MEASURING SOFTWARE ENGINEERING

CSU33012

ADAM CARTY

18321109

Adam Carty
18321109

ABSTRACT:

This report aims to investigate and document the ways in which the software engineering process can be quantified, measured and assessed. This is done considering the following topics;

- measurable data
- computational platforms available for this analysis
- algorithmic approaches
- ethical concerns relating to this form of analysis

This report is presented with a mixture of academic papers and online resources, both of which are referenced throughout and within the bibliography at the end, along with information and other resources disseminated throughout the CSU33012 module.

INTRODUCTION:

Before beginning to attempt to assess and measure software engineering, one must delve deeper into what the term “software engineering” actually describes. The first instance of this term dates back to the 1950’s in the Massachusetts Institute of Technology. A computer scientist and systems engineer named Margaret Hamilton first coined the term whilst working as part of the Apollo space missions on the development of their guidance and navigation systems (1, Utah State University). Hamilton was acutely aware to the complexities and intensities associated with the development of software and so she wanted those working in this field developing software to receive the credit and recognition due to them.
Software engineering can be defined as, “the process of analysing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages.” (2, Techopedia). It is the application of engineering principles to the development of software. It is a broad term which encompasses a wide range of tasks pertaining to the development of software. Ian Somerville defines it as “an engineering discipline that is concerned with all aspects of software production”. Software production consists of these four main activities:

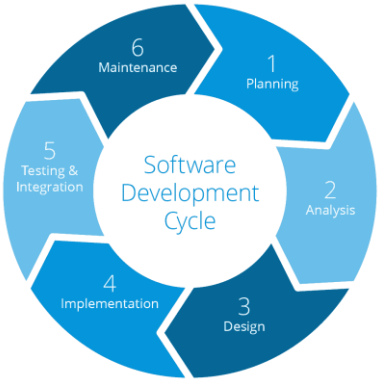
1. Software specification
2. Software design and implementation
3. Software verification and validation
4. Software maintenance and evolution

This is where the challenge arises, a method or approach must be developed which can be applied to all the aforementioned tasks relating to software development. However, the software development process can vary heavily from project to project and individual software engineers may have very different coding techniques or styles which can provide a huge point of comparison or even contrast. It is difficult to definitively determine which is the optimum or best techniques or styles.
There are multiple quantifiable and non-quantifiable variables and factors that are involved in the development process. So, methods for measuring and assessing software engineering must be developed which can be applied generally to all software engineering tasks and techniques.
In order to do this, one must first look at the actual process itself.

SOFTWARE ENGINEERING PROCESS:

As mentioned, software development can vary quite heavily from project to project. However, the software development process will remain consistent between projects and software engineers. A process must contain

the four previously mentioned attributes; software specification, design and implementation, verification and validation and finally maintenance and evolution.

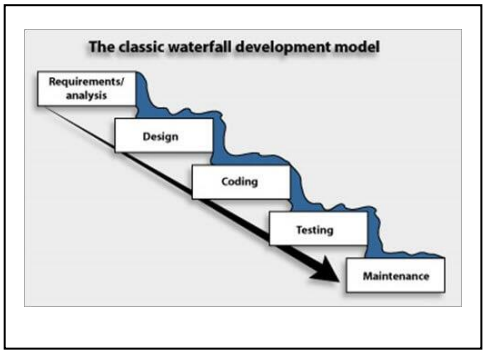


This sequence of tasks is referred to as the Software Development Life Cycle. This is a very broad description of the software development process, but it gives insight into steps and tasks involved in the development of each project. There are several different software process models, this report will briefly expand on some of the most common of these; Iterative Development, Waterfall and Agile (4, Osetskyi, 2017).

ITERATIVE DEVELOPMENT:

Iterative development is a method in which the development of a large application/solution is broken down into smaller, more manageable components/chunks. Each of these small components is designed, developed and tested continuously adding to the current application until it is ready or has the functionality which can be deployed to the customer/user (3, TechTarget).

