



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

Measuring Software Engineering Report

Aislinn Addison-Smyth – 19337226

3rd January 2021

Contents

Introduction	1
1. Measurability of Software Engineering	2
2. Development Analysis Platform	7
3. Algorithmic Approaches	9
4. Ethical Analysis	11
Conclusion	11
References	13

Introduction

The purpose of this report is to investigate the measurability of Software Engineering. This report will consist of four sections, the first section outlines how one can measure engineering activity. It will pose questions such as, is software engineering able to be measured? If so, how can we measure it and what can we measure? Are the mechanisms we use a fair and accurate way of measuring software engineering?

The second section of the report is about the emergence of Development Analysis Platforms. These platforms assist us in the processing and understanding of large data sets within software engineering. This section will discuss the frameworks and infrastructures created to help us with data processing and analyzation.

The third section of the report is regarding the various Algorithmic Approaches we take in to successfully measure software engineering. I discuss in detail the necessity to have abductive reasoning integrated when measuring a process, outputs generated by an algorithm are not sufficient and can often be misleading.

Our final section will be a discussion of the Ethical Analysis, this section considers all three previous parts of the report. It details my opinion of whether the measuring of the productivity of a software engineer is ethical or not.

1. Measurability of Software Engineering

When I initially read the assignment specification, I was confused by the simplicity of the question at hand, is software engineering measurable? Of course! I assumed that as engineering follows a process and processes can be measured, that there is no doubt that software engineering can be measured.

I began my research by watching YouTube videos and reading countless articles to figure out what metric is the best method for measuring the productivity of a software engineer. To my surprise my efforts were unsuccessful, I was unable to come to any concrete answer regarding the best metric to use in my report. Upon further researching and investigating, I realised that the question I was originally posed with was a lot more complex than I had expected, it was not a one answer question. I concluded that this report is not a matter of whether you can measure software engineering, it is a matter of whether you can accurately and fairly measure the productivity of a software engineer. I came to an answer of yes and no. This report will take you through my thought process and reasoning for this answer.

I will start off by saying yes, there is no doubt in my mind that software engineering is measurable, it follows a distinct process, therefore it can be measured. I will begin my argument with this famous quote, *Albert Einstein*, “*Not everything that can be counted... counts*”. There are many ways which we can measure the productivity of a software engineer, however there are very few ways in which we should measure. I will list in my opinion the worst ways software engineering can be measured.

The most famous metric that was initially used to measure software engineering was to count the number of lines of code written by a software engineer. This meant that the more lines of code that were written by a software engineer meant the more productive they were. In my opinion this is the worst possible way a software engineer can be measured; as software engineers we have been taught to keep our codes short, precise, and efficient, not long, extensive, and confusing... for no real reason. There are numerous issues created with this metric, for starters, logistically it does not make sense, how can we compare software engineers that write in different languages? See figure 1.1 below:

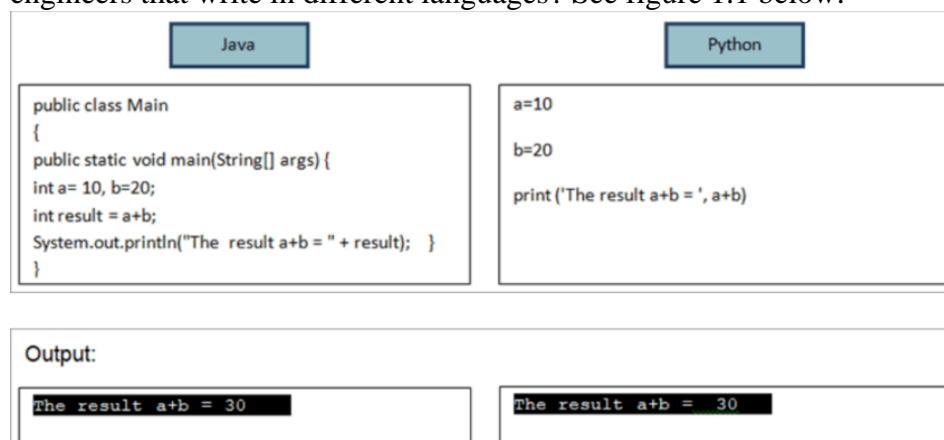


Figure 1.1: Java vs Python, snippet of code extracted from <https://www.softwaretestinghelp.com/java-vs-python/>

Let us say software engineer X wrote the first snippet of code in java and software engineer Y wrote the second snippet of code in python. The code written by software engineer X has a total of seven lines of code (including lines for brackets), whereas the code written by software engineer Y has a total of three lines. Both codes follow the same procedure and produce the same output, however, using this metric software engineer X would be deemed to be more productive than software engineer Y.

