Measuring Software Engineering Report

Introduction

Measuring Software Engineering means gathering data, analysing the data and essentially seeing the amount of output gained for a certain amount of input. More output for the same amount of input simply means greater productivity.

The aim of this report is to examine the ways in which the software engineering process can be measured and assessed. This will be done by looking at the types of measurable data available and the computational platforms available to perform these measurements, as well as looking at the algorithmic approaches and ethical concerns surrounding this type of analysis.

Why measure Software Engineering?

In order to measure Software engineering, we first must interpret what the term means. Software Engineering is used to design, develop and test software systems. It is usually applied to the field of engineering physical systems. Software engineering is used for large and intricate software systems, where it applies a structured, disciplined approach to programming with the goal of improving the quality, time and efficiency of the software system.

The first Software Engineering conference was held and sponsored by NATO in 1968 in response to the 'Software Conference' at the time. Better quality and reliability of software systems were the main topic of this conference, as the inconsistency and unreliability in software development was highlighted. Fast forward to today, and the Software Engineering Institute (SEI) is the organization that works to improve the efficiency of software engineering projects in the U.S. (WhatIs, 2019).

The main reasoning behind measuring software engineering is to do with the management of business. Managers want to be able the measure the productivity and performance of their software engineers, but this is easier said than done as we learned from Frederick P. Brooks, "Software entities are more complex for their size than perhaps any other human construct because no two parts are alike" (Brooks, 1987). The assessment and improvement of software processes are a vital part of industry in software engineering, since all of these processes, be them technical, managerial or quality make up the backbone of software organisations. The maturity of these software organisations can be measured based on their process orientation

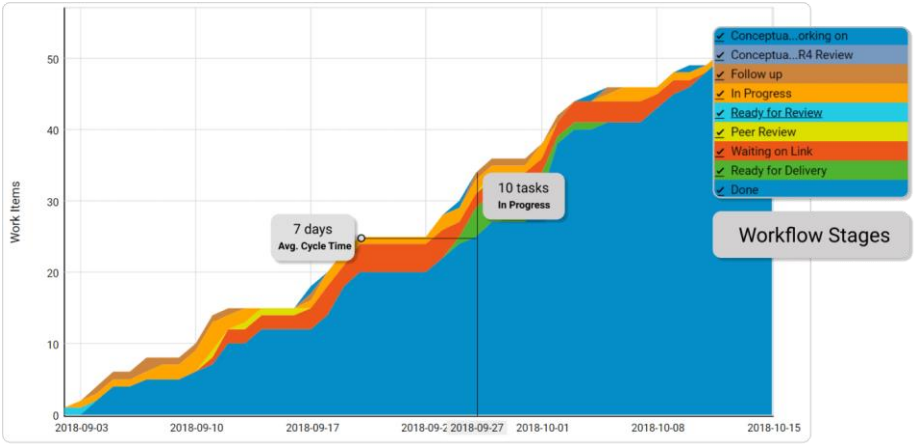
and correlation of processes to a reference model (Grambow & Oberhauser, 2013). For this reason, a manager needs a sufficient amount of measurable data to be able to analyse his employee's performance and thus become more productive.

Measurable Data

There are numerous different ways in which one can measure productivity, but the million-dollar question is, 'which one is best?'. "Industrial metrics activity is invariably based around metrics that have been around for nearly 30 years (notably *Lines of Code* or similar size counts, and defects counts). While such metrics can be considered as massively successful given their popularity, their limitations are well known, and misapplications are still common. The major problem is in using such metrics in isolation" (Fenton and Martin, 1999). Essentially, every person is different, and they find themselves most productive in different ways to others, meaning choosing just one way of measuring a software engineers' productivity could lead to inaccurate results and may affect the engineer in a negative way. I will analyse a number of methods to try and explain the positives and negatives of each type of measurable data, but there is no silver bullet when it comes to measuring the productivity of software engineers.

