MEASURING SOFTWARE ENGINEERING

CSU33012 - Report

Laura Stack

17325734

1 ABSTRACT

This report aims to offer a wide-ranging overview of the different ways in which software engineering processes can be measured and assessed. I will discuss this though the following topics: measurable data, computational platforms available, algorithmic approaches available and the ethical concerns surrounding this type of analysis.

This report will be compiled using an array of academic material as well as the content discussed and referenced in the lectures of CSU33012.

2 INTRODUCTION

Software Engineering is 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.*" (Techopedia) It is the application of engineering principles to software development. The products of software engineering form part of our daily lives. It determines how we interact with technology on a daily, if not hourly, basis. Due to the rapid growth of the indsutry, the need arose for a method of measuring and assessing the process of software engineering.

How do we approach the task of measuring this discipline? Furthermore, how can we rank a software engineer's performance? How can we rate the product of the software engineering process? How can we assess this process itself? In this report, I aim to answer these questions under the following main headings:

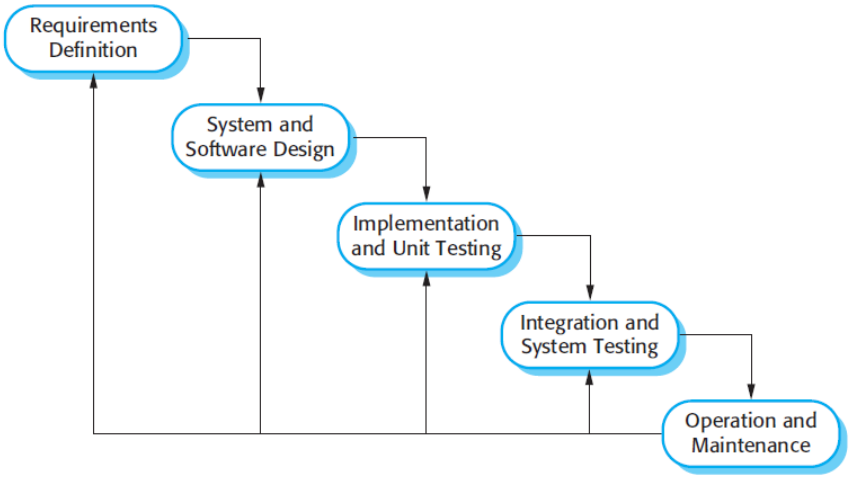
1. Measurable Data
2. Platforms Available
3. Algorithmic Approaches
4. Ethics of Analysis

It is essential to note, however, that the software engineering process can vary from project to project. We can never definitively determine the "best" technique, the "best" style of coding, or the traits of the "best" individual software engineer. Various quantifiable and non-quantifiable factors are at play. For this reason, we must find general methods of measuring the process, that are applicable to all software engineering techniques.

In order to investigate how software engineering is measured and assessed, we must first look at the actual process itself.

3 SOFTWARE ENGINEERING

It is important to understand what the process of software engineering involves before delving into its measurement and assessment. The process involves a series of tasks that lead to the development of software, and the below steps must be carried out regardless of the project at hand.



This sequence of activities is commonly referred to as the Software Development Life Cycle (SLDC). Each of these activities can be implemented in a variety of SLDC models, the most popular tending to be waterfall, agile and iterative. (Osetskyi, 2017) It cannot be said which process is better than the others, but rather the suitability of a process will depend on the type of project. In the *waterfall* model, the development process is sequential and pre-defined. This simplifies project management, but software is not ready until the last stage is completed. The *agile* model allows the customer to view the result and approve/disapprove the software after each stage of development. This improves software by integrating correction and functional requirements into the development process but poses the risk of new requirements conflicting with the existing architecture. *Iterative* models develop different components of the system in cycles, adding each component to components developed previously. The process is repetitive, allowing new versions of the product to be made in every cycle. This allows for easier control of risks but can be more difficult to manage than other processes. As afore mentioned, the core steps above must be carried out regardless of the fact that processes can differ slightly.