Secondly, if the metric used encourages the writing of more lines of code, software engineers will produce a standard of work to fit this metric, i.e., unnecessary extensive pieces of code, codes that are a lot more bug prone, and codes will a lack of efficiency. This metric is not only useless in determining the productivity of a software engineer but can also be harmful to individual or groups of software engineers. This metric may be interesting to see how many lines of code it takes to produce a code, however, that is all it is good for. An important quote which relates to measuring productivity based off lines of code is the infamous Bill Gates, quote, *'Measuring programming progress by off lines of code is like measuring aircraft building progress by weight'*.

The second metric I would like to discuss is the counting of the number of commits to a repository. This metric works off the fact if there are no commits to a repository it means there has been no work done and the more commits to a repository the more work that is getting done. Originally, this was the metric I believed to be the best way to measure software engineering, however, the more videos I watched the more I saw the flaws in this metric. The idea of the commit's metric is good, it incentivises small frequent commits as opposed to large infrequent commits, the idea was to reward teams with a high number of commits and improve teams with a low number of commits. An issue that arises with this metric is that the number of commits to a given repository does not tell us anything about the value and quality of the given commit, the only information about the commit is when it was done and to what repository. It becomes a lot easier for software engineers to again, change their work style to fit this metric, commits begin to become small and pointless as it did not matter what was in the commit, only that there was a commit. I have attached in figure 1.2, a diagram of the number of commits I have made to my GitHub as a whole. There is a key in the bottom right hand side corner which guides us on the colour scheme.

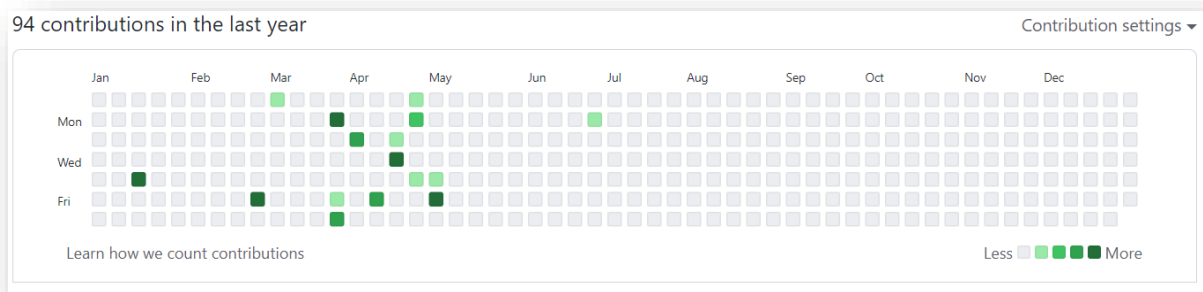


Figure 1.2: A diagram of commits taken from my GitHub:

<https://github.com/aislinnsmyth?tab=overview&from=2021-08-01&to=2021-08-31>

For example, using figure 1.2, if I hover my pointer over the dark green square in line with May and Friday, it informs me that I made 26 contributions on May 7th, and below it informs me that I made these 26 commits to one singular repository, 'Algorithms-Project-21'. I have attached a screen clipping showing this information in figure 1.3.