Agile Metrics

Lead time and cycle time are examples of agile metrics, and they are used to improve the process of software development. The time taken for an idea to become developed fully and delivered as a software is what determines lead time. It is a piece of data which is used for companies to see how responsive they are to customers. Companies will try to keep lead time as short as possible because a short project will please customers. As well as this, it will cost them less, and time and money are very important when it comes to measuring the software engineering process. A flaw involved with lead time is that teams may sometimes sacrifice quality for speed in order to get a project finished, and this negatively impacts the resulting project. The distinction between lead time and cycle time is said to be that lead time is a 'users' point of view', whereas cycle time is a 'developers' point of view'



(Agile Alliance, 2015). A cumulative flow diagram can be used to measure average cycle time and average lead time, as shown in the image.

Code Churn

Code churn represents the amount of code that was modified, added or deleted in a certain period of time. It is typically measured in Lines of Code (LOC). Churn spikes can be an indicator that an issue may require attention, as it shows that a developer has made limited progress on a feature although they have spent an amount of time on it that is greater than usual. It can alert a manager that one of their developers is struggling more than their fellow employees if their code churn is particularly high. As well as this, it can identify a perfectionist spending too much time on a certain aspect of a project when this time could be more useful elsewhere. Team issues can also be identified by code churn which is useful, as it may mean that they havent been given enough details about the project they are doing together.

Source lines of code

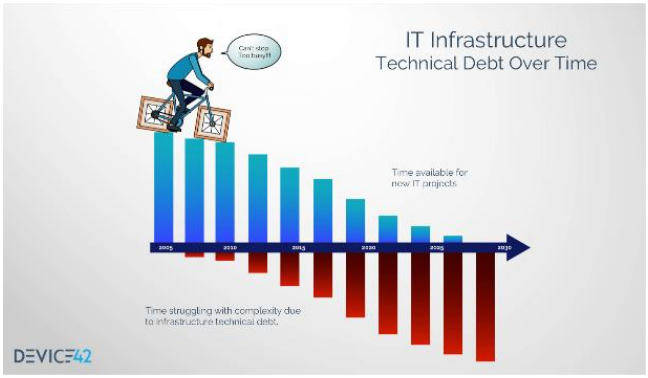
Source lines of code (SLOC) is a simple measure for the size of a program. It is determined by counting the number of lines of code in a program. It also helps to estimate the programming productivity once the software is produced. It involves two main measurements, physical and logical. Physical counts all of the lines of code including comments, whereas logical only measure the number of 'statements', meaning physical is much easier to calculate. SLOC isn't a good measure for effectiveness of a project, although it can be good for measuring effort. That isn't very useful though, as it can change depending on language used as well as the fact that a good software developer may design much more productive code in less lines. SLOC encourages longer code which can just become messy and confusing, and we know that quantity does not mean quality. At the same time, short code that is illegible is also useless. There is a fine line between short code that is only legible to the creator, and multiple lines of repetitive code which could've been done with half as many lines. This further shows that SLOC is a poor measure of data.

Number of Commits

The number of commits represents the amount of times a developer contributes to their code. It may not be the most accurate measure, as it can vary greatly between developers. It gives us a good idea of how active a developer is, but it doesn't necessarily tell us the difference in capabilities between developers. It can be a useful measure when managers may want to distinguish between developers when they know that these developers have similar skill levels.

Technical debt

Technical debt is essentially the opportunity cost of solving a problem. The extra effort that it takes to add new features or fix the problem is the technical debt. There will always be some element of technical debt in software engineering projects, but it should be kept to a minimum in order to maximise productivity. (Fowler, M., 2019)



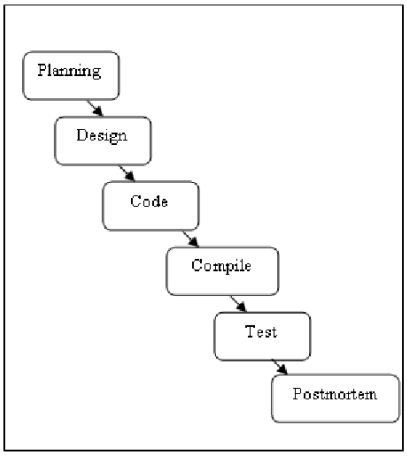
Computational Platforms available

Tools are a vital aspect of the process of measuring software engineering. After management have chosen which metrics they are going to use to collect the relevant data, they must then decide where they will compute it to obtain the results. The field of computing measurable data has undergone major developments in recent times, with many companies offering ways to measure and assess software metrics.