The software engineering process can be lengthy and complex, requiring attention from the engineer at every stage of development. How can we measure the process?

4 MEASURABLE DATA

Data is an important part of quality improvement in any industry. Vast amounts of data can be collected throughout the software engineering process, but the challenge is to ensure that this data can provide useful information for process improvement, with specific business goals in mind.

Measuring software engineering is different to most other disciplines of work. Productivity or value cannot simply be measured by the number of units produced or hours worked. Software metrics are defined as tools and analysis used to measure performance in software engineering. *(Fenton & Martin, 1999)* The term refers to numbers that characterise properties of software code as well as models that help predict software quality and resource requirements. Software metrics play a key role in improving the way we monitor, predict and improve various attributes of the software engineering process.

A. WHY DO WE MEASURE THE SOFTWARE ENGINEERING PROCESS?

Ultimately, the purpose of software engineering is to complete a task to improve or create a process and create and/or add value for a user. Like in all aspects of business, it is crucial to measure and monitor performance using metrics with the aim of increasing productivity and reducing waste within the organisation. The following question demonstrates the importance of measurement: would it be better to have 3 expensive, extremely talented, "10x" software engineers, or 10 less talented engineers, but less costly?

B. IN WHAT WAYS CAN WE MEASURE THE SOFTWARE ENGINEERING PROCESS?

1. Lines of Code/Source Lines of Code (LOC/SLOC)

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight"
– Bill Gates

Weiss (2002/03) noted that, at the time, SLOC was a popular metric in cost estimation for the development of a system. Despite several proposed cost estimation models based on SLOC, and their widespread use, Weiss highlighted several flaws with the approach. Measuring the number of lines of code written had long been the standard for the examination of an engineering process. However, in the same sense that taking the amount of bricks a bricklayer lays as a measure of quality, the number of lines of code gives little or no indication of code quality, or indeed its impact. This could entice engineers to add unnecessary lines of code that have no value, simply to add to the apparent quality of the code.

2. Number of commits

Measuring the performance, quality, and efficiency of a project by simply counting the number of commits made by developers is undoubtedly a flawed approach. Frequency of commits has no correlation to the quality or size of a project, nor the level of completion. Albeit, it can be used as a measurement of consistency of development where common standards are adhered to. Commits can remove code previously added to the repository which places a crucial limitation on this metric.

3. Code Coverage

A crucial metric which is used as a measure of completion of a project is the code coverage of tests written by developers. Code coverage is a percentage of the number of lines executed in the program as a percentage of the entire program. Coverage criteria includes testing whether each Boolean scenario has been tested, if every statement in the code has been tested, etc. Incomplete code coverage reduces the quality assurance of a project.

4. Observation/Keystroke tracking

This interesting metric arises from the question of whether a correlation exists between an individual's speed of work, and the reliability of their code. An individual's activities can be tracked by recording every activity they carry out on their computer down to each click, with this information collated to analyse the above question. I have read anecdotes online about colleagues who worked at a noticeably slow pace through the engineering process but produced consistently high quality, bug-free code. My intuition would say that at any point in time the value of one's work should outweigh that of a less talented developer, but of course, there may be exceptions to this.

5. Code Churn

Code churn measures the rate at which code evolves, i.e. measuring how code changes over the development life cycle. (CodeScene, n.d.) As a metric, it can be simplified to a moving average of the number of lines added against the number of lines removed. It is most useful in project management in identifying delivery risks to customers. Code churn can be positive in cases if a developer has conducted poor programming practice, and another developer has found a more efficient way of designing the system.

6. Self-Quantification

An increasingly prevalent trend, even implemented in some part in Google, is where employees measure their own performance and define their own "key performance indicators". This measurement has an obvious limitation in that it is subjective but can also be used as an indirect measure of staff satisfaction.

