Measuring Software Engineering.

# Abstract

The aim of this report is to look into the different ways that software engineering can be measured and assessed. I will do this be looking into the four topics outlined in the brief. They are: measurable data, computational platforms available, algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

I will compile this report using a mixture of academic papers, online resources and information discussed in lectures on the topic of software engineering.

# Introduction

So what exactly is software engineering. It is defined as “*a process of analysing user requirements and then designing, building, and testing software application which will satisfy those requirements*” (Guru99). The term was first coined back in 1968 at a NATO conference. It was at this point in time that it was decided that a systematic approach should be applied to the development of software. This refers to the designing, building and testing of software systems.

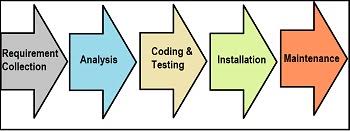
The years leading on from 1968 saw rapid growth of the industry. It became important to measure and assess the process of software engineering in order to answer questions about performance and how different processes work. I will look at how this can be done in detail under these four headings:

1. Measurable Data
2. Computational Platforms Available
3. Algorithmic Approaches Available
4. Ethics of Analysis

Before I analyse software engineering, I am going to take a deeper look into the current software engineering process. It is important to note that this process does change from project to project, and there is no “fixed“ or “ridged” rules or techniques that all software engineers should follow. As such, I was look at general methods for measuring the software engineering process.

# Software Engineering Process

The process involve a set of steps that must be carried out in order to develop software. This occurs before we can look into measurement and assessment. The below outline these necessary steps.



This process is known as the Software Development Life Cycle (SDLC). Each type of process must include the above activities in order to properly develop a software system (Gadagkar, 2020). They tend to follow the below steps:

1. **Customer Requirements:** This step is about communicating with the customer, and finding out what they are looking for prior to building a software. A Customer Requirement Specification (CRS) is filled out to best describe what the software is going to do.
2. **Requirement Analysis:** This step focuses on creating functional specifications. It deals with any financial, technical and operational issues that may arise and questions the feasibility of the project.
3. **Creating a design:** A high level design of the software along with a low level explanation of each component in created.
4. **Coding, Testing and Installation:** The design in implemented by writing code. This undergoes tests to ensure each requirement is met, and the software is then uploaded to a server.
5. **Maintenance:** The software can be modified or updated to add features to fit the customer’s needs.

There are many types of software engineering processes: waterfall, agile and iterative being the most popular. The *agile* model allows the customer to view the software after each stage of development, whereas in the *waterfall* model, the development process is pre-defined and the customer cannot see the software until the final step is completed. *Iterative* models add new components to previously developed ones, so in this case there are continued new versions to the software. All these models have their own advantages and disadvantages, but each model must follow the SDLC steps outlined above regardless (Gadagkar, 2020). So how is the software engineering process measured and assessed?

# Measurable Data

Nowadays, vast amounts of data can be collected throughout each stage of the software engineering process. However this was not always the case. In the early days of software engineering, data was collected by manually inputting figures and information into spreadsheets. This meant that not much data was actually collected as this task was very tedious and took up a lot of time. Software engineers thought their time would be better spend developing the software itself. Any data that was collected was also prone to human error.

***Why do we collect and measure data?***

Over time, the importance of recording data became apparent. It can help highlight to companies what areas they can improve on in order to maximise time, costs and efficiency. There have been rapid growths in recent years in data from technology systems in terms of both communication and information. This increase is directly from the growing use of technology in the world today.

Data can provide very useful information to software engineers. Ultimately, the importance of collecting data is to ensure that scarce resources are being made use of most efficiently. So now that we know the importance of collecting data, how do we use it to measure software engineering?

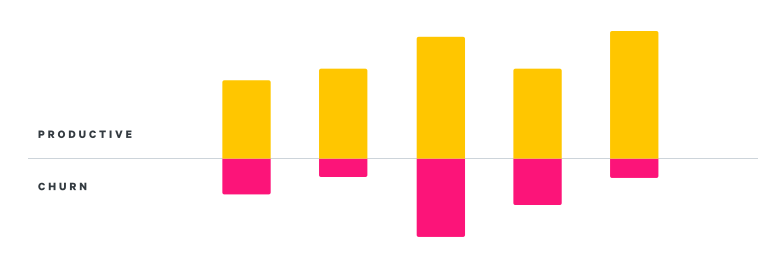
***How do we measure software engineering using data?***

##### Metrics are used to monitor performance. A software metric is “*a measure of software characteristics which are quantifiable or countable”.* All process have the same aim: to increasing productivity while reducing waste in doing so (Altvater, 2017). It is important that software metrics are understandable, computational and easy to use. Software engineering processes can use different software metrics to record the data. I am going to look at how productivity, process, and quality software metrics can measure software engineering.