WATERFALL:



The waterfall model is a methodology for software development in which progress flows sequentially towards the ultimate solution. In this model, each phase must be completed before moving on to the next one. Therefore, there is no overlap between phases (5, TutorialsPoint). This is a simple and easily understood methodology however, the solution is more rigid and not suitable for ongoing changing requirements.

AGILE:

The agile software development methodology is any development method which encompasses and is aligned with the Agile Manifesto (6, Agile Manifesto). This the is centred around iterative development with emphasis on communication between teams to assess the requirements of the software application and continuously deliver solutions to these requirements until the complete solution can be delivered. This allows the user to view the solutions in development and approve/disapprove the software at each stage of the development process. This can improve the solutions as it ensures the software encapsulates the full functionality required by the user. Two sub methods of the Agile methodology are Scrum and Extreme.

MEASURABLE DATA:

Data is an essential part of any analysis and assessment. The data collected can inform innovation and improve quality when analysed correctly. So, it is extremely important to establish the different types of data which can be collected from the software engineering process and how this data can inform the user. Measuring software engineering poses more of challenge than quantifying other disciplines of work. Productivity or value produced cannot be measured as simply as the number of hours work or the volume of units produced. Metrics to quantify software engineering are tools and analysis used to measure performance. (8, Fenton & Martin, 1999) These metrics must encapsulate not just the quantity but also the quality of the software products produced. They are needed to improve the way we monitor, control and predict various attributes of the software engineering process. (8, Fenton & Martin, 1999)

The use of metrics is traced back to the 1960's where the dominant form of measurement was Lines of Code. As mentioned, software engineering is not so easily quantified as volume of units produced just as this metric would seem to reduce it to. This showed a complete disregard for the development of measurement techniques and metrics to accurately quantify the software engineering process which encapsulates all aspects of it. Whilst the academic work around the subject quickly developed, R.L Glass described it best, "What theory is doing with respect to measurement of software work and what practice is doing are on two different planes, planes that are shifting in different directions." (9, Glass 1994) What developments have been made to rectify this discrepancy between theory and practice?

There are three distinct metric types by which software engineering may be measured as detailed by Xeno and Stavrinoudis; Project Data, Process Data and Product Data. These three metrics can be subdivided into either entities or attributes. The attributes can be simplified further internal attributes and external attributes. External attributes relate to the product links to its environment while contrasting this, internal attributes are "measured directly by examining the product on its own irrespectively of its behaviour" (7, Xeno & Stavrinoudis). Both category of attributes poses their own distinct complications. Namely, external attributes can be quite subjective and require manual user inputs. However, internal attributes are relatively easy to collect, but pose difficulties with respect to their interpretation (7, Xeno & Stavrinoudis).
With regards to the different types of metrics outlined above, project data is considered the most accessible data. The main entities analysed in this metric are tools, personnel and environment. This metric is only concerned with the resources of the project and can be quantified in a similar way to the measurement of projects in other industries. Despite this metric not being unique to software engineering, it is still an essential measure which must be considered (7, Xeno & Stavrinoudis).
The next metric detailed by Xeno and Stavrinoudis is process data which simply is a measure of the steps and tasks involved in creating and producing a software product. This metric essentially quantifies each of the previously mentioned steps in the software development process namely; specification, implementation, validation and maintenance. The internal attributes related to process data are time and effort, along with several others pertaining to the project. For instance, test coverage. (7, Xeno & Stavrinoudis)
The third and final metric as outlined by Xeno and Stavrinoudis is product data which is also considered the most complex metric of the three to define and measure. These metrics relate directly to the product, its quality deliverables and manuals. (7, Xeno & Stavrinoudis)

Contrasting with Xeno and Stavrinoudis, Stephen A. Lowe raises his concerns as to the relevancy of metrics in software Engineering at all. Lowe suggests previously developed metrics are irrelevant and a software product should be measured or quantified based on customer satisfaction and the value it delivers to a business. However, these forms of measurement are far more subjective than the use of metrics. Lowe also notes that one should remember that with regards to software development, almost each project is unique or a "snowflake" as he described them. This implies that metrics are only comparable within their specific projects and not comparable externally to other software development projects. (10, Lowe)
In his article entitled, "9 metrics that can make a difference to today's software development teams", Lowe details nine metrics applicable to the industry;