7. Lead Time

Lead time is defined as the time elapsed between the identification of a requirement and its fulfilment. It measures the total time it takes for work to move through the development process from the moment the project is instigated to the time it is delivered. Lead time can be used to estimate future delivery expectations, assess developers' adherence to deadlines as well as levels of customer satisfaction.

8. Cycle Time

Cycle time is similar to lead time, with the difference being that for cycle time the clock starts when the software engineer actually begins working on the project rather than when the request is made. Other time metrics used include mean time between failures (MTBF), mean time to recover (MTTR) and mean time to repair, which is the time between the discovery of a security breach and a deployed, working solution. (Lowe, 2018)

9. Technical Debt

Technical debt is a concept in software engineering that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution. (Techopedia) In the long term, technical debt can be cortical to a business as it may struggle to keep up with competitors. A good software engineer pays back technical debt promptly with a rewrite. Tools to measure technical debt as a metric focus primarily on code quality and the overall system complexity.

10. Bug Fixing

Management should not necessarily be interested in how many bugs are arising in the developer's code, but rather the severity of these bugs and how they deal with these. The time spent on bug fixes could be used as a metric, but the time spent in relation to bug severity/quantity will vary from developer to developer.

C. LIMITATIONS OF METRICS FOR PREDICTING SOFTWARE QUALITY

It can be very easy for managers to look at a metric and jump to a conclusion. However, we cannot use metrics to assume direct causes. Communication with the team and investigation into any alarming metrics is essential, before deciding whether the metric is a cause for concern, and metrics must be considered in context. An example of this is when we look at the absolute number of defects versus the defect density, which is the number of defects per new or altered line of code. Intuitively, longer code is expected to have more defects. The number of post-delivery faults in our software is our priority but hypothesis tests carried out on the metrics detailed in academia indicate that complexity indicators are poor predictors on fault density post-release. (Fenton & Martin, 1999)

D. WHAT CANNOT BE QUANTIFIABLY MEASURED?

While it is very important to monitor much of these metrics it must be noted that it is essential for managers to also assess the non-measurable observations of software engineers. Developers' lives outside of the workplace can have a considerable impact on their performance at work, and while their metrics could lead one to believe that they are lacking in talent, there may be personal issues at play. It has been claimed that happy employees may have up to 12% more productivity than their unhappy colleagues. (Oswald, et al., 2009)

5 PLATFORMS AVAILABLE

Once data has been gathered, it must then be interpreted to analyse the performance of a software engineering process. Various tools and programs which collect software engineering metrics exist for this purpose. Efficiency of these tools can be key for a business, as rapid analysis facilitates faster decision making. Given the rapid growth of the data analytics industry, there are a huge number of platforms available to analyse data, and their usefulness for software engineers in understanding and improving their software engineering process cannot be understated.

I. PERSONAL SOFTWARE PROCESS (PSP)

The concept of PSP was first introduced in the 1990's by Watts Humphrey. (2000) He believed that a structured development process as well as disciplined tracking of progress would allow the engineer in question to streamline their process, in a more efficient manner. PSP provides a framework for self-evaluation and best practice on software engineering. PSP is mainly focused on manual input, where the engineer fills out various forms, answering based on their personal judgement. In a structure similar to Maslow's Hierarchy of Needs (1943), once a certain level of utility, or in this sense increased performance has been achieved, engineers must strive to improve their processes once again. A clear limitation of PSP is that it is susceptible to human error, and this inefficiency triggered the development of the Leap toolkit. (Johnson, 2013)

II. LEAP TOOLKIT

The Leap toolkit attempts to address the problems encountered in the PSP by automating and normalising data analysis. Although the toolkit still requires manual developer input, it then automates subsequent PSP analysis and provides additional tools, such as regression analysis. Leap data is also portable, creating a repository of personal process data that developers can keep with them as they move from different projects and organisations. To employ a metaphor: "*The Leap toolkit replaces the PSP candle with a campfire".* After the development of the Leap toolkit and its longevity of use industry-wide, it was accepted that user input would be required in some form and human error could not be fully eradicated. With the goal of carrying out unbiased analysis, a way around manual input was required which inspired the development of more automated tools, namely a decade-long research project called Hackystat. (Johnson, 2013)

III. HACKYSTAT

Hackystat is an open source framework for collection, analysis, visualisation and interpretation of the software engineering process. (Hackystat, n.d.) The platform reverses the conventional approach of defining high level goals and consequently figuring out what data collection analysis is necessary to achieve them; indeed, it does the very opposite.