Personal Software Process

The Personal Software Process (PSP) is a process for software development that was designed to guide engineers to understand their own performance, therefore reducing defects in their software and increasing the accuracy of their estimations of the development duration. The idea behind it is that instead of relying on tests, developers should try and write good quality code from the start.

The PSP takes into consideration that all developers are different, meaning they work in their own ways and therefore must analyse their own performance in their own way. If there was a universal format, it wouldn't work as the developers need to personalise and plan their own assessment. If engineers are responsible for their own mistakes, they will produce better quality final products. The PSP also emphasises the importance of finding mistakes earlier, making it cheaper to fix them.



The PSP is broken up into multiple phases to allow developers to personalise it as much as they can. It begins with developers planning the project and writing a Design document, and this

design must be reviewed before advancing. The next step is the coding, which also needs to be reviewed before getting to the testing. Lastly, a post-mortem must be done to describe how the project went, meaning the developer must constantly be keeping track of their own performance including things such as missed deadlines or defects in code. The size of the project must also be recorded using things such as the lines of code.

The PSP process provides software engineering students with empirical support for improving estimation and quality assurance, according to case study research. Despite this, many software engineers tend to stop using the process once they aren't required to do so anymore. Research suggests that this is due to two main problems:

- PSP style metrics collection and analysis has high overheads.
- The process requires users to switch context between product development and process recording. (Johnson, P., 2003)

These factors can make users feel like they are almost wasting time. A process which automates data collection and analysis would therefore be more useful, which is the case for Hackystat which I will analyse now.

Hackystat

Although the PSP is widely praised, it comes with some issues. Firstly, the developers are put under pressure to manipulate numbers to satisfy management. Secondly, there is a big manual overhead in collecting much of the data required to complete the analysis using the PSP. To avoid these problems, tools such as Hackystat were made to collect all of the data for you using automated data collection, data tracking and data analysis.

Hackystat can be used for researchers, engineers and more. It allows user to attach software 'sensors' to their development tools which unobtrusively collect and send 'raw' data about development to a web service called the Hackystat SensorBase which can be alter queried by other web services to form higher level abstractions of this raw data (Hackystat, 2018). The objective of Hackystat is to facilitate 'collective intelligence' in software development, giving us an insight and knowledge about the raw data it collects.

Hackystat's detailed analysis allows it to collect data consistently throughout the day, checking whether the developer is calling methods or testing as the go for example. This means it can check if the developers are doing Test Driven Design, which relies on short work cycles, writing tests and the restructuring the code to pass all the tests, repeating this process until the code passes all test cases. "The tool focuses on individual data collection and individual use of such data. This structure allows a high level of privacy, very important inside industries, but limiting advantages for the workgroup and for the entire company" (Sillitii, Janes, Succi & Vernazza, 2003).

Hackystat has been integrated into many IDEs because of how popular they are, but this has raised ethical concerns due to the levels of personal data being collected. This will be further examined later in the report.

Code Climate

Founded in 2011, Code Climate is a privately held software company. Code Climate is an automated code review tool that analyses code on several different levels before recommending what can be improved in the program. Code Climate analyses huge amounts of code every day which is why it is the most popular platform currently available to developers. In partnership with Github, it offers a service which shows a user their code coverage, technical debt and progress report. It enables an automated analysis of multiple languages including Python, PHP and JavaScript. As well as this, it is renowned for its reliability, stability and test coverage, while also allowing team sharing meaning it can provide direct access to an entire development team. However, Code Climate is also quite expensive when compared to its rivals.