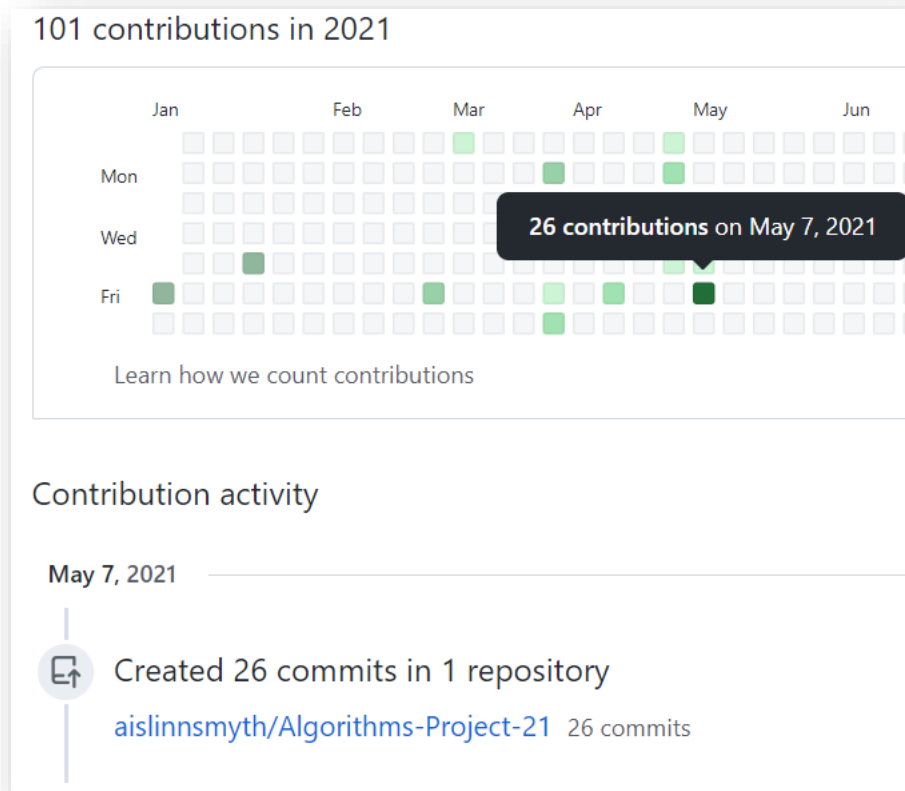


Figure 1.3: A diagram of a specific day of commits taken from my GitHub:

<https://github.com/aislinnsmyth?tab=overview&from=2021-08-01&to=2021-08-31>

A commit is simply a “save point” in a project, we know nothing about what has been saved in this save point, only that an alteration of unknown degree has been saved and committed to the project. By using this metric to reward software engineers, it incentivises them to commit to the code every time they alter a line of code.

The final metric I would like to discuss in this report is the number of hours spent on a task. In my opinion, I believe there is such an issue in comparing software engineers and measuring their productivity based off a singular figure. Every software engineer works differently, some like to do the bar minimum of what they are required to do, and some like to go beyond what is asked of them. I do not find it fitting that a ‘lazy’ software engineer whom although is doing the bar minimum requirements can be seen as equally as productive as a driven software engineer, that not only does the task at hand, but excels to make additions/alterations to the project to optimise performance. This metric is only useful when there is an extreme outlier, if there is someone taking notably longer than all other software engineers. This metric does not assist for those which take the same amount of time to deliver

a high-class standard of code to those delivering a low-class standard of code which although works, has not been created to the highest standard.

I encountered numerous interesting sources regarding the worst possible metrics to use when measuring software engineering. I have attached the YouTube link below, referenced under number 2. I have also attached a relevant blog post which explains the YouTube video in a more condensed version. Both sources outline a theory called 'The Flawed 5 Metrics', here they list the five worst metrics to date when measuring a software engineers' productivity. Watching this video and reading this blog post helped my better understand why these metrics were detrimental and damaging to teams and individual software engineers. The metrics that were outlined in these informants were Lines of Code, Commit Frequency, Pull Request Count, Velocity Points, and "Impact". The three metrics I discussed above, are in my opinion, the worst metrics to use when measuring a software engineer. All these metrics incentivises negative behaviour and results in software engineers changing their work style to reach quotas as opposed to working for efficient product.