Hackystat collects data from both client and server-side data, allowing for more comprehensive data analysis, and collects data unobtrusively, with users unaware of when data is being collected. This, however, can lead to conflict with developers who may be uncomfortable with the idea of others having an accessible platform into their work. (Johnson, 2013) It is suitable for any institutions who use GitHub or another git derivative, as they can easily integrate the repositories with Hackystat and gain valuable insight.

IV. HUMANYZE

If measuring the software engineering process alone doesn't suffice, Humanyze could prove an option for companies. Humanyze provide companies with sensors to track everything their employees do throughout the day. Their credit-card sized badges contain a microphone to track speech and tone of conversation, an accelerometer to track movement, infrared to detect with whom one is speaking and Bluetooth to track location. (Goyal, 2018) Humanyze has already established itself, working with over 50 Fortune 500 companies across a diverse range of industry spanning pharma, energy and technology. Employees may feel uneasy with the level of data collected but Humanyze stress that the system is built to anonymise data, analysing the organisation's productivity as a whole rather than specific individuals. To speak in terms of Big Brother – he is always watching but watching data points rather than people.

V. FURTHER PLATFORMS

Many more platforms exist for managers to analyse software engineering data and metrics. These include Codacy, GitPrime, Code Climate, Waydev, Velocity, Squore, Sonar etc. The vast array of platforms available is indicative of the ever-rising popularity and sheer importance of measuring software engineering.

6 ALGORITHMIC APPROACHES

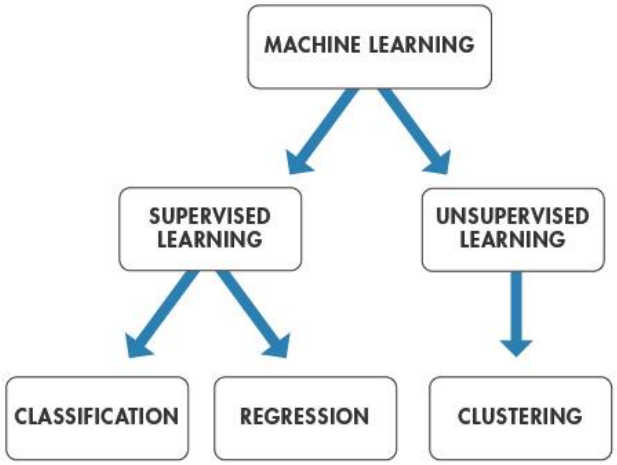
It is clear that data analytics is an essential part of measuring the software engineering process. Manual analysis of data has long been replaced by algorithms that automate the process in a more efficient and informative way, and the computational platforms mentioned above have a number of algorithms running in the background. Algorithms can produce a large amount of information in a short frame of time, as well as identifying relevant structures, patterns and trends. Large tech companies like Google have proprietary

algorithms in use to analyse employee performance. A huge number of techniques are available, but given the increasing prominence of machine learning (ML), I will mainly focus on the algorithms driving this technique. ML has greatly sped up the decision making process by enabling computers to analyse information and make decisions themselves.

