20TH NOVEMBER, 2018

SOFTWARE ENGINEERING PROCESS – MEASUREMENT & ASSESSMENT

INTRODUCTION:

This report will look into how software engineering can be measured in four specific ways. The first of these is the ways in which the software engineering process can be measured and assessed in terms of measurable data. Next, I will provide an overview of some of the computational platforms available that allow us to measure the software engineering process. Following that, I will delve into the algorithmic approaches that are currently being used to quantifiably measure a software engineer. Finally, I will look into the ethics concerns surrounding this kind of measurement.

Firstly, before we look into ways that are used to measure the software engineering process, we should first establish what software engineering is and its history briefly. The term "Software Engineering" was first coined at a NATO conference, when the difficulties and pitfalls of designing complex systems were frankly discussed (Wirth N, 2008). A search for solutions to allow programmers to work more efficiently began. The search concentrated on tools that would help programmers work better. Their search led them to developing programming languages that reflected the procedural, modular and then object-oriented styles. This has led us to nowadays, where constantly new languages are being created to further help the software engineer to be more and more efficient

MEASURABLE DATA:

One of the first problems for measuring the software engineering process, is to define what data should be measured. The collected data is only useful if it can "*provide useful information for project, process, and quality management, and at the same time that the data collection process will not be a burden on development teams*" (Kan, 2014). Therefore, all data collected should provide powerful insights into the work a software engineer is doing, and obtaining this data should not be difficult. This leads us back to the question as to what data should be collected

In any working field, a project will be assessed by the time it takes and the proportional revenue it is creating. These are two basic metrics that are applicable to most, if not all, software engineering projects globally. These two parameters form the basic constraints that face most firms operating today. Most firms operating are forever concerned about the time and money they are putting into a specific project, so to measure the productivity of the software engineering process using these two metrics would form a very stable base to quantifying how

productive the software engineering process is. But these two metrics would provide a very limited scope into the software engineering process, and we should expand the number of variables collected to further assess this performance.

There are two further types of measures needed in software development: product metrics and process metrics. Product metrics are concerned with things like code length, maintainability, reusability, complexity etc. Process metrics covers a number of changes in a class or in a file, editing time, commits etc. There is no consolidated way to measure these in the software engineering due to its intangible nature. "*Most of software development costs are human resources costs: experience, skills, etc. Moreover, the productivity of very good programmers is ten times better than average. For these reasons, it is very important to understand how top developers work and to encourage all developers to adopt a process that helps them to achieve the best possible results*" (Sillitti A et al, 2003)

In terms of collecting these metrics for measurement, we must first look at the techniques that will be used to collect this data, as these techniques are integral to collecting data on the process of software engineering. These techniques should be chosen on what sort of data is desired (Lethbridge, Sim and Singer, 2005). Some methods are only applicable in certain firms, as this is due to the amount of personal contact that is possible with the software engineer. Some techniques would require direct communication with the software engineer, or perhaps being able to view the software engineering process itself. Researchers who are collecting data must be aware of the access they will have to a software engineer before they commence collecting data. Each method will have its obvious advantages and drawbacks, but it will up to the researcher conducting the measurements to decide on the metrics and the techniques used in the research

Table 1. Data collection techniques suitable for field studies of software engineering.

Category	Technique
First Degree (direct involvement of software engineers)	Inquisitive techniques • Brainstorming and Focus Groups • Interviews • Questionnaires • Conceptual Modeling Observational techniques • Work Diaries • Think-aloud Protocols • Shadowing and Observation Synchronized Shadowing • Participant Observation (Joining the Team)
Second Degree (indirect involvement of software engineers)	• Instrumenting Systems • Fly on the Wall (Participants Taping Their Work)
Third Degree (study of work artifacts only)	• Analysis of Electronic Databases of Work Performed • Analysis of Tool Use Logs • Documentation Analysis • Static and Dynamic Analysis of a System

As we can see from the table above, the interaction with the software engineer will heavily influence the techniques chosen to collect data. This in turn will limit what sort of data can be collected

COMPUTATIONAL PLATFORMS AVAILABLE:

With the continuous advancement of technology, there is now a multitude of computational platforms available to measure the software engineering process. They provide a product to companies to allow them to monitor and keep track of the performance of the software engineering process at different levels. It is infeasible to create tailored platforms for the development process as it wouldn't be using time efficiently. These platforms would be one use and have to be tailored to each and every development process. Therefore, firms opt to use more generic platforms from other companies to ease the process.