In my opinion, there is not one answer for the best metric to use when measuring software engineering. Metrics were originally created to be able to sufficiently measure a process. I do not believe that every software engineering team or every software engineer can be measured using one metric, a metric which may work for a team of software engineers in Amazon may not fit the work style of a team of software engineers in Microsoft. If we outline what a metric should not measure and what we want from the metric, we may be able to create a metric which best suits the individuals being measured. All the badly regarded metrics I listed above all have one thing in common, they measure the output of software development. By measuring output, we are not measuring the process of software engineering, and the process is the aspect which the most time is spent on. When we decide to measure an end product using metrics which promote irrelevant aspects of code or unnecessary additional steps, the end produce will be disappointing. As we normally aim for efficient algorithms, small changes in code can usually take a long time which these metrics do not factor in. I know for me I would rather take longer to be proud of the efficiency of my code than get my work done sooner to a standard I know may not be up to par. These points bring me back to my original statement that although technically there are many things in software engineering that can be measured, there are very few which should be measured.

In my opinion, although I cannot give a set metric which will measure software engineering accurately and fairly, I can highlight what a good metric should measure. As I touched on above, I do not believe one metric fits all, a manager must figure out what he wants for his team and fit a metric to that. Based on the research I conducted, for a metric to succeed there are certain aspects it needs to address, for starters metrics need to measure a process as opposed to an output. A way this can be done is by using a Response Time metric, how this works is that within a team of software engineers', members will open issues within codes, the response time is the average time it takes for a specific team member to address the issue. This metric measures the process of code building through open and closed issues and can measure a software engineers productivity based off their average response time. This metric too has its own flaws, for example, if a code issue is opened on a Friday and does not get addressed until Monday it can completely skew the average response time. Another



characteristic to consider when creating metrics is to create a metric which measures against outlined targets as opposed to measuring software engineers against absolute numbers. We cannot use a single number to compare teams or individual software engineers, the productivity of a software engineer should be factored by multiple different metrics. Much like many things in life, one figure does not fit all. Every software engineer brings a different aspect to a team; therefore, a software engineer will never be able to be successfully compared to fellow software engineers based off an absolute number. Finally, software engineers tend to work in teams, although it may be tempting to grade individual software engineers, I do not believe this to be beneficial. As I highlighted within this paragraph, everyone contributes differently to a team so measuring individuals singly based off one metric will always prove to be unsuccessful. The conclusion I have come to is that it would be more beneficial to measure the productivity of teams of software engineers as opposed to individuals.

To conclude, I do believe that with the right tools and understanding software engineering can be measured, however, metrics must be created and shaped accurately to a particular team.

2. Development Analysis Platform

The first section of my report was concerned with whether we can measure software engineering, where I concluded that yes, we could measure software engineering **only** if the metrics used address beneficial aspects of the work done. Within software engineering we can have huge volumes of data that we would like to dissect, process, and create different analytics from. These datasets can be so large that we need specialist infrastructures to assist us in analysing the data efficiently and cheaply. In this section of the report, I will discuss the various frameworks and infrastructures that can be used.

The main platforms, in my opinion, which can gather and perform calculations on our data sets are the newly developed cloud services. Platforms such as Amazon Web Services (AWS), Google Cloud Platform, IBM Cloud, Microsoft Azure are what we call Cloud services, these are all a part of utility computing. Cloud services are infrastructures, platforms or software that are hosted by third-party providers, in these cases Google, Amazon, IBM and Microsoft. These services are made available to users through the internet. As AWS is statistically the most popular cloud service provider in the world, I will be using and discussing this platform in detail. AWS is a cloud computing platform which offers many different cloud-based products, a few examples of these being: compute, storage, databases, analytics, networking, developer tools and many more applications. Amazon Web Services has object storage with, as described on their website 'industry-leading scalability, availability, and security' to store and retrieve any amount of data from anywhere. There is no limit to the storage size for your data, the volume of data and number of objects you can store in the cloud is unlimited, the only aspect of storage which has a limit is that in one single PUT, as we know commit, the largest object which can be uploaded is 5 gigabytes. It is safe to say that there is no question regarding whether AWS can handle large datasets. Amazon has a specific website where it discusses the different storage options they offer to users. The first batch of storage they offer is called Object, file, and block storage, in this section there are 4 different storage types listed: Amazon Simple Storage, Amazon Elastic File System (EFS), Amazon FSx, and Amazon Elastic Block Store (EBS). Under each section there is a short description regarding the benefits of choosing that specific storage option. Figure 1.4 will be screen clippings of these storage options with their descriptions for context.