Codebeat

Codebeat is another popular tool used for software development due to the fact that its is simple, free and open source. It has grown significantly in recent time and it supports multiple programming languages including Python, Ruby, Java and more. Codebeat's tools have evolved recently, but it is still most popular for taking users feedback into consideration. Codebeat generates its own algorithms from scratch, with details that allow developers to focus on important aspects of code review such as architecture and business logic. Despite this, Codebeat still has some missing features, "The tool does not allow performing any security checks (perhaps a "simple" integration with Brakeman would do the job). It also does not support any open-source tools or linters (in fact, we are still using Hound) and does not support analysing CSS/SCSS" (Ozimek, L., 2017).

Algorithmic Approaches

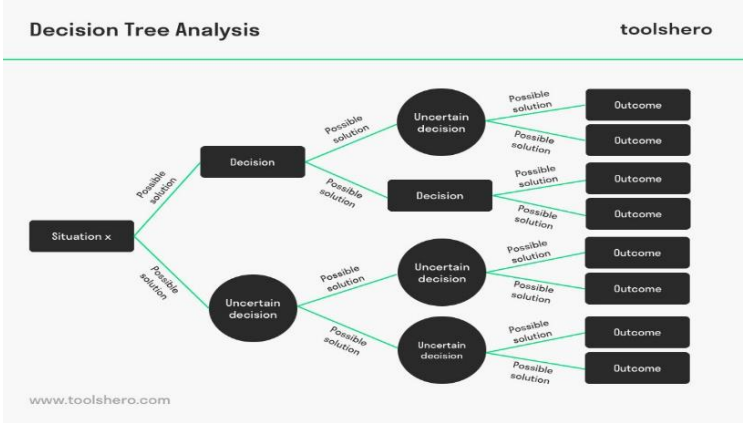
Accumulating data is essential when measuring software engineering, but this 'raw' data means nothing if we can't get a useful analysis of this data. This is when algorithmic models come in to play. Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention (Sas, 2019). This is the reason why Machine learning has increased in popularity as it is considered an unbiased technique. Two of

the most widely adopted machine learning methods are supervised learning and unsupervised learning, which I will examine.

Supervised machine learning

Supervised learning methods are the most popular type, and they assume a given structure within data, examples being Decision Tree Analysis or K Nearest Neighbours. They are trained using labelled examples, such as an input where the desired output is known. The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors (Sas, 2019). It is called supervised learning because it learns from the initial dataset, almost like a teacher supervising the process. The algorithm iteratively makes predictions on the training data and is corrected by the teacher. The learning is then finished when the algorithm achieves an acceptable level of performance. (Brown Lee, J., 2016)

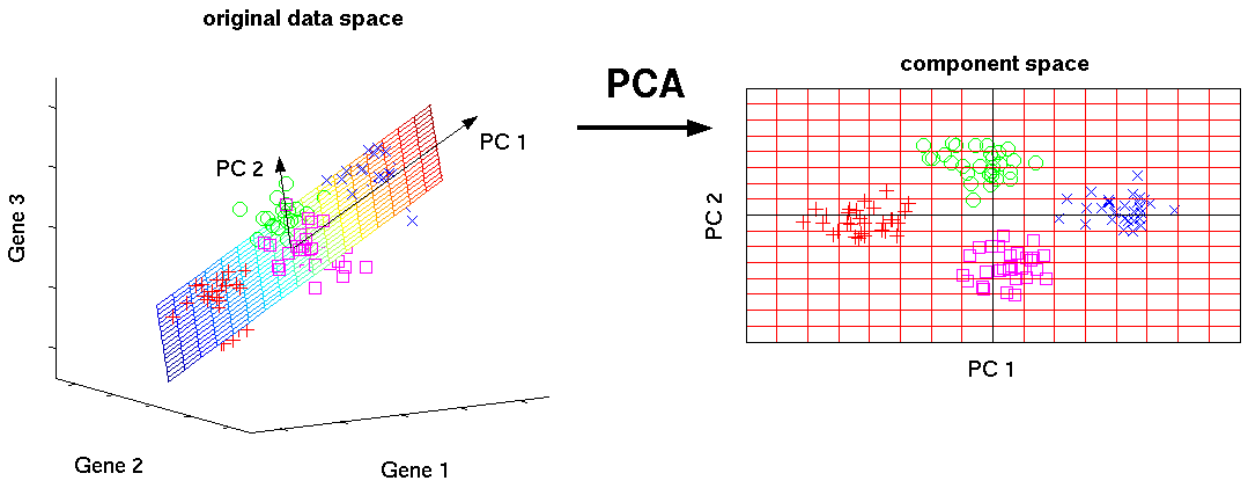
Decision tree analysis is an example of supervised machine learning. It is a graphical representation of various alternative solutions that are available to solve a problem. A final answer is achieved by answering a number of questions after each other giving positive or negative answers (Mulder, P., 2017).