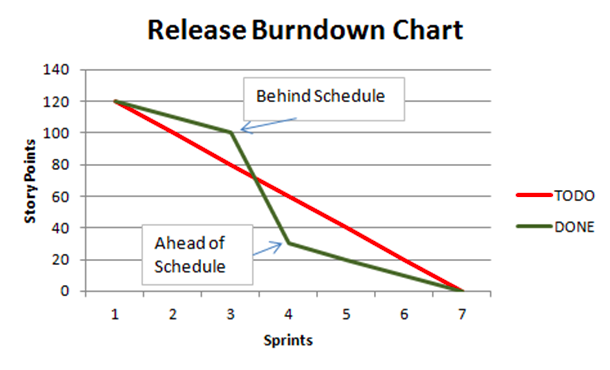
1. **Productivity Metrics**

Productivity metrics are used to measure how productive your project is. There are many different types of metrics that can accomplish this, but the two that I am going to look at are ‘code churn’ and ‘sprint burndown’, used when the priority is solely to measure how productive a project is.

**Code churn:** This measures the percentage of a software developers code that needs to be re-written (Pluralsight, 2019). In order to compute it, we measure the lines of code that have been altered in any way in the short period of time (generally around three weeks) after the code is first written. This will tell us a lot about the project on hand – high churn would suggest a difficult problem or perhaps that the projects requirements are unclear. Low churn could tell us that the engineer is prioritising speed over precision. In this case, the data required to calculate the churn percentage would be the lines of code added or changed. This is typically between 13-30%. You can access the productivity of a project by examining how the churn levels changed throughout its duration.



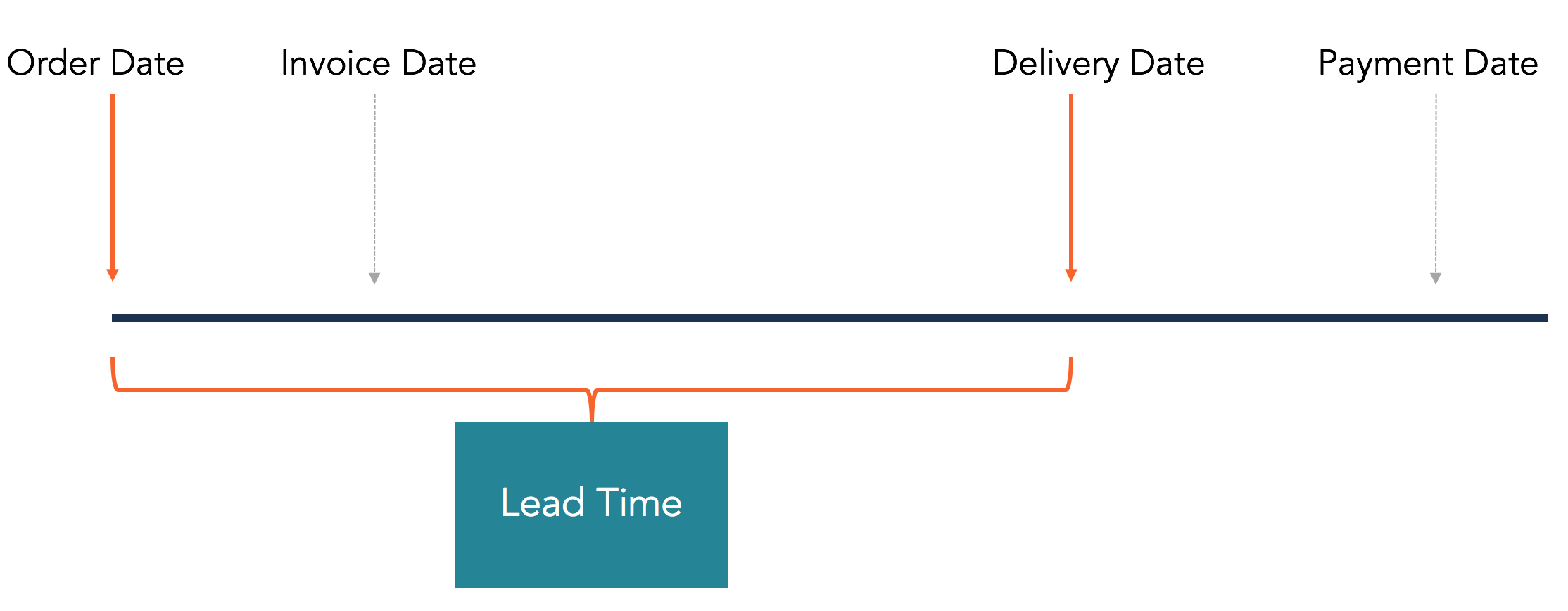
**Sprint burndown:** This gives us an a valuable insight into how a team is working. It can graphically represent the work completed versus work to be done (SolDevelo, 2019). It can help to paint a good picture of when the work needs to be completed by, and whether the team will complete the work within the ‘sprint’ (period of time within which certain tasks must be completed). This shows how productive a team of software engineers were during these sprints. Finishing early could suggest not enough work was done, and finishing late could mean that too much work was taken on. It is important to find the right balance.