There are many companies out there that have created platforms to easily measure and assess these metrics. Software as a service is very common in this area, which is when a company provides a software for a recurring fee as opposed to a one off payment. The firm in return gets ongoing support and up to date software. Researchers at the Collaborative Software Development Laboratory (CSDL) at the University of Hawaii have looked for analytics that help developers understand and improve the development process for the past 15 years. They have looked at various platforms, and how the platforms' capabilities have improved as the years have gone on. According to them, "*the easier an analytic is to collect and the less controversial it is to use, the more limited its usefulness and generality*"
(Johnson P, 2013)

Personal Software Process:

The first platform they looked at was the "Personal Software Process", originally developed by Watts Humphrey in his book "A Discipline of Software Engineering" (Humphrey W, 1995). This platform required the user to fill in a form, inputting data such as number of lines of code, time taken to complete a task, the quality of the code produced and progression against a schedule.

Name: Jill Fonson **Program:** Analyze.java

Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

Figure 1.
A sample defect-recording log. In the Personal Software Process (PSP), even compiler (syntax) errors are recorded. Developers typically find this aspect of the PSP to be onerous.

In one implementation, the PSP required the user to complete twelve spreadsheets and manually calculate data. This manual input of the data has its benefits and disadvantages. On the plus side, if the developer wanted to track a new statistic, all they would have to do is create a new spreadsheet. On the other hand, the manual nature of data input brought up data quality concerns, and in turn led to inaccurate conclusions being drawn from the data. This method was tedious and time-consuming, forcing the developer to fill in the data themselves, taking away time from them where they could have been writing code. To address these problems, the LEAP toolkit was developed.

LEAP Toolkit:

The LEAP toolkit addressed some of the problems that arose from the PSP. It automated the data analysis side, but still required manual input of the data. LEAP creates a portable repository of personal process data that developers can bring with them, as they move from project to project. "*After several years of using the LEAP toolkit, we came to agree with Humphrey that the PSP approach could never be fully automated and would inevitably require significant data entry*" (Johnson P, 2013)

Hackystat:

The study conducted by the CDSL goes on to describe a project called Hackystat, which tracks the software development process. *"Hackystat users typically attach software 'sensors' to their development tools, which unobtrusively collect and send "raw" data about development to a web service called the Hackystat SensorBase for storage"* (Johnson P, 2013). Hackystat has four main design features. It collects data on the client and server side, to gain a more complete view of the development process. The next feature of the program is it unobtrusively collects data, so users shouldn't notice when data is being collected. This raises concerns about the ethical nature of the software, as the user has no control over what data is being collected, but I will deal with this later. This feature allows for offline data collection, and does not require the user to log in every time they wish for data to be collected. The third feature of this program is fine grained data collection. By instrumenting client-side tools, it can collect data on a minute by minute basis, or even in real time as a developer is editing the code. The fourth feature is it allows for both personal and group based development. It can handle projects that are being worked on by one developer, as well as a collaborative project being worked on by more than one developer.

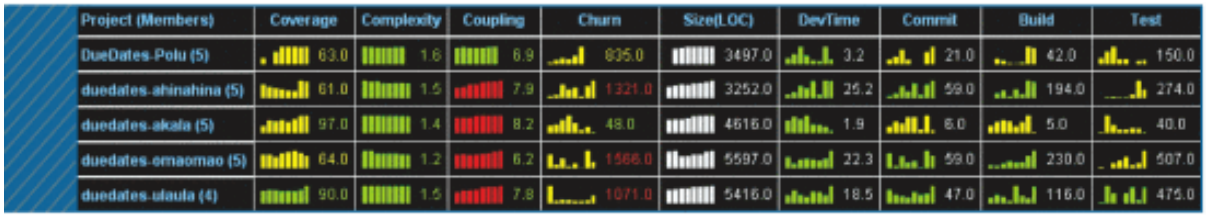


Figure 3.
A Software ICU (intensive care unit) display based on Hackystat. The Software ICU assesses a project's health both alone and in relation to other projects.

This program also offers the data in an easy to read and digestible format, as seen above. The metrics are colour coded, where green is the metric is performing well and red indicates that the specific metric is performing poorly.