A. TYPES OF ML ALGORITHMS

ML algorithms can be divided into the following main methods (Brownlee, 2019):

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



I. SUPERVISED LEARNING

Supervised learning is where the algorithm generates a function that maps inputs to desired outputs, i.e. , i.e. $Y = f(x)$ where y is the output variable and x is the input variable. It is "supervised" because the data scientist acts as a guide to teach the algorithm what conclusions it should be drawing from the data. To "teach" the data, a "training dataset" is used where there are a number of inputs and corresponding outputs. The algorithm iteratively makes predictions based on the data and corrects any outlying predictions. The process is then repeated until a satisfactory level of accuracy is reached. Types of supervised learning algorithms are as follows:

A. K-NEAREST NEIGHBOURS

K-Nearest Neighbours is a non-parametric method of classification. The algorithm makes no assumptions about the spread of data within each class. The algorithm is implemented by receiving a point and classifying it using the data we already have, in order to make a prediction. The number of K-Nearest Neighbours that will help classify the point is decided, e.g. k = 3, and the machine then selects the variable which dominates the choice of the 3 nearest observations to the new point. (Houlding, 2019)

B. LINEAR DISCRIMINANT ANALYSIS

Linear discriminant analysis is a classification technique that allows us to place multivariate data into classification groups. This algorithm defines a classification decision bound, say 0.3,

and classifying multiple instances of this data by the boundary. The method is widely used with ML and AI. (Houlding, 2019)

II. UNSUPERVISED LEARNING

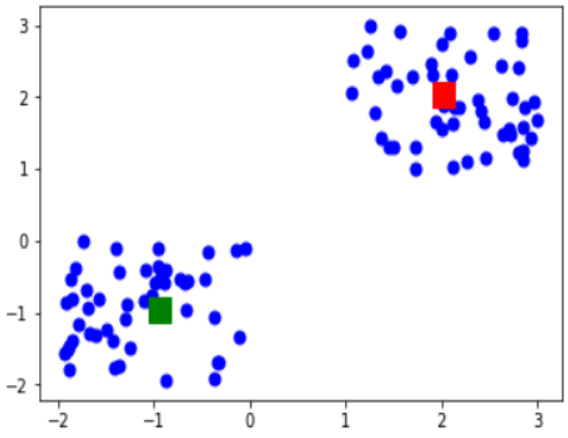
Unsupervised learning is regarded by many as being more synonymous with machine learning, as the aim is to have the computer learn how to do something without us telling it how. The goal is for the algorithm to learn to identify complex processes and patterns without human guidance. Input data is provided, but output is unknown. Types of unsupervised learning algorithms are as follows:

A. PRINCIPAL COMPONENT ANALYSIS

This is a method used to compress a dataset with a lot of dimensions into something that captures the essence of the original data. The goal is to identify which variables explain most of the variance in the data. This method can help to identify relationships between variables and possible correlation. For example, if a developer has a high level of code churning, they may be expected to have a higher lead time. (Houlding, 2019)

B. K-MEANS CLUSTERING

Clustering algorithms are very useful in data analytics as any dataset can be divided into a set of clusters according to their different characteristics, hence identifying a group structure in the dataset. For example, if there is a group of fast-working and slow-working engineers we can examine the characteristics of such clusters. This algorithm is an iterative method that divides the data into k distinct groups so that observations with similar characteristics are in the same group, while observations between groups are different. (Houlding, 2019) Other clustering methods include mean-shift clustering and hierarchical clustering.



III. REINFORCEMENT LEARNING

This type of learning trains the algorithm to make a sequence of decisions by interacting with their environment, in a form of dynamic programming. The algorithm takes on a penalty or a reward for each act. The algorithm aims to maximise long-term reward and minimise long-term punishment. Reinforcement learning algorithms are centred around decision making. This use of this type of ML is becoming increasingly useful with the development of driverless cars. In this case, the car should learn from previous mistakes and not make the same mistake in future.

However, in relation to measuring the performance of a software engineer, this algorithm can have a limitation. As time goes on and more inputs are fed to the system, the accuracy of decision making will improve, but could improve so much that it may end up lacking in human common sense. Human error presents a drawback, but human intuition should not be completely disregarded, especially in situations where "soft" data is being considered in social interaction. In my opinion the balance between human error and common sense has not yet been reached by an algorithm, but the growth of ML has opened the door to a range of new possibilities.