1. **Process Metrics**

Process metrics are used to measure the performance of a project. I am going to look at ‘lead time’ and ‘number of commits’, although them are many different metrics that can measure performance.

**Lead time:** This measures the total time it takes for work to be completed, by calculating the duration of a project (Lynn, 2020). This includes every state that work must undergo, for example sitting in queues. It is a good metric to use to track exactly how responsive a software engineer is to their customer.



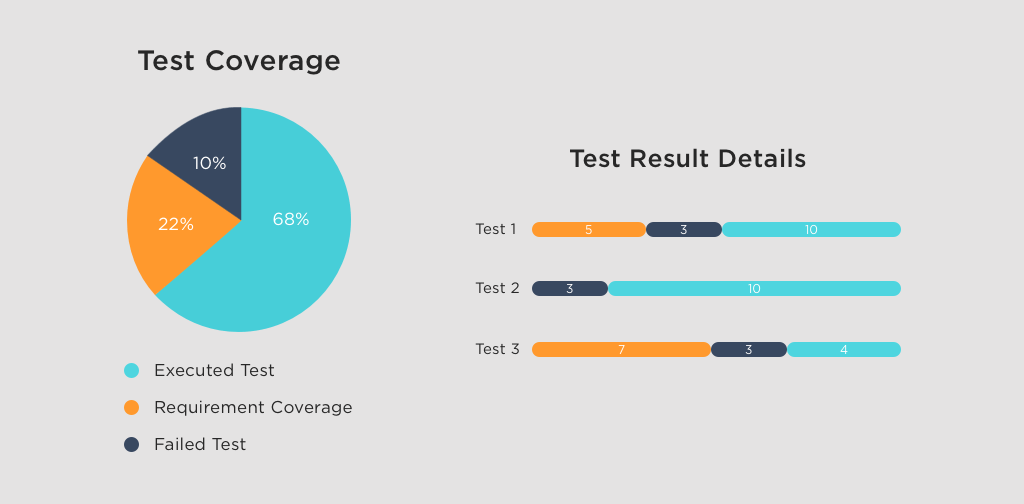
**Number of commits:** This is another useful metric to measure performance. Some software engineers tend to continuously add small commits, while others submit larger ones as they near the end of the ‘sprint’ that we previously talked about. It is important however to note that a larger number of commits doesn’t always result in a better overall performance.

1. **Quality Metrics**

Quality metrics are used to measure that the project meets the users’ needs and allows changes to occur with little to no disruption. Here I will look at both ‘change failure percentage’ and ‘test coverage ratio’.

**Change failure percentage:** This metric tracks the proportion of failures that occur in deploying a project. This is calculated by examining each individual deployment, and then taking note of whether or not it failed (Duvall, 2018). High quality performers tend to have a percentage between 0-15%, with low performers from 45-60%.

**Test coverage ratio:** This measures the ratio between the overall lines of code in a project and the total number of lines that are used in testing the code. It can be used to identify gaps in the tests, as well as the quality of the whole project. Generally around 80% test coverage is required, but higher quality projects will have a higher test coverage ratio (Shah, 2019).



# Platforms Available

Now that data has been gathered and stored, the next step is for it to be analysed for interpretation. Following the rapid growth of the data analytics industry, there have been many new platforms available to help software engineers analyse data. This knock of effect improves the software engineering process. The more efficient the platform is, means that this process takes less time, leading to faster decision making. These metrics can be difficult to measure, and often vary between different companies. However I will be looking at proven platforms that allow metrics to be analysed.