Unfortunately for Hackystat, certain issues have arisen that have prevented it from being fully adopted into industry. The unobtrusive nature of data collection has been fought against by developers constantly. They do not benefit from the fact that they can't see what data and when it is being collected. The fine grained nature of the data collection creates disharmony among developers, as their up to date code is collected every minute or even faster. They are also uncomfortable with the power that the management have on them, as they can see the work that they are doing at every minute of the day. This has prevented Hackystat from fully establishing itself in the industry of data collection.

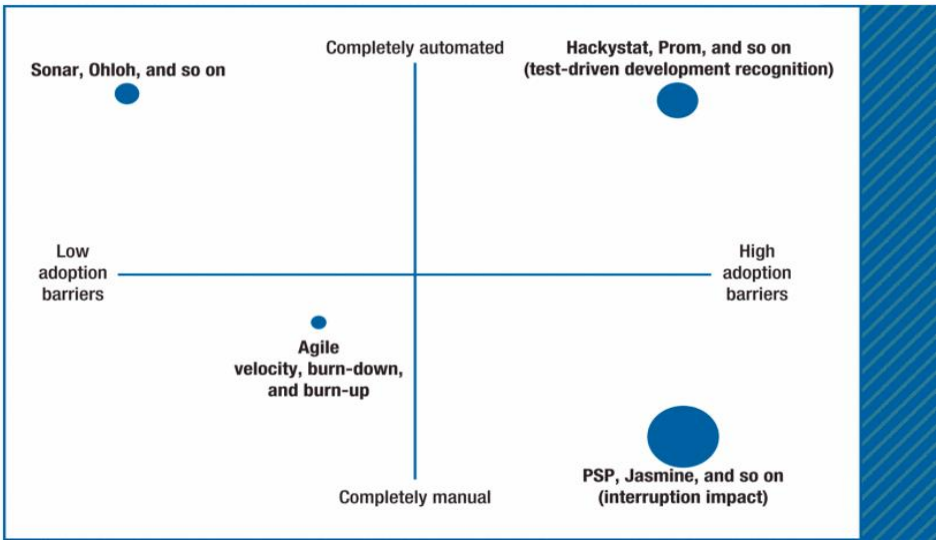
Other companies have filled the gap that was left by Hackystat, as the industry still longed for a data collection service or program that would use the automated nature of Hackystat, yet allow control over the data that was collected. One of these such companies is Code Climate.

Code Climate:

"*Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous and ubiquitous*" (Codeclimate.com, 2018). Code Climate is just one of a number of systems that have built on the foundation of Hackystat to provide an automated data collection service. Analytical techniques are applied to the data and the results are displayed to the user in a simplistic, graphical manner. Code Climate integrates with Github, an online environment where software engineers can host repositories which contain code for the public to see, and to also add to. Code Climate gets certain metrics from Github, such as technical debt, and can set acceptable levels of code quality and decide that can prevent development from going any further. This platform can also allow developers to view the most important changes made in a specific week, and can provide long term visuals.

Code Climate is just one of a number of companies that provide this analysis of the software engineering process. Each has their own benefits and drawbacks, and each are more suited to different types of data. So the type of data you wish to collect should influence your choice in which platform you would find the most applicable. If developer behaviour and practices is sought after, then the PSP method might be the most effective, but comes at the cost of manual entry and the question still remains about the quality of the data collected. If however a more automated system is sought after, then Code Climate or another system of the same ilk would be more appropriate, but it would not provide as comprehensive an analysis as the PSP method.

Figure 6.
A classification for software analytics approaches, including automation, adoption barriers, and the breadth of possible analytics the approach supports (indicated by the circles' size). In the parentheses are analytics that would be difficult to implement with techniques or technologies in the other quadrants.



ALGORITHMIC APPROACHES AVAILABLE:

To measure the performance of software engineers, it is often required that we design algorithms to help us assess the performance. An algorithm is a set of rules to be followed in calculations or in solving a problem. In this case the problem is to assess if the software engineer is performing adequately enough for the company. One way to do this is through artificial intelligence, or computer intelligence.

