the number of possible execution paths inside a given piece of code—for instance, a function. The more decision structures you use, the more possible branches there are for your code.

Cyclomatic complexity has an important role in testing. By calculating the cyclomatic complexity of a function, the minimum number of test cases you'll need to achieve full branch coverage of that function can be quantified. Therefore, cyclomatic complexity can be used as a predictor of the level of testability of a piece of code.

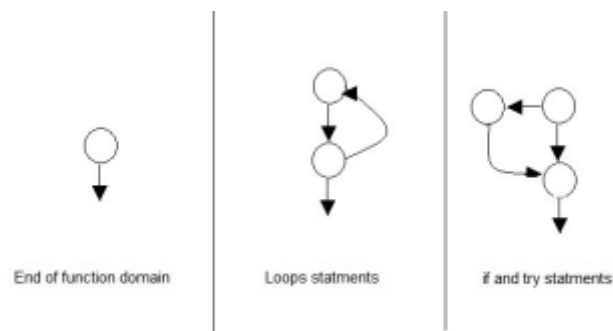


Figure 1. Cyclomatic direct graph

```
Cyclomatic complexity = E - N + 2*P
where,
  E = number of edges in the flow graph.
  N = number of nodes in the flow graph.
  P = number of nodes that have exit points
```

Problems might arise with a lower cyclomatic complexity in terms of cognitive complexity. The first thing that came into my mind on how to implement cyclometric complexity is through recursive code. I often have a hard time understanding how it works, how it creates a stack until the final bit and so on. Cognitive complexity is how difficult it is to understand a piece of code. From my example, cyclomatic complexity can be one of the factors in increasing the code's cognitive complexity.

To reduce cyclometric complexity, it is better to have smaller functions that are simple to understand yet concise. There will be lesser potential bugs on smaller functions and it is easy to reuse them. Switch cases and If-Else functions need branches and testing for each case. Hence, another solution is to reduce the number of decision structures to keep the complexity low.

## Halstead Volume/Effort

Similar to Cyclometric Complexity, Halstead volume is another approach to create a better understanding of software complexity. Developed by Maurice Halstead, Halstead Volume/Effort determine the mental effort required to develop or maintain a program. The lower a program's Halstead effort, the simpler the program is to configure.

---

<https://linearb.io/blog/reduce-cyclomatic-complexity/>

[https://www.tutorialspoint.com/software\\_testing\\_dictionary/cyclomatic\\_complexity.html](https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.html)

$$V = N \log_2 (n_1 + n_2) \quad (2)$$

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \quad (3)$$

where:

$n_1$  Number of distinct operators in application

$n_2$  Number of distinct operators in application

$N_1$  Total of operators in application

$N_2$  Total of operands in application

(2) Complexity volume of program V

(4) Test Effort

(5) Program Level

$$effort = \frac{V}{PL} \quad (4)$$







$$PL = \frac{1}{(n_1/2) * (N_2/n_2)} \quad (5)$$

This approach is not optimally accurate, as there are many factors which can affect the program's complexity, for example, functions, loops, IF statement, and so on.

## Shao and Wang's Cognitive Functional Size

*"D. I. De Silva et al. [8] made a comparison to test relationship between three cognitive complexity metrics: Kushwaha and Misra (KM's) cognitive information complexity measure (CICM), Shao and Wang's (SW's) cognitive functional size (CFS), and Misra's cognitive weight complexity measure (CWCM). As in [?] they used the same ten java programs and asked thirty expert developers from five huge companies to rank the programs from least to highest complexity. Finally, the Spearman's rank correlation coefficient is conducted to compare between the ranks resulted from the manual calculation of the three metrics and thirty experts judgments. As a result, they confirmed that Shao and Wang's (SW) cognitive functional size is the best metric to be considered in real world."*

This approach also uses a flow hart to represent each program step. What differs to it from Cyclometric complexity(CC) is that it has two different techniques to calculate the code's complexity. Firstly, it does not have predictive nodes unlike CC. Secondly, each control on the code has different weights.

Category	Control name	Flow diagram	Weight
Sequence	Sequence step		1
Branch	If – else control		2
	Switch control		Number of cases
Iteration	Loops controls		3
Embedded	Call functions		2
	Recursion function		3

The table shows how this metric give different weights for the controls. Because of these weight difference, this metric is more accurate in calculating and capturing the complexity of a program than CC. This metric takes recursive functions and loops into consideration.

## Ethical Concerns

In this module, we are slowly being introduced to measuring software engineering. We are being familiarized to this new environment where we are encouraged to test, code, commit and repeat. Are there any ethical implications in this? Are we being conditioned to being used to be measured as prospective software engineers? Is it a bad thing? How much are information are leaving behind? These are just some ethical concerns I have thought of while making the report.

How far should we go when measuring a software engineer's productivity? It is important that both the employer and the employee are aware of these boundaries. While these data might be crucial in improving a company's capacity and potential, humans are complex beings driven by emotions and goals. If people are made aware that they are being measured, there are many ways a person might react. Some might start to take less breaks, write longer codes, go to work early, start getting close to their managers, etc. It is only natural for us to ask how we are being measured. As software engineers, or at aspiring to be one, who are more than aware of data footprint, they might be more sensitive, and rightfully so, to how

their data is being collected. A company cannot not inform their employees about it. Hence, as humans, we will unconsciously condition ourselves to make ourselves have a good record with how the company is measuring our productivity. One might say that it is better for the company that way, and I agree. However, it defeats the purpose of what we want, raw and untainted data.

Measuring a person's process might incur competitiveness in the workplace. Whilst competition increases productivity, it might also come with repercussions where instead of working together, they see each other as competitors. This is scenario is counterproductive to the ultimate goal of measuring software engineering. Perhaps as a solution, employees should be able to access data collected from them where they can consistently track their record to find out in which areas, they might need help.

## Conclusion

My research on this report has taught me is that while measuring software engineering is complicated, it is a practice that companies should undertake to boost their performance. All methods of measuring have underlying ethical concerns, but I think it is most important for the companies to be completely transparent with how their data is taken and measured to prevent issues between two parties.

Software engineers should always look for ways to improve their code. Whether it is for the company they are working for, someone told them to do so or just for personal growth, it is good to measure one's productivity to reflect on it and find areas that they can improve on. Fortunately, through the use of tools and metrics, one evaluates the health of a codebase and project. What is interesting in measuring software engineering is that there is no one way. It can be customised on what an engineer wants to measure in particular be it their productivity, the complexity of code they contribute to the team etc.

Measuring software engineering enables up to improve and optimize what currently exists and encourage innovation in the field of technology.