* **Personal Software Process (PSP)**

Watts Humphrey wrote a book in the 1990’s outlining how a structured development process along with constant tracking of progress would make a project more efficient. PSP focuses on manual input, which does mean it can be prone to human errors (Johnson, 2013). Software engineers fill out forms, answering self-evaluation questions based on their personal judgement. Because of risk of human errors, PSP was often resulted in incorrect conclusions. Leap Toolkit followed on from this to help deal with these inefficiencies.

* **Leap Toolkit**

Leap Toolkit addressed the data quality issues with PSP. It was able to both automate and normalise data analysis. It was required, however, for the user to manually input data from the developer, so as such was also prone to human error. It did provide a more in-depth analysis, for example regression analysis, which was not available with the PSP. In order to carry out an unbiased analysis, there should be no user input, and this was always required in some form for the Leap Toolkit. This led to further development of automated tools, Hackystat being one of them (Johnson, 2013).

* **Hackystat**

Hackystat is an open source platform that gathers data from both the client and the server. This can allow for a comprehensive, unbiased analysis. It works through the use of sensors which are attached to development tools. These sensors gather data and send it to the Hackystat server. The data in collected unobtrusively, so the users don’t even know that data is being taken. A lot of software engineers were unhappy with how accessible their work was to others on this platform.

* **Humanyze**

Humanyze provides options for companies to track what sort of work employees are doing throughout the day. Similarly to Hackystat, it operates through the use of sensors – a credit-card sized badge collects data to analyse what work is being conducted. It contains a microphone, an accelerometer and Bluetooth, to track an employees conversations, movement throughout the day and location. A lot of employees would, understandably, feel uncomfortable being tracked with this level of detail but Humanyze enforce that the purpose of these sensors is to track the company as a whole and not focus on specific individuals. It is similar to Hackystat but operates over a much wider ‘big picture’ level.

* **Other Platforms**

There are a vast array of other platforms available that are vital to measuring and assessing software engineering processes as a whole, and this number is ever-growing due to their increasing importance along with advances in technology. These include GitPrime, Sonas, Codacy, Sonar, and many others.

# Algorithmic Approaches

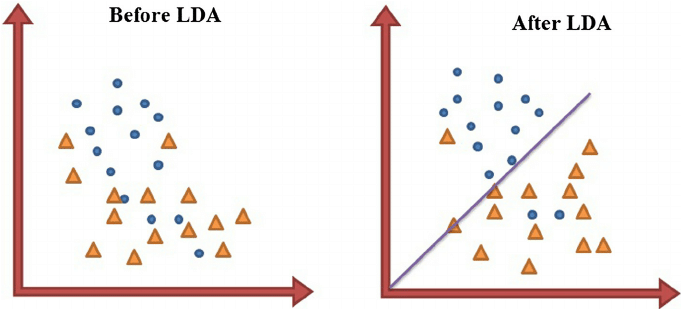
Looking at the platforms available, it is clear that they have changed from manual to more automated ones. The same can be said with the nature of analysis. Algorithms are used to produce large amounts of information in a short space of time if needs be. I am going to look at some machine learning algorithms which are used to help software engineers. These machine learning algorithms can be either supervised or unsupervised.

***Supervised learning algorithms:***

Supervised learning occurs when there is both an input and output variable. Another way of looking at this from a mathematical point of view would be *Y = f(x)* where *y*  is the output variable and *x* is the input variable. It is supervised because the software engineering acts as a guide, and should know the expected answer from the original data. After some trial and error, the algorithm should be able to approximate the variable *Y* for different input levels. I am going to look at the following two supervised learning algorithms.

1. **Linear Discriminant Analysis:**

Linear discriminant analysis (LDA) uses Bayes’ Theorem to estimate the probabilities that a new set of inputs belong to a particular classification group. It is generally used in both machine learning and artificial intelligence (Mehta, 2019).