1. Leadtime – how long it takes to go from forming an idea to completing the final software.
2. Cycle time – the length of time it takes to change a software system and deliver that change
3. Team velocity – how many units of software the team typically completes in an iteration
4. Open/Close rates – how many production issues are reported and closed within a specific time period
5. Mean Time between Failure
6. Mean Time to recover/repair
7. Application crash rate – how many times the software fails divided by the amount of times it was used

8. Endpoint incidents – the number of endpoints that have been impacted by a virus infection over a given period of time
9. MTTR – mean time to repair – the time between finding a security breach and delivery a working solution

Kan outlines in his paper entitled, "Metrics and Models in Software Quality Engineering" that there are three distinct metrics which are used to effectively analyse and measure the software engineering process; product metrics, process metrics and maintenance quality metrics. (11, Kan 2003)

PRODUCT METRICS:

With regards to product metrics, Kan outlines several ways in which this metric is measured, and the data related directly to this;

- Mean time to failure (MTTF)
- Defect density
- Customer problems
- Customer satisfaction

MTTF and Defect Density are the two metrics which relate directly to ascertaining the intrinsic value of product quality. Both measures have complications. With regards to MTTF, gathering accurate and meaningful data poses a challenge. Gathering data around time between failures is costly in resources such as time and money. It requires the developer to record the occurrence time of each software failure to a high degree of accuracy. (11, Kan 2003) Also, with regards to the defect density metric, complications arise due to the requirement of the developer to know the software size which can often be measured as lines of code (LOC) or thousands of lines of code (KLOC). (11, Kan 2003) Issues arise when the developer must find this factoring the difference between physical lines and instruction statements, comments and differences between coding languages. Customer Problems can be expressed as problems per user month (PUM). This is given by the sum of true defects and non-defect-orientated problems for a given time period divided by the total number of license - months of the software during the period. This is a very useful metric to ascertain product quality given that it incorporates usability problems for the user. Customer satisfaction metrics are more qualitative and simplistic and can be measured via customer surveys. (11, Kan 2003)

MEASUREMENT OF PROCESS:

With regards to process metrics, Kan outlines several ways in which this metric is measured, and the data related directly to this;

- Defect density during formal machine testing
- Defect arrival pattern during formal machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

Defect density during formal machine testing is a good indication as to the quality and useful for release-to release comparisons. (11, Kan 2003) When comparing a release to a previous iteration, if the defect rate is the same or lower and the defect rate did not deteriorate then the outlook is positive. However, if the defect rate deteriorates, the converse is also true.
With regards to the defect arrival pattern, analysing the pattern of defect arrivals or times between failures can give us information that we might not necessarily have picked up on from the total defect density. (11, Kan 2003) The overall aim of this metric is to get the defect arrivals to stabilise at a very low level.
When analysing phase-based defect removal pattern, it is similar to the aforementioned defect density metric. (11, Kan 2003) However, this metric transcends phases and provides an overall pattern of defects in the development process irrespective of the phase.

Defect Removal Effectiveness can be given by defects removed in a given development phase divided by the defects latent in the product. (11, Kan 2003). The denominator is an estimate of actual defects found summed with a predicted value of defects which will be found later in that given phase. (11, Kan 2003)

MEASUREMENT OF PRODUCT QUALITY:

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

The fix backlog and backlog management index (BMI) is simply a measure of the rate of defects found and the rate of defect solutions becomes available. This metric is given by the number of problems closed during the month over the total number of problem arrivals during the month. (11, Kan 2003)
The fix response time is an essential metric in the software development process. It can ascertain the ability to mee deadlines when fixing errors. This is a crucial element of the software maintenance process. In critical situations, an error may debilitate a business and they cannot operate until this error is resolved. It is also essential again in the development process as a product may be halted in an iteration by a single error and may not progress until its resolution. This metric can give excellent insight. It is given by mean time to resolve a problem.
With regards to percentage delinquent fixes metric if the turnaround time notably exceeds the required response time, it is classified as a delinquent. The formula excludes open problems and is in reality not a metric for real-time delinquent management.
fix quality keeps a record of all fixes which in turn, were defective themselves. These defective fixes can decimate the level of customer satisfaction with a given software product and can cause major delays to the product's release or in more extreme circumstances, financial loss to the business. The metric is a percentage of all solutions to defects which turn out to be defective. There is unrest with regards to the use of a percentage for this metric, the data can be skewed due to large defects and high volumes of solutions.