 Amazon Simple Storage Service (S3)	 Amazon Elastic File System (EFS)	FSx Amazon FSx
Object storage with industry-leading scalability, availability, and security for you to store and retrieve any amount of data from anywhere.	A simple, serverless, elastic, set-and-forget file system for you to share file data without managing storage.	Fully managed, cost-effective file storage offering the capabilities and performance of popular commercial and open-source file systems.

Figures 1.4: Screen clippings of various storage options and their associated perks.

In addition to the various storage options, AWS also offers options such as Data Migration, Hybrid cloud storage and edge computing, managed file transfer, and disaster recovery and backup. An option I found particularly fascinating was in hybrid cloud storing and edge computing, AWS has an

option called AWS Snow Family. Aws Snow Family offers “edge compute, data collection, and data transfer services with security and end-to end logistics for mobile and rugged deployments”.

Now that we are aware of the different storage options we are given with this cloud-based service, we can dive into the analytics side of things. AWS offers numerous different analytics services, they split these services into four different categories. 1) Analytics, 2) Data Movement, 3) Data Lake, and 4) Predictive Analytics and Machine Learning. See figure 1.5 for a detailed description of these analytics services associated with the analytics category.

Category	Use cases	AWS service
Analytics	Interactive analytics	Amazon Athena
	Big data processing	Amazon EMR
	Data warehousing	Amazon Redshift
	Real-time analytics	Amazon Kinesis Data Analytics
	Operational analytics	Amazon OpenSearch Service (successor to Amazon Elasticsearch Service)
	Dashboards and visualizations	Amazon QuickSight
	Visual data preparation	Amazon Glue DataBrew

A short use case description follows each AWS service, most are self-explanatory for what they are used for. I will discuss the last two AWS services, as they are the visualisation analytics services. Amazon QuickSight uses interactive dashboards and automatically looks for patterns and outliers to better understand the data. It has a built-in feature where you can ask the dashboard to show particular analytics from the data. For example, if you were given a dataset regarding different colour cars bought from a

Figure 1.5: A screen clipping of the Analytics category on the AWS website.

dealership within the last 10 years, if you were to search for all the different colour cars bought from a dealership in 2019, the dashboard can create visual analytics to fit your demand. Finally, Amazon Glue DataBrew is AWS’ newest data visualisation tool which capabilities range from profiling your dataset, cleaning, and normalising the data, mapping data lineage, and automating data cleaning. The most amazing thing about all these new Cloud-based platforms is that for one, storing and processing data has become so much cheaper and so much more practical. Now being able to upload and store data to a cloud online, now enables you to be able to have access to this data from anywhere in the world, and from any internet source. AWS offers a pay as you go scheme for payment when using its services, therefore, you are only required to pay for what you are using.

Apart from AWS there are other software platforms available to us which can store and process large amounts of data. These platforms include Pluralsight Flow and Waydev. Pluralsight Flow is a cloud-based productivity analytics solution which gives insights into project, progress, productivity, and workflow using objective metrics, KPIs and graphs. Waydev is a software which can integrate with tools you use for your day-to-day activities such as GitHub and can assist in gaining visibility into engineering work, the software measures organization-level efficiencies and starts optimizing your development process. These two applications differ from the use of AWS and Azure Microsoft as their target audience are business companies and want to be used for business purposes. AWS and all other cloud-based platforms are aimed at a general user and has a service for everyone. Platform Flow and Waydev both specialise in gathering large volumes of data from big companies, analysing, and generating data regarding a business’ productivity, its progress and it will give insights into its projects using different visualisation and computational tools.

3. Algorithmic Approaches

The third section of this report is heavily tied in with our first section which was, is Software Engineering able to be measured? In the first component of the report, I described the worst ways I think software engineering can be measured and gave my own opinion of the aspects of software engineering to consider when creating measurable metrics. This section will delve into further detail regarding the aspects of software engineering is measurable and the algorithmic approaches we take to successfully measure it.