1. **K-Nearest Neighbours:**

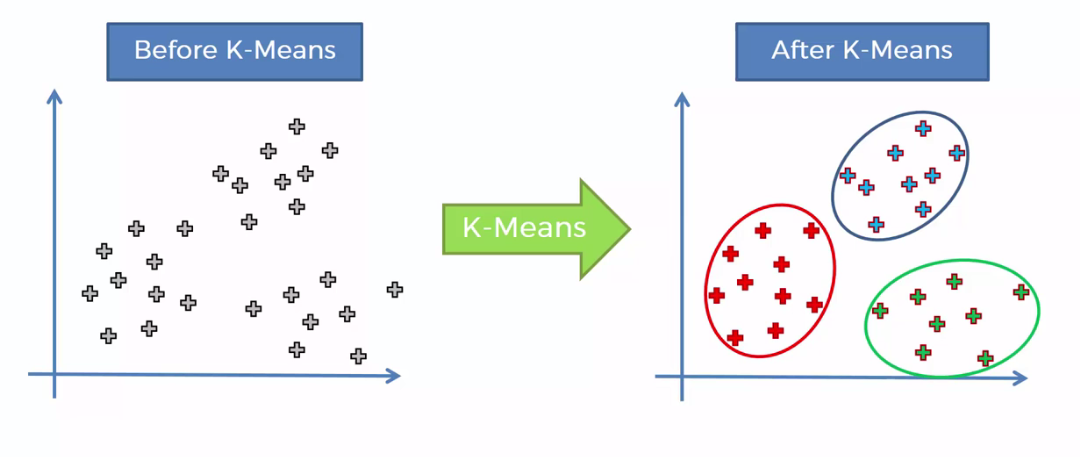
This algorithm makes no assumptions about the spread of data within each class, as it is a non-parametric method of classification. It works by finding the distances between a query and all the examples that we have in a our data. It then selects the specified number of examples (decided by the developer) that are closest to the query, and either averages these examples (in the case of regression) or gets the most frequent example (in classification)(Harrison, 2018). Choosing the right value for *K* is important in this algorithm.

***Unsupervised learning algorithms:***

As you can imagine, unsupervised learning is the opposite of supervised, where an input variable has no set output result. It is regarded as being more synonymous with machine learning, as the aim is for us to not tell the computer to do anything for these algorithms, and they learn to identify and deal with complex problems. This is difficult as there is no guarantee what the data will look like.

1. **K-Means Clustering:**

Data can be divided up into clusters based on what characteristics they have. The first step is to specify the number of clusters you want to analyse and then randomly assign a point to each cluster. You can then re-assign the data points to whatever cluster it is closest to. The central data point is re-calculated, and data points are re-assigned. This process is continued until there are no more possible improvements (Houlding, 2019).



1. **Principal Component Analysis**

This method is used for reducing the dimensionality of data. It is done while increasing interpretability but at the same time keeping information lost to an absolute minimum. It creates new uncorrelated variables that maximize variance (Jolliffe and Cadima, 2016). This method helps to identify relationships between variables and possible correlations.

Machine learning has become a lot more reachable over the last few years. This growth has opened the door to a range of new possibilities, as there are difficulties and drawbacks when dealing with human inputs. Automating algorithms will enable both faster analysis and decision making, although a human element to analysis is still required.

# Ethics

As previously discussed, the measurement and assessment of software engineering is both readily available and accessible through the use of the different platforms and algorithms used by software engineers, but it does however raise the question: Is it ethical to record all this data? I think it is currently unclear as to what exactly is right and wrong in the field of software engineering. If you look at different practises, such as law or medicine, there are certain regulations set in place. This doesn’t seem to be the case for software engineering. Large multi-national organizations collect and store amounts of data that you could even begin to count. I will look at the measures being taken with the software engineering industry to protect both the customer and employee from these giant amounts of data.

***Data collection:***

The process in which data is collected can raise some concerns. Previously, we talked about Humanyze, who are able to track what work employees conduct throughout the day using a credit card shaped sensor. It has a microphone and a Bluetooth chip in it, so detects everything the employee says, and records everywhere the employee goes. But at what cost? I for one wouldn’t like my employer being able to track my every movement, and pick up on very comment I make. Let’s be honest, there’s bound to be a lot of people caught out saying something they shouldn’t be saying! And what happens to this data once it is recorded? Does it get deleted at the end of every day?

I for one think that a balance need to be struck between gathering useful data, and taking action to respect the privacy of the employee while doing so. It is also important to keep in mind that only data that will benefit the software engineering process should be collected. Any additional, personal data should not. Unfortunately it is extremely difficult to separate raw data into these two groups without delving into the data itself. This is something that needs to be looked at in the near future.

***Regulation:***

The general Data Protection Regulation (GDPR) was released by the EU in 2018. The aim of this was to replace the previous Data Protection Act. The main point of this was that everyone has to consent to their data being taken, and that can be withdrawn at any time if that person wishes to do so. Following on from this, deletion of data can also be requested at any time. This is very important and I think that this regulation is certainly a step in the right direction. Companies that do not adhere to these rules can be fined 4% of their income, or up to €20 million. As a result from this, any companies that are looking to measure their software engineering process must do so in a way that complies with this new legislation (Citizens Information, 2018).