Unsupervised machine learning

Unsupervised machine learning is closely related to supervised machine learning, except in this case you only have input data and no corresponding output variable. This means that the algorithm must figure out what is being shown by itself. The goal is to explore the data and find some structure within, meaning there is no ‘correct answer’ and no way of correcting its output. Unsupervised learning works well on transactional data, for example it can identify segments of customers with similar attributes who can then be treated similarly in market campaigns. Popular techniques include Principal Component Analysis, nearest-neighbour mapping and k-means clustering (Sas, 2019).

Principal Component Analysis (PCA) is a data-reduction technique use to emphasize variation and bring out strong patterns in a dataset. It is often used to make data easy to explore and visualize (Explained Visually, 2019). PCA converts a set of correlated variables into a reduced set of uncorrelated variables.

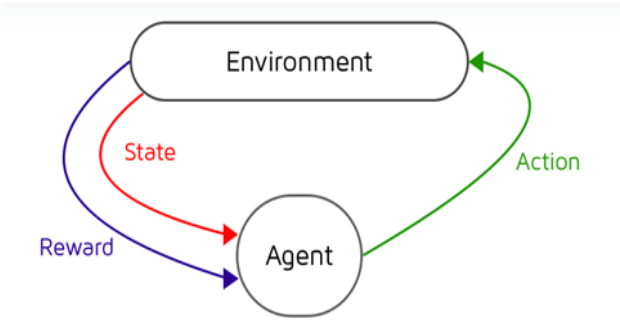


Reinforcement learning

Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience (GeeksforGeeks, 2018). It essentially adopts a trial and error approach, learning by feedback from its own actions.

This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do) (Sas, 2019). The learner must decide which actions return the most reward by testing them. The ideal behavior which maximises reward will occur if a reinforcement learning algorithm converges to the global optimum. The agent will reach the goal much faster by following a good policy, showing the main objective in reinforcement learning is to learn the best policy. A problem with this is that often, it can be expensive in terms of memory as more complex problems require more memory. An example of a reinforcement machine learning method is the Markov Decision Process.

The Markov Decision Process (MDP) is a framework used for making decisions in a stochastic environment. The algorithm is put in a state where it knows the actions it can take as well as the consequences involved, allowing it to make decisions and define its new state. If a bad decision is made, it learns and doesn't make this mistake again. This happens repeatedly until an optimal solution is found.



Ethics

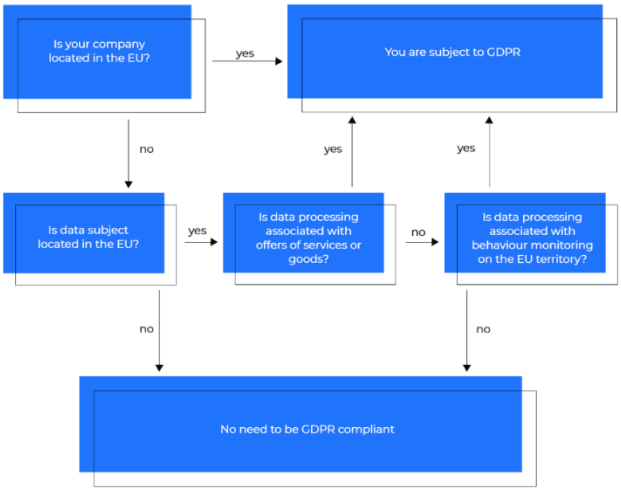
Throughout this report, it is clear to see that measuring software engineering is an essential component towards being successful in business. In order to obtain data that is useful and worth gathering, you must deal with privacy and overhead concerns. Problems occur when the data measuring may get too personal or intrusive, making employees feel uncomfortable. More importantly, there are also laws and regulations which companies must follow regarding data collection and privacy.