Artificial intelligence is intelligence displayed by machines. In contrast to natural intelligence displayed by humans, these machines mimic human thinking functions such as learning and problem solving. There has been a big push in recent years to implement AI into everyday life, such as driving. They are becoming closer and closer to fully automating the way we drive, and allowing machines to make decisions for us. Through artificial intelligence, we can train a computer how to do a particular process. Daniel Susskind preaches that artificial intelligence will transform professionalism and aid professionals in the long run by replacing the most repetitive work through automation. In a YouTube video in which he presents the idea of AI fallacy he says,

"The mistaken assumption that the only way to develop systems that perform tasks at the level of experts or higher is to replicate the thinking processes of human species" (YouTube,2018)

This is a very powerful statement, and brings up certain ethical issues that I will deal with at a later point in this report. There is only a certain amount of the human nature that can be replicated. In the video, Daniel identifies aspects such as creativity, judgement and empathy that characterise the human nature. We have yet to see a machine that is capable of displaying these characteristics in an accurate manner. Due to the fast nature of the learning curve of AI,

it is unwise to make the suggestion that one day a machine will be unable to display these characteristics that define the human species. With machines becoming increasingly capable of taking on more and more tasks that were once fully manual, it will not be long till artificial intelligence is part of everyday life.

As machine capabilities are ever-growing, their use to assess the software engineering process increases also. Computer intelligence is the ability of a machine to learn a specific task from a set of data and/or from experimental observations. CI is composed of a set of computational methods and approaches to address real world problems, where the process might be too complex for mathematical reasoning. CI can also be implemented in instances where the process is of a random nature and where there is an amount of uncertainty during the process. Therefore, it is also applicable to measure the software engineering process, as there is a high computational nature to the work of a software engineer, and the unique nature of coding from engineer to engineer can be handled by CI. There are five main components of CI (Siddique and Adeli, 2013):

- Fuzzy Logic is the measurement and process modelling that is made for real world problems. It can deal with incomplete data and therefore is applicable to a wide range of human activities, such as decision making. It does not, on the other hand, encompass learning capabilities

- Neural Networks has three components that work in unison. The first component is the one which processes the information, the next one send's the signals received from the processing and the final component controls the signals that are sent. It is a distributed information system that learns from experimental data. It accounts for fault tolerance like human beings.

- Evolutionary Computation is based on the theory of natural selection where only the strongest survive. Applications of this kind are found in the area of optimisation and multi-objective optimisation, where the problems are generally too complex for traditional mathematical techniques to be used, and so must implement computer intelligence to compute the solution

- Learning Theory in CI tries to approximate as closely as possible the reasonings of humans. It helps understanding how cognitive, emotional and environmental effects and experiences are processed and then making predictions off it.

- Probabilistic Methods utilises a more probabilistic approach to problem solving, and tries to evaluate outcomes of systems based on randomness, which is more align to the real world. It provides possible solutions to a reasoning problem, based on past experiences and knowledge.

These five components work together to allow computer intelligence to be applied to a wide range of applications. One such application of CI is its use in the cost estimation of a future project where the cost is modelled on a database of historical projects. Unfortunately, many of these historical databases contain missing information, but this is where the characteristics of the fuzzy logic component of CI is implemented. One of the characteristics of fuzzy logic is that it can handle the incompleteness of data. This means that the common practice of ignoring observations with missing data (which leads to inaccurate results) is replaced, allowing for a fuller representation of all of the data available. The neural network component of CI connects the needed data, and improves the overall result as more and more experimental data is fed to it. By feeding it with a vast amount of data, it can be trained to give accurate results/decisions depending on the application. It is also able to handle faults or errors more appropriately than humans, and as this occurs regularly in the real world, this is a real asset to CI. This enables CI to be used for real life applications such as software fault tolerance.

Evolutionary computation can be applied to a problem to find the optimal solution. This could be optimising the number of developers needed for a project so that no resources are wasted, finding the optimal budget to assign to a project, which developer to fire or optimising the software development process. Learning theory is the component of CI that tries to approximate behaviour and reasoning. This component allows the model to learn from what is has been working on, and aid it in making informed decisions based on its past experiences. It also allows it to perform future predictions based on this learning. This component accounts for the CI model to learn from a mistake, so for example if a project failed, the model would take this new knowledge into account and ensure that this same mistake is not repeated, and that better decisions are made in the future. The Probabilistic Methods account for the natural variance and errors in every model, which is an everyday occurrence, as it generally is impossible to predict with 100% accuracy what will happen. All we can do is estimate with varying degrees of certainty the most likely outcome.