7 ETHICS

As discussed, the measurement and assessment of software engineering is greatly advanced through use of data analytics and machine learning, but the question now left to answer is: Is all of this ethical? I think that this is a grey area at the moment. In other disciplines such as law, medicine, finance, ethics are clear and regulatory frameworks are in place. However, the distinction between right and wrong is not clear in the field of software engineering when we consider whether this level of data procurement is ethical, let alone analysing it. The algorithmic techniques outlined previously simply cannot operate without data. The means of data collection is often where privacy is breached, and ethical lines are blurred. A recent example of this is the well-documented Facebook and Cambridge Analytica Scandal, which caused major controversy in the field of data analytics. How far should firms go to collect our data? How much data do we feel comfortable giving these companies? What responsibilities do these companies have in order to keep our information safe?

A. DATA COLLECTION

The way in which data is collected can raise a number of ethical concerns, especially when data is collected unobtrusively. In the case of Hackystat, developers can feel uncomfortable if data is being collected without their knowledge. Also, as previously discussed, Humanyze technology tracks employees' every movement and interaction, analysing it in real-time. This can hamper productivity and morale, and consequently quality of work, as developers will not work freely. Personally, my **gut reaction** to the idea of having my every move monitored and analysed is one of stress. In that situation, I imagine I would feel added pressure and scrutiny that would prevent me from working to my full potential. My sentiments are in line with Sommerville's (2011) claims that confidentiality, honesty and integrity are key to the successful performance of a software engineer. There is an obvious trade-off between easily obtaining analytics, keeping in line with ethics, and richer analytics secured with privacy concerns. Management need to strike a balance between gathering useful data while respecting the privacy of the employee, but the reduced value of ethical data drives the temptation to act otherwise.

B. TRANSPARENCY

Transparency in what data is being collected, how it is being used and whether or not it is being passed on to other parties is a key factor. The metrics that management are analysing and their purpose for doing so should always be communicated to the software engineers. This can break down the negative attitude among developers regarding data collection and demonstrate how useful it can be to improve their performance and add value to the organisation. It could be a possible idea to give developers a level of control over which data they wish to have collected, after explaining the positive impact that performance analysis can have. Developers are likely to be somewhat more comfortable about being monitored if they have an element of control. If software engineers are not content, it can severely damage a company's reputation. There are more platforms than ever for employees to air their grievances, such as Glassdoor or even word of mouth. The opposite is also true, that satisfied employees can help to attract top talent, further emphasising the importance of transparency and clear lines of communications between management and developers.

C. REGULATION

In May 2018, the EU released the General Data Protection Regulation (GDPR) to replace the previous Data Protection Act (DPA). Under this set of regulations, data subjects must actively agree to the use of their data, and consent can be withdrawn at any time. Deletion of data can be requested at any time. These regulations apply to all companies who process data of individuals/employees living in the EU, regardless of where the company itself is incorporated. Therefore, the large technology conglomerates including Facebook, Google and Amazon must adhere to these rules. Under the legislation, a company can be fined 4% of their income or €20 million, with such a large penalty enforcing compliance, as well as the threat of reputational damage. (Citizens Information, 2018) GDPR is undoubtedly a step in the right direction but a lack of transparency and unethical actions remain in the field of data analytics.

D. SOVEREIGNTY / OWNERSHIP

Data sovereignty and data ownership are grey areas that may raise ethical concerns. Data sovereignty is the idea that data are subject to the laws and governance structures within the nation it is collected. It is important for organisations to have clarity about where the data is located and what laws it is subject to, and like with GDPR, where the customers whose data they control live. Furthermore, it is not always clear to customers when they transfer ownership of their data. If you post on Instagram, do you transfer ownership of your image? If you commit to GitHub, do you retain ownership of your code? Data collected by software engineers is widely considered to be owned by the company, but is it owned by the company as an entity? Who controls it? This is not always clear.