PLATFORMS AVAILABLE:

Collecting the data detailed above is only one step in the measurement of the software engineering process. What gives this data its inherent value is its analysis and interpretation. To complete analysis on such large data sets as are previous outlined by hand would prove to be hugely inefficient in resources to the user such as time and money. It is essential that there are efficient computational platforms which can perform the analysis necessary to extract useful information from the data. Time is of the essence in this regard, as the more efficient computational platforms can allow for quicker extraction of tangible information which can in turn inform the decision making and responses of the business to problems or inefficiencies within their software engineering process.
Many computational platforms have been produced and released which are broadly used to measure and quantify software engineering. Some of these platforms include;

PERSONAL SOFTWARE PROCESS (PSP):

PSP was created and developed by Watts Humphrey as a structured software development process which enabled users to assess and measure their progress and facilitates the improvement of performance by tracking their predicted development code against the actual development code. (12, Humphrey, 2000)
Humphrey based PSP on a previously created process known as the Capability Maturity model (CMM). He wanted to provide developers with a structured disciplined approach to writing programs and software development. (13, Humphrey, 1995) PSP is structured with three distinct stages; planning, development and

post-mortem. The developer can manually input their data and once PSP has established a sufficient data set, analysis can be performed, and the user can asses their performance. However, the reliance on manual data inputs from the user led to accuracy issues. PSP focussed on certain key metrics including productivity, defect information, reuse percentage and earned value metrics. (13, Humphrey, 1995)

LEAP TOOLKIT:

The Lightweight, Empirical, Anti-measurement dysfunction and Portable (LEAP) software process measurement toolkit was created as an advancement and extension of the previously outlined Personal Software Process (PSP). LEAP Toolkit sought to solve some of the problems associated with PSP. It provided further and more comprehensive analysis that had previously not been built into PSP. This included statistical methods such as regression analysis along with the automation and normalisation of data analysis. Another big change implemented to LEAP Toolkit from PSP is the creation of a repository for which users store personal process data which developers could bring with them from project to project. However, LEAP Toolkit could not fully eradicate the requirement of manual user input and so, human error remained a problem within LEAP Toolkit as it was in PSP. With the goal of carrying out unbiased analysis, an alternative to manual user input was required. This goal is what drove the development of a new computational platform, Hackystat. (14, Johnson, 2013)

HACKYSTAT:

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. (15, Hackystat, n.d.) Hackystat was developed in order to eliminate the previously outlined problems associated with the other computational platforms regarding manual input and human error. Hackystat removes the element of the user inputting high-level goals related to a project and deciding on what data is to be collected to encapsulate and analyse these goals fully. Hackystat automatically collects data from both sides; the client and server-side. (15, Hackystat, n.d.) This is very useful as it can be highly efficient as no time or resources are wasted manually inputting data to the computational platform. It is easily integrated with git software such as GitHub and can provide invaluable insights and inform decisions regarding the software engineering process. A possible drawback of this, is that users may not be comfortable with their data being taken at times unknown to them and analysed automatically.