The main regulation when it comes to data protection is the General Data Protection Regulation (GDPR), which came into effect last year. GDPR addresses data protection by design and by default as a general obligation for data processors and controllers. From a software engineer's perspective, it is really challenging to distill the relevant information and concrete action points from this new regulation (Software Improvement Group Inc., 2017). Organizations are becoming more and more cautious regarding data protection due to this. The GDPR emphasizes the importance of privacy and it imposes strict laws on organizations that hold personal data.

For EU residents, the GDPR requires that businesses get consent to collect any information when people use their online stores or new portals. The regulation reshapes the way organizations approach data privacy and guides them to build GDPR-compliant software. GDPR expands on previous European Union data laws (SteelKixi Inc., 2018).

Under the GDPR, individuals have a set of rights, such as the right to consent, access and erasure, as well as many others. If businesses fail to meet the requirements stated in the GDPR, authorities can halt your personal data processing activities and fine you heavily. These fines can range from 10 to 20 million euro or else 2% to 4% of a company's annual global turnover. Individuals also have the right to receive compensation for any material or non-material damage (SteelKixi Inc., 2018). This one regulation means companies must be very careful when measuring software engineering.

"Data sovereignty is the concept that information which has been converted and stored in binary digital form is subject to the laws of the country in which it is located." (WhatIs, 2013). Many concerns regarding data sovereignty relate back to the topic of privacy regulations. Cloud computing services are being adopted more and more, meaning traditional geopolitical barriers are being broken down. This means that if you are working in a multinational company that is based in numerous countries, it can be hard to clarify which laws apply meaning there could be conflict regarding your information.



A well-known example of a lack of ethics regarding technology is in the Cambridge Analytica case. Cambridge Analytica is a political data-analysis firm that worked on the 2016 Trump campaign. The issue was about how they got a hold of a massive dataset of the American electorate, allegedly keeping data on 50 million Facebook users without their permission. The trump 2016 digital campaign relied heavily on ad targeting, which may have utilized Cambridge Analytica's Facebook data. The full story of where the blame lies is still emerging, but it still makes for a shocking story about data protection and our privacy when it comes to technology (Thompson & Vogelstein, 2018).

In my opinion, there is no definitive answer regarding ethics in measuring software engineering. Companies do need to monitor their staff, otherwise the staff could be incredibly unproductive without management even knowing. This could prove to cause major problems for a company. At the same time, it isn't reasonable to track every single thing an employee does, both legally and ethically. Each software engineer is different too, meaning they must be tracked in different ways to get a true reflection of their productivity. To conclude, I think the most important thing is the skill level as well as the working environment of the engineer. A skilled employee in a workplace with a good atmosphere should be trusted to maximise their output, meaning minimal data collection is required.

Conclusion

In summary, this report has examined the ways in which software engineering can be measured and assessed, primarily through measurable data, computational platforms, algorithmic approaches and ethical concerns. I have learned that measuring and assessing software engineering is certainly more confusing than one might imagine. After examining the multiple different ways in which it can be done, it is easy to see that there are many possible answers, but no definitive answer about the best way to measure software engineering, and there may never be one. There is no 'one size fits all' technique to the process, so it is just a matter of combining different metrics to find out which combination works best. Technology and software engineering are constantly developing year on year, and I imagine it will continue to do so for the foreseeable future. For this reason, I believe the measurement of these metrics will continue to play an important role in the world of software engineering.