8 CONCLUSION

To conclude, in this report I have outlined the ways in which the software engineering process is measured and assessed, as well as discussing the limitations, risks and responsibilities associated with this. Software engineering has come a long way from its humble beginnings, and it is clear that measuring the performance of a software engineer can prove extremely beneficial. Management can track progress, evaluate productivity and quality, and use the data to aid managerial decisions. There are various competent platforms available for analysing software engineering metrics, and as long as these metrics are being measured correctly, they can add significant value to a project and indeed to the developer themselves. Of course, it must be noted that there are numerous soft skills required to make a good software engineer that cannot necessarily be quantified very easily. (Hackernoon, 2018) While it can be easy to focus on the metrics, a good software engineer must have skills of empathy, patience, open-mindedness, communication, creativity, time management, and many more. This should always be kept in mind when measuring and assessing the performance of a software engineer and the processes they implement.

BIBLIOGRAPHY

Brownlee, J., 2019. *Master Machine Learning Algorithms.* s.l.:s.n.

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
[Accessed 07 November 2019].

CodeScene, n.d. *Code Churn.* [Online]
Available at: https://codescene.io/docs/guides/technical/code-churn.html
[Accessed 01 November 2019].

Fenton, N. & Martin, N., 1999. Software metrics: successes, failures and new directions.. *The Journal of Systems and Software,* Volume 47.2, pp. 149-157.

Goyal, M., 2018. *Protecting individual privacy is important: Ben Waber, cofounder, Humanyze.* [Online]
Available at: https://economictimes.indiatimes.com/tech/hardware/protecting-individual-privacy-is-important-ben-waber-cofounder-humanyze/articleshow/64045690.cms
[Accessed November 2019].

Hackernoon, 2018. *10 Soft Skills Every Developer Needs.* [Online]
Available at: https://hackernoon.com/10-soft-skills-every-developer-needs-66f0cdcfd3f7
[Accessed 01 November 2019].

Hackystat, n.d. *A framework for analysis of software development process and product data.* [Online]
Available at: https://hackystat.github.io/
[Accessed November 2019].

Houlding, B., 2019. *STU3011 Notes.* [Online]
Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides3.pdf
[Accessed 03 November 2019].

Houlding, B., 2019. *STU3011 Notes.* [Online]
Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides9.pdf
[Accessed 04 November 2019].

Houlding, B., 2019. *STU33011 Notes.* [Online]
Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides7.pdf
[Accessed 02 November 2019].

Humphrey, W. S., 2000. *The Personal Software Process (PSP).* [Online]
Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

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

Lowe, S. A., 2018. *9 metrics that can make a difference to today's software development teams.* [Online]
Available at: https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams
[Accessed 04 November 2019].

Maslow, A. H., 1943. A Theory of Human Motivation. *Classics in the History of Psychology.*

Osetskyi, V., 2017. *SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral.* [Online]
Available at: https://medium.com/existek/sdlc-models-explained-agile-waterfall-v-shaped-iterative-spiral-e3f012f390c5
[Accessed 02 November 2019].

Oswald, A., Proto, E. & Sqroi, D., 2009. Happines and Productivity. *Journal of Labor Economics,* Volume [online] 33(4), pp. 789-822.

Sommerville, I., 2011. *Software Engineering.* 9th Edition ed. s.l.:Pearson Education.

Techopedia, n.d. *What is Software Engineering.* [Online]
Available at: https://www.techopedia.com/definition/13296/software-engineering
[Accessed 04 November 2019].

Techopedia, n.d. *What is Technical Debt? - Definition.* [Online]
Available at: https://www.techopedia.com/definition/27913/technical-debt
[Accessed 02 November 2019].

Weiss, E., 2002/03. Measuring LOC and other basic measurement. *Seminar on Software Cost Estimation.*