HUMANYZE:

Humanyze extends the measurement of only the software engineering process and provides sensors which encapsulate and measure all aspects of the user's day. The sensors contain microphones which track speech and tone of conversations, accelerometers to track the user's movements, infrared to detect who the user is speaking with and Bluetooth to track location. (16, Goyal, 2018) Again, this automates the collection and analysis of data and removes the human error aspect. Humanyze is established in over 50 Fortune 500 companies involved in a wide range of industries including Technology. However, the same drawback applies to Humanyze as it did to Hackystat. That is, employees may not feel comfortable to constantly being measured and having their data analysed and assessed. (16, Goyal, 2018)

VISUALISATIONS OF DATA IN COMPUTATIONAL PLATFORMS:

GITHUB:

GitHub is a web-based version control and collaboration platform for software development. GitHub is open source whose platform facilitates the measurement of the software engineering process for free via the developer data. A simple, but very effective tool available on GitHub is the member's contribution heat map. This displays data behind the user's total number and frequency of commits over a specified period of time. The REST API in GitHub is particularly useful as it shows data related to specific projects for which the user is

involved. It shows the frequency and number of commits, the number of developers who have contributed to the project and a breakdown of each developer's contributions in the form if commits. This data can be extracted and analysed using the functionality of statistical programming languages such as R. From there, visualisations can be produced to aid and inform in decision making regarding the software engineering process.

CODE CLIMATE:

"Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous and ubiquitous". (17, Codeclimate.com, 2018) Code Climate is based on the foundations of Hackystat. However, it was created to resolve or reduce the problems relating to Hackystat regarding manual input and human error. It provides an automated data collection service and facilitates automated data analytics on the collected data. Code Climate can also be integrated and used with git-based applications such as GitHub where it can display developer data like code coverage, technical debt and a progress report. (18, Carpenter, 2018)

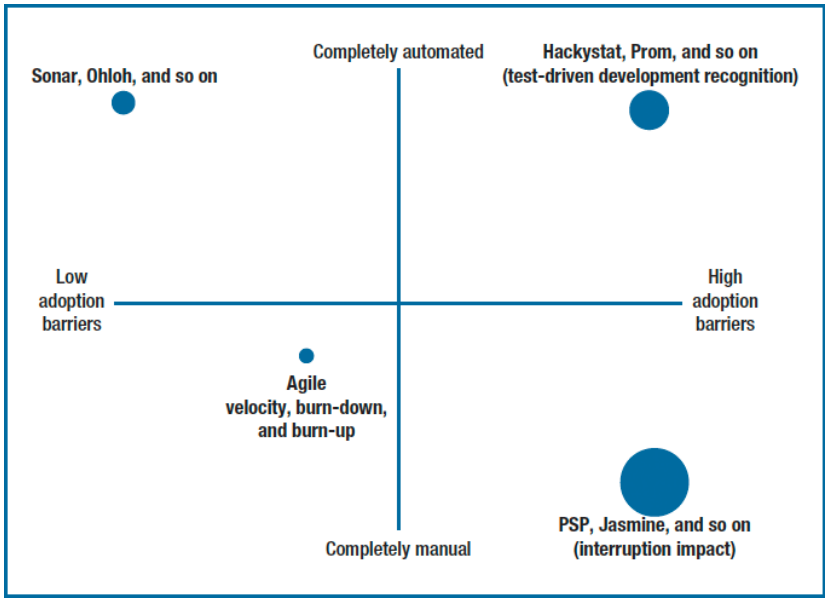


Figure 1: classification of software analytics platforms from "Searching under the Streetlight for Useful Software Analytics."

ALGORITHMIC APPROACHES:

The data required to measure software engineering has been detailed above. Also mentioned was that data collection is only one element of measuring the software engineering process, the other one which will be discussed is the analysis of the collected data. This analysis is what gives the data its value. It is of utmost importance to have clear analysis in order to extract meaningful and tangible value which can in turn, inform decisions regarding the software engineering process. Without such analysis, the data would essentially be useless.