Let us first begin by going over what the measurable data of a software engineering is, first thing is all aspects of a software engineers programming. This ranges from the code generated, the cyclomatic complexity of the code, the readability of the code to the number of bugs located within the code. The cyclomatic complexity (CYC) of a code is, as it says a software metric to measure the complexity of a program. It is the quantitative measure of the number of linearly independent paths within a code, in more straight forward terms, the count of the number of decisions in the source code, the higher the count the more complex the code is. The cyclomatic complexity is measured by developing a Control Flow Graph of the code, and this Graph measures the various decisions laid out within the source code. The cyclomatic complexity can be calculated by the formula: $E - N + 2 * P$, where E is edges, N is nodes and P is connected components. The issue with this metric is that much like hard coding being a substandard idea, so can enforcing a hard rule. It does not consider the readability of a code. Sometimes the complexity of a code must be sacrificed for a code to be readable. The metric does not consider other aspects of the program and thus, sums up a single value for the complexity of the code without any description. As I previously stated in my initial part, I do not agree with using absolute numbers to describe a code as they can be extremely misleading. There is a range of contradictory and confusing ways that we can measure software engineering, but there is no real way to capture the complexity of something to a level where you can understand all aspects of the code.

The prospect of having a hard rule leads me into the topic of measuring software engineering through the tools of machine learning. There are two main algorithmic approaches in machine learning, Supervised learning, and Unsupervised learning. Supervised learning methods means that you are assigning labels to observations of provides structure within the data, examples of these would be either classification or discriminant analysis. Unsupervised learning methods are methods which only use the internal structure of the data to explain the data in a reduced number of dimensions, examples of these are Principal Component Analysis and cluster analysis. These are black and white algorithmic approaches to analysing a data set, meaning that although you can analyse data using these methods, these are examples of what I would call hard rules. Abductive reasoning is a key part to any and every process which is something that machine learning techniques lack. Machine learning is only good at using statistics and mathematics to get an output, as opposed to using human judgement to gauge whether something is productive or not. As I have reiterated several times throughout my report, we should be measuring the process of software engineering, not the output.

It is vital that we have human judgement and opinions when interpreting the productivity of a team or the measurability of a discipline. The modern artificial technologies used to gather data are

missing the point of why we measure and what we are measuring for. These technologies do not consider external variables. They have a set process they follow and give certain output from that. It is all well and good applying an algorithmic method to dataset, but all that will give us is an outputted value or graph, we as software engineers must be able to interpret these results. It is important that the fate of measuring software engineering is not left done to a single machine, a single algorithm, or a single metric as this will result in incorrect and misleading information. There is always a need for an individual to be involved in the measuring process as we need abductive reasoning to inform us on whether the performance data sets, and processing done is useful or whether the outputted results are void. We can do so by setting parameters, creating models to these parameters, and assessing the results of these metrics. It is clear that software engineering, and software engineers are not easily measured at all.

On the topic of single use metrics, algorithms, and machines, additionally I believe it is important to point out that just as single use metrics are detrimental, so can metrics which try and gather too much data from a data set. New metric was set up called “Impact”, on the surface this metric sounded like a great idea, it took as many components and aspects of the code into account as it possibly could. The problem began to arise when it would give software engineers “Impact scores”, these figures gave the impression that the higher the figure the more impactful you were to the business and to the team. Impact is a way to measure the significance of a code change, this metric takes the following into account: 1) The amount of code changed. 2) What percentage of the work edits to old code. 3) The number of edit locations. 4) The number of files affected. 5) The severity of the changes when old code is modified. 6) How this change compares to others from the project history. This metric has tried to measure every aspect of a software engineers code, the higher the impact score of a code commit, the more ‘expensive’ the contribution was. We went from two extremes, metrics only measuring a small aspect of software engineering, to a metric that tries to measure too many things at any given time. In my opinion, although the concept of this metric is correct, I do not think it is executed as well as it could have been. I believe it is trying to measure too many components at once. I think it is unfortunate that they have used the term “Impact” and created “Impact scores”, as if you do not score high on this metric, it gives the impression you have no impact on the team. The metric does not consider bug fixers, and the fact that a small change goes a long way, it could take days to alter a small section of code, but if the impact score is not high for your change you are deemed to be unimportant.