***Data usage:***

As well as data collection, there are also ethics around the way that data is used by companies for analysis. Some of the concerns around this is to do with keeping analysis impartial and fair. One employee shouldn’t be rewarded for writing more code that another. The extra code may not be up to the required standard, and it would be unethical for an employee to write more just because they know they are going to be rewarded for doing so.

Following up on this, data is collected on people that are applying for jobs for example. It is important that this data does not sway the interview or managers opinion, for example where the applicant is from, or the gender or the applicant. This creates a very fine line ethically when using this type of data.

# Conclusion

To finish off this report, I think I have outlined in detail the different ways in which software engineering, and software engineering process can be both measured and assessed. I also looked into the risks and limitations associated with this, and tried to illustrate my own personal view from an ethics point of view.

Management can track progress, assess efficiency and use data in order to make decisive, correct decisions. Software engineering has come a long way since 1968 and there are now various platforms and algorithms in place that can analyse software engineering metrics. I have also looked at the significance that these metrics add to projects and to the software engineer themselves. It is clear that the main focus of companies that partake in the measurement of software engineering should be how the analysis will help their employers, and nothing more. It is important to ensure that consent is given to those taking data.

# Bibliography

Altvater, A., 2017. *What Are Software Metrics? Examples & Best Practices*. [online] Stackify. Available at: <<https://stackify.com/track-software-metrics/>> [Accessed 26 November 2020].

Citizens Information, 2018. Overview of the General Data Protection Regulation. [Online] Available at: [https://www.citizensinformation.ie/en/government\_in\_ireland/data\_protection/overview\_ of\_general\_data\_protection\_regulation.html](https://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_%0Dof_general_data_protection_regulation.html) [Accessed 27 November 2020].

Duvall, P., 2018. *Measuring Devops Success With Four Key Metrics | Stelligent*. [online] Stelligent. Available at: <<https://stelligent.com/2018/12/21/measuring-devops-success-with-four-key-metrics/>> [Accessed 27 November 2020].

Gadagkar, S., 2020. [online] Study.com. Available at: <<https://study.com/academy/lesson/software-engineering-definition-process-methods.html>> [Accessed 26 November 2020].

Guru99.com. 2020. *What Is Software Engineering? Definition, Basics, Characteristics*. [online] Available at: <<https://www.guru99.com/what-is-software-engineering.html>> [Accessed 26 November 2020].

Johnson, P. M., 2013. Searching Under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.

Harrison, O., 2018. *Machine Learning Basics With The K-Nearest Neighbors Algorithm*. [online] Medium. Available at: <<https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>> [Accessed 27 November 2020].

Houlding, B., 2019. STU3011 Notes. [Online] Available at: <https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides9.pdf> [Accessed 27 November 2020].

Lynn, R., 2020. *Lean Metrics To Improve Flow | Planview*. [online] Planview. Available at: <<https://www.planview.com/resources/guide/what-is-agile-program-management/lean-metrics-improve-flow/>> [Accessed 27 November 2020].

Mehta, A., 2019. *Everything You Need To Know About Linear Discriminant Analysis*. [online] Digital Vidya. Available at: <<https://www.digitalvidya.com/blog/linear-discriminant-analysis/>> [Accessed 27 November 2020].

Pluralsight.com. 2019. *Guide | Understanding Code Churn*. [online] Available at: <<https://www.pluralsight.com/blog/tutorials/code-churn>> [Accessed 26 November 2020].

Shah, H., 2019. *Why Test Coverage Is Important In Software Testing?*. [online] Simform.com. Available at: <<https://www.simform.com/test-coverage/>> [Accessed 27 November 2020].

SolDevelo Blog. 2019. *7 Effective Metrics For Successful Scrum Projects Delivery - Soldevelo Blog*. [online] Available at: <<https://www.soldevelo.com/blog/7-effective-metrics-for-successful-scrum-projects-delivery/>> [Accessed 26 November 2020].

T. Jolliffe, I. and Cadima, J., 2016. *Principal Component Analysis: A Review And Recent Developments | Philosophical Transactions Of The Royal Society A: Mathematical, Physical And Engineering Sciences*. [online] Royalsocietypublishing.org. Available at: <<https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202>> [Accessed 27 November 2020].