COMPUTATIONAL INTELLIGENCE:

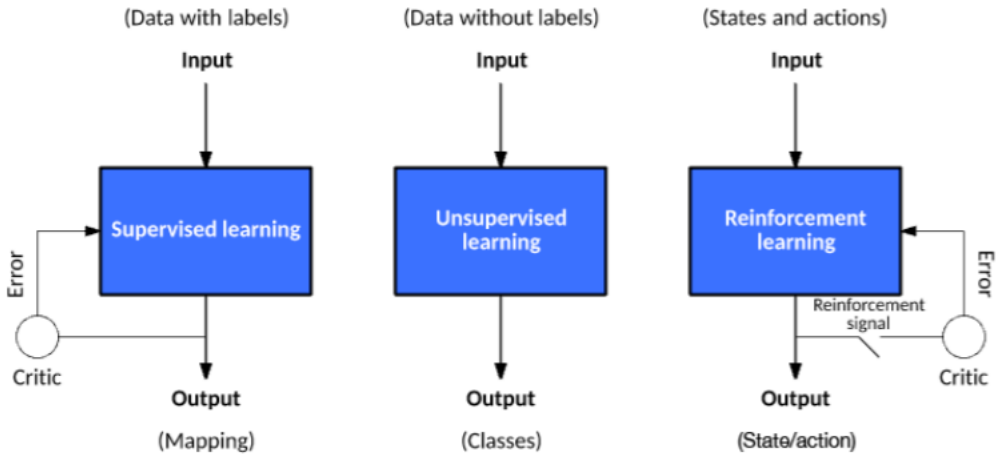
CI is a tool employed by software companies to gain a more in-depth insight into the data collected regarding the software engineering process and its measurement. Manual analysis of these large data sets has long since been replaced with algorithms which improve efficiency and accuracy. These algorithms can also be automated as seen within the aforementioned computational platforms which can have these algorithms running and performing analysis on these data sets in the background. Algorithms can be extremely useful and

allow the extraction of valuable information from data sets in very short timeframes. They can be extremely useful in the detection of structures patterns and trends within these data sets. There are five main components of CI (23, IEEE, 2018);

1) **Fuzzy Logic:** This technique is utilised to alter the binary nature/bluntness of Boolean statements which are either 'true' or 'false'. It is employed to introduces the concept of partial truths. This has a wide range of applications.
2) **Neural Networks:** they provide a framework for machine learning algorithms to process data inputs and work together. There are three components to NNs, the first processes the information, the next sends the signal and finally one to control the signals when sent. (23, IEEE, 2018)
3) **Evolutionary Computation:** these are a set of algorithms which are applied for global optimisation. These algorithms are based on the theory of natural selection. These algorithms are applied to a wide range of problems where traditional mathematical techniques are inadequate.
4) **Learning Theory:** This technique is applied in order to encapsulate the emotional and cognitive effect on human reasoning. It allows algorithms to make predictions from what it learns.
5) **Probabilistic Methods:** these methods compute outcomes of systems which are based on and defined by randomness. These methods provide a range of possible solutions within a certain level of confidence.

This report will focus on a particular technique, machine learning , the three categories and the algorithms associated with this. Machine learning has grown hugely in popularity in recent years. It is essentially derived from the concept of pattern recognition and fundamentally grounded in the theory that computers can learn without being programmed. Machine Learning can be divided into the following three categories (19, Brownlee, 2019);

1. **Supervised Learning**
2. **Unsupervised Learning**
3. **Reinforcement Learning**



SUPERVISED LEARNING:

Supervised learning is a technique where an algorithm essentially function which can map particular inputs to desired outputs. This algorithm is classified as 'supervised' due to the fact that it undergoes a form of "training". The developer or data scientist will input a "training dataset" into the algorithm with known outputs for the inputted data. This acts as a guide to help "teach" the algorithm. The algorithm iteratively

predicts the outputs based on the inputted data. This is repeated several times until the algorithm reaches desired levels of accuracy. (20, MachineLearningMastery) Examples of supervised learning algorithms which can be useful to measure software engineering include the following;

LINEAR DISCRIMINANT ANALYSIS:

LDA is a classification technique which facilitates the classification of multivariate data into groups. A classification decision bound is defined by the algorithm and then classifies multiple instances of the data by the decision boundary. This is a popular and widely used algorithm in ML and AI. (21, Houlding, 2019)

K-NEAREST NEIGHBOURS:

K-Nearest Neighbours is classification method which is non-parametric. The algorithm essentially receives a point or data point and classifies it using pre-existing data which was already available to the algorithm. The number of nearest neighbours used to classify the point is selects and then the algorithm selects the classification of the new point based on which class is most prevalent of the x nearest neighbours. (21, Houlding, 2019)

UNSUPERVISED LEARNING:

Unsupervised Learning is similar to supervised learning in the regard that the goal is to have a machine learn to identify complex processes and patterns. However, the difference is that unsupervised learning is done without being taught by the developer at all. A test dataset is input but, the outputs are unknown and there is more reliance on the accuracy of the algorithm. (21, Houlding, 2019)

PRINCIPAL COMPONENT ANALYSIS:

PCA is a method applied to large datasets with a lot of dimensions, this algorithm essentially reduces the dimensions of the data set as it aims to encapsulate which variables account for the most amount of variance within the dataset. This can help identify relationships between variables and correlations. (21, Houlding, 2019)

K-MEANS CLUSTERING:

Clustering algorithms can be very useful in measuring software engineering. Essentially the algorithm separates a group into k clusters based on similar characteristics. This identifies structures within the data set as similar data is clustered into distinct groups based on these characteristics. (21, Houlding, 2019)

REINFORCEMENT LEARNING:

Reinforcement learning is based on trial and error. This type of learning trains the algorithm to make decisions based on interactions with its environment. Each action is weighted as a penalty or reward and the algorithm's goal is to maximise its long-term reward whilst also minimizing its long-term penalties. This form of learning is centred around decision making.

ETHICS:

The process of measuring software engineering, as previously discussed entails the collection, storage and processing of an individual's data. This process can give the impression to employees that they are under surveillance in the workplace which van lead to unease or discomfort with developers. However, this measuring of software engineering is essential for businesses in order to maintain their competitive edge and dominate their market share. So, is this sort of data analysis ethical?

There are certain issues with regards to the legality surrounding data collection. In May of 2018, the EU introduced the General Data Protection Regulation (GDPR) which has greatly influenced the way in which data can be handled. It has strengthened the laws relating to the protection and safeguarding of an individual's private data. Any company which collects an entity's private data must maintain their GDPR compliancy.

Another major change introduced by GDPR is that companies must obtain the consent of the entity for whom they wish to collect the data from. They must also specify for what purpose this data is being collected for. This can heavily influence the data which a company may collect and process in order to measure software engineering.

The other ethical concern with regards to this sort of analysis and the depth to which it extends is privacy. This particular issue can be quite controversial as has been seen in the past. Where should the line be crossed? It has been seen how widespread of an issue it is when one considers the Cambridge Analytica scandal in 2018. In this instance, it was found that Cambridge Analytica was collecting, storing and analysing personal data from millions of Facebook users without their knowledge nor consent. This was branded grossly negligent on Facebook's part and highly unethical. The question "is it ethical?" can be murky at times and situations can be interpreted differently from entity to entity. For example, Humanyze which is a computational platform which was previously outlined above provides data analytical solutions to businesses through the use of their sensory badges. As mentioned, these badges collect various forms of data including speech from users and those around them, personal data including location and movements. It automates the collection and analysis of this data and does so behind the scenes without visibility from the user. (16, Goyal, 2018) Humanyze is established in over 50 Fortune 500 companies so this form of analysis and monitoring is growing in popularity. However, id this taking it too far? I would argue yes, this form of constant monitoring would create a sense of unease and discomfort amongst staff which would lead to counter-productivity in my opinion. In this extreme example, there is no longer a division between what should remain private and what data should the company be allowed to collect. The badges are worn by employees throughout the day and so even the bathrooms would cease to be private.

A complication can also arise when companies attempt to base decisions solely on their processes which measure software engineering. The algorithms they employ to analyse the data which they have collected can only encapsulate so much of the process. Certain aspects are very difficult to quantify. These can include "soft skills" such as communication, teamwork or leadership. These skills can improve the team's productivity and efficiency but may not necessarily be reflected in the data or the analysis. The algorithms employed in this type of analysis do not understand what the data is, they simply seek out trends and patterns which underly the data. This can cause ethical concerns as decisions could be made to fire what appears to be an inefficient developer, according to the algorithms. However, they could be highly valuable developers which possess valuable soft skills which aid greatly in the overall team performance. An algorithm cannot recognise this and it could give rise to misleading analysis.