4. Ethical Analysis

Is measuring the productivity of a software engineer ethical? This question is always very subjective and can generate a lot of mixed emotions and mixed opinions. In my opinion, I believe that measuring productivity in software engineering is ethical. How are we meant to improve and excel if we do not measure how productive we truly are? We see people measuring their own productivity everywhere. I will demonstrate with a personal example, when I was younger, I was a competitive 100m sprinter. I used to train four to five times a week, I used to time myself every session and compare my times against previous times. The reason for this is to determine whether I am excelling or whether I am not training effectively. If I was not satisfied with my times or with the progression of my personal bests, it would mean that my training is not as effective as I would like. Much like what I have discussed in software engineering, what I learnt was that instead of measuring my output, which was my end time, I needed to measure my process to dissect what aspect of my training needed to improve. Instead of measuring against finalised time figures, I would measure my reaction times, at what section of the race was I at my weakest point, etc. When it comes to running there are many different aspects which can be measured that will affect your end product, but not everything that can be measured in running will be beneficial to measure. This example is a simplistic scenario of self-measuring and self-improvement. How is this much different from measuring productivity as a software engineer? A key component to this analysis, is to make sure we do not take the aspect of humanity and abductive reasoning out of it. Although algorithms and analysis platforms are extra-ordinary, they lack one vital part: human judgement. For something to be ethical it must follow most social normalities which are actions that are acceptable to the public, so how could we say that measuring software engineering is ethical if we did not have a human component tied into it? I think it would be a horrific mistake to leave the measurability of software engineering down to machines and algorithms, this topic is only ethical when there is human judgement at the heart of it. We do not wish, as software engineers that measuring becomes purely calculated, there are always going to be extraneous variables which a machine will not account for. Therefore, to conclude I truly believe that measuring the productivity of a software engineer is not only ethical but is the way forward in the technological world.

Conclusion

An important thing I have learnt whilst writing this report is that I have always believed that Software Engineering was based off fact as it is a discipline within the mathematic world. I thought these disciplines were non-subjective, based purely off information that has been researched, proven, and deemed to be non-negotiable. I have come to realise, now more than ever, that software engineering and measuring software engineering is entirely subjective. There is no one way to code, no one way to measure code, no one way to interpret code, software engineering is a completely subjective discipline that can be approached hundreds of different ways. Additionally, to note, none of the ways I have listed in my reports are the 'wrong' way of measuring software engineering or the incorrect algorithms to use on a data set within software engineering, they are my opinion of what I believe should be used to measure software engineering and what I believe is a waste of time to measure.

To sum up, I do believe with the right metrics and resources, software engineering can be measured, each metric must be shaped to a team's desire, not one metric fits all.

To conclude I will re-iterate the Albert Einstein quote, "*Not everything that can be counted... counts*".

References

1. <https://www.twi-global.com/technical-knowledge/faqs/engineering-design-process>
2. <https://www.youtube.com/watch?v=cRJZldsHS3c>
3. https://www.gitclear.com/four_worst_software_metrics_agitating_developers
4. <https://www.usehaystack.io/blog/software-development-metrics-top-5-commonly-misused-metrics>
5. [https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services#:~:text=Cloud%20services%20are%20infrastructure%2C%20platforms,to%20users%20through%20the%20internet.&text=Users%20can%20access%20cloud%20services,virtual%20private%20network%20\(VPN\).](https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services#:~:text=Cloud%20services%20are%20infrastructure%2C%20platforms,to%20users%20through%20the%20internet.&text=Users%20can%20access%20cloud%20services,virtual%20private%20network%20(VPN).)
6. <https://searchaws.techtarget.com/definition/Amazon-Web-Services>
7. https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=*all&awsf.Free%20Tier=*all&awsf.tech-category=*all
8. <https://aws.amazon.com/big-data/datalakes-and-analytics/>
9. <https://aws.amazon.com/products/storage/>
10. <https://waydev.co/>
11. <https://www.perforce.com/blog/qac/what-cyclomatic-complexity>
12. <https://www.geeksforgeeks.org/cyclomatic-complexity/>
13. <https://medium.com/swlh/pros-and-cons-of-cyclomatic-complexity-as-a-metric-b25000dcda9c>
14. <https://www.pluralsight.com/blog/teams/impact-a-better-way-to-measure-codebase-change>