References

- Agile Alliance, 2015. *What is Lead Time in Software Engineering?* [Online] Retrieved from https://www.agilealliance.org/glossary/lead-time/ on 25/10/19.

- Altieri, M., 2019. Infrastructure Technical Debt. [Online] Retrieved on https://www.device42.com/blog/2019/05/infrastructure-technical-debt/ on 25/10/19.

- Brooks, F. P., 1987. *No Silver Bullet - Essence and Accident in Software Engineering.* University of North Carolina: s.n.

- Brown Lee, J., 2016. Supervised and Unsupervised Machine Learning Algorithms. [Online] Retrieved from https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ on 26/10/19.

- DeepAI, 2019. *Markov Decision Process.* [Online] Retrieved from https://deepai.org/machine-learning-glossary-and-terms/markov-decision-process on 26/10/19.

- Explained Visually, 2019. *Principal Component Analysis explained visually.* [Online] Retrieved from http://setosa.io/ev/principal-component-analysis/ on 26/10/19.

- Fenton, N. E., and Martin, N., 1999. "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

- Fowler, M., 2019. TechnicalDebt. [Online] Retrieved from https://martinfowler.com/bliki/TechnicalDebt.html on 26/10/19.

- GeeksforGeeks, 2018. *Reinforcement learning.* [Online] Retrieved from https://www.geeksforgeeks.org/what-is-reinforcement-learning/ on 2/11/19.

- Grambow & Oberhauser, 2013. Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology.

- Hackernoon, 2018. *A Layman's Introduction to Principal Components.* [Online] Retrieved from https://hackernoon.com/a-laymans-introduction-to-principal-components-2fca55c19fa0 on 2/11/19.

- Kanbanize, 2019. *6 Agile Metrics to Boost Your Project Management*. [Online] Retrieved from https://kanbanize.com/agile/project-management/agile-metrics/ on 25/10/19.

- Mulder, P., 2017. Decision Tree Analysis, a great decision making tool. [Online] Retrieved from https://www.toolshero.com/decision-making/decision-tree-analysis/ on 2/11/19.

- Ozimek, L,. 2017. Comparison of automated code review tools: Codebeat, Codacy, Code Climate and Scrutinizer. [Online] Retrieved from https://www.itproportal.com/features/comparison-of-automated-code-review-tools-codebeat-codacy-code-climate-and-scrutinizer/ on 7/11/19.

- P.M. Johnson et al., "Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined," Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.

- Sas, 2019. *A guide to machine learning algorithms and their applications.* [Online] Retrieved from https://www.sas.com/en_ie/insights/analytics/machine-learning.html on 2/11/19.

- Sas, 2019. *Machine Learning: What it is and why it matters.* [Online] Retrieved from https://www.sas.com/en_ie/insights/analytics/machine-learning.html#machine-learning-workings on 2/11/19.

- Silliti, Janes, Succi and Vernazza, 2003. Collecting, integrating, and analyzing software and personal software process data.

- Software Improvement Group, 2017. *Practical engineering issues of the GDPR*. [Online] Retrieved from https://www.softwareimprovementgroup.com/blog/practical-engineering-issues-gdpr/ on 7/11/19.

- SteelKixi Inc., 2018. *Creating GDPR-Compliant Software.* [Online] Retrieved from https://medium.com/swlh/creating-gdpr-compliant-software-2866db9d08f6 on 7/11/19.

- Thompson and Vogelstein, 2018. Facebook Struggles to Respond to the Cambridge Analytica Scandal. [Online] Retrieved from https://www.wired.com/amp-stories/cambridge-analytica-explainer/ on 7/11/19.

- WhatIs.com, 2019. *What is software engineering?* [Online] Retrieved from https://whatis.techtarget.com/definition/software-engineering on 25/10/19.

- WhatsIs, 2013. *What is data sovereignty?* [Online] Retrieved from https://whatis.techtarget.com/definition/data-sovereignty on 25/10/19.