This is an issue which is growing in concern. I think companies must exercise extreme caution as they tread a very fine line between what data they may collect in order to establish a competitive edge and more importantly what they should not do to maintain ethical standards. Professional engineers must conduct themselves in an ethically and morally responsible way. Principles like confidentiality, honesty and integrity are key to the successful performance of a software engineer (22, Sommerville, 2011).

CONCLUSION:

To conclude, this report has outlined the ways in which the software engineering process can be measured and assessed with regards to the measurable data, computational platforms available and the algorithmic approaches. It has also discussed the importance of this quantification along with the limitations, risks and responsibilities associated with this. As shown, there is no perfect approach to measure software engineering. It is not an easy process, but great strides have been made in recent years. However, many things are still left unmeasured with regards to a developer's 'soft skills' like communication. Attempts have been made to quantify these factors such as Humanyze. However, how can these methodologies and processes progress in an ethical manner? This is one of the main problems facing the measurement of software engineering. Great progress has been made, but there is still work to do before all aspects of the software engineering process can be encapsulated completely.

BIBLIOGRAPHY:

1. https://web.archive.org/web/20160917191701/https://cs.usu.edu/news/pioneers-in-cs-margaret-hamilton
2. https://www.techopedia.com/definition/13296/software-engineering
3. https://searchsoftwarequality.techtarget.com/definition/iterative-development#:~:text=Iterative%20development%20is%20a%20way,and%20tested%20in%20repeated%20cycles.
4. Osetskyi, V., 2017. SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral. https://medium.com/existek/sdlc-models-explained-agile-waterfall-v-shapediterative-spiral-e3f012f390c5
5. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
6. http://agilemanifesto.org/
7. Xeno, M. N., & Stavrinoudis, D. (2008). Comparing internal and external software quality measurements. Proceedings of the 8th Joint Conference on Knowledge-Based Software Engineering, 115-124.
8. Fenton, N. & Martin, N., 1999. Software metrics: successes, failures and new directions. The Journal of Systems and Software, Volume 47.2, pp. 149-157.
9. Glass, R., 1994. A tabulation of topics where software practice leads software theory. Journal of Systems Software, Volume 25.
10. Lowe, S. A., 2018. Why metrics don't matter in software development (unless you pair them with business goals) https://techbeacon.com/whymetrics-dont-matter-software-developmentunless-you-pair-them-businessgoals
11. Kan, S. H. (2003). Metrics and Models in Software Quality Engineering. Addison-Wesley Professional.
12. Humphrey, W. S., 2000. The Personal Software Process (PSP). https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283
13. Humphrey, W. S., 1995. A Discipline for Software Engineering. Addison-Wesley
14. Johnson, P. M., 2013. Searching Under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.
15. Hackystat, n.d. A framework for analysis of software development process and product data. https://hackystat.github.io/
16. Goyal, M., 2018. Protecting individual privacy is important: Ben Waber, cofounder, Humanyze. https://economictimes.indiatimes.com/tech/hardware/protecting-individualprivacy-is-important-ben-waber-cofounder-humanyze/articleshow/64045690.cms
17. Code Climate. (2018). About Us: Code Climate. https://codeclimate.com/about/
18. Carpenter, W., 2018. Code Climate: How it Works and Makes Money. https://www.investopedia.com/articles/investing/022716/codeclimate-howit-works-and-makes-money.asp#ixzz5XCtdZq26
19. Brownlee, J., 2019. Master Machine Learning Algorithms. s.l.:s.n.
20. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
21. Houlding, B., 2019. STU3011 Notes. https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides3.pdf
22. Sommerville, I., 2011. Software Engineering. 9th Edition ed. s.l.:Pearson Education.
23. IEEE, 2018. What is Computational Intelligence? https://cis.ieee.org/about/what-is-ci