In summary, there are indeed algorithms out there that can be trained to be capable of thinking and decision making. It is still unclear whether an algorithm will be able to fully capture the thinking process of a software engineer. But if computer intelligence is the way forward, there are clearly some tangible benefits to it. These models can be trained to implement a specific method and can be fed vast amounts. As these models take in more data than is imaginable for a human to be able to compute, it should in theory lead to better and more accurate results. There are limits to these models due to lack of creative and emphatic abilities, but for raw computational power they certainly excel.

ETHICS OF MEASURING THE SOFTWARE ENGINEERING PROCESS

Having looked at the data available, the computational platforms collecting the data, and algorithmic approaches to the data, we must now look at the ethical implications these approaches have on measuring the software engineering process

Tracking employee productivity is quite common in the workplace. The vast majority of companies partake in these sort of data-collecting activities to assess employee productivity and contribution. For example, call centre employees can be graded based on average call time, and sales people can be judged on how much they have sold. These statistics can be seen to reflect directly the work of the employee, whereas for a software engineer, these particular metrics are often not suitable. It may be unethical to track an employee to the levels that are required to fully assess the software engineering process. Personally, I believe that if the company is transparent with the data collecting method they are implementing, and are not collecting data just for the sake of collecting data, then I believe the software engineering process can be monitored like other professions.

From a legal standpoint, employers must be extremely vigilant when it comes to data protection. Due to the new general data protection regulation that has been in effect since the 25th of May 2018, organisations must place emphasis on "*transparency, security and accountability by data controllers and processors, while at the same time standardising and strengthening the right of European citizens to data privacy*" (Dataprotection, 2018). This regulation states that firms in breach of the regulation can be fined either €20 million or 4% of annual turnover, whichever is the larger of the two. This regulation is causing waves across all industries, as it is implementing a rigorous framework to keep data privacy standards high. Companies are moving to combat this extreme risk to the best of their abilities. This new regulation is extremely applicable to firms looking to measure the software engineering process, as collecting data as a software engineer is working is very close to stepping over the boundaries outlined in the regulation. As such any firm looking to implement measurement over their software engineering process must do so in a prudent and legal manner. Compliance with the regulation is critical.

The data that is being collected should relate solely to the work that is being conducted by the software engineer. There is no problem, in my eyes, with the data collection of employees so long as it does not cross the line where data pertaining to an employee's personal life is collected. The data collected should be related to the work that the employee is doing within an organization. Collecting too much data can infringe on the employee's right to privacy , and can sometimes reveal personal details about an employee. For instance, if the data collection revealed that an employee has a disability, or is a member of another marginalised group but did not disclose this to the employer. This would be a huge breach of the employee's right to privacy.

Although an employee is contracted to an employer, this does not provide the right to the employer to know the ins and outs of an employee's life. One must be careful not to passively track the personal details of an employee's life, say for example if a company phone or car is provided. In this day and age, mobile phones have GPS tracking enabled on them as a default. Things such as these should be thoroughly outlined in any employee contract, and strictly adhered to. It is all well and good stating your intentions to act accordingly to the new data privacy regulations, but if the actions do not follow suit, then there will be serious backlash. For example, it was recently revealed that Google continue to track some mobile phones even after location service have been disabled. If some scandal of similar nature came out about another company, its reputation would suffer a serious setback.

An example of a company potentially crossing the line is Humanyze, a firm that provides "people analytics" (Humanyze, 2018). Each and every one of their employees wear a credit card sized ID badge around their neck, which is full of the latest technology. These badges have the capacity to perform speech analysis, thanks to the built-in microphone. It is also able to track an employee's whereabouts, and is equipped with radio frequency identification (RFID) and near field sensors (NFC). The badge is even capable to detect when the wearer is face to face with another person. The function of the Humanyze badge is to collect data and analyse it in real time to provide useful information for management, but it all seems a bit too much. These capabilities seem to breach the privacy of an employee, as the management of the company are able to capture data on each and every movement of the employees, not just the work they are doing. The product currently claims that is does not record data when the wearer is in the bathroom, but how are we to know how accurate these claims are (Heath, 2016).

There are many benefits to collecting data in the workforce, but the welfare of the employee must come first. If the employees can't feel free to do whatever they want outside of work without feeling like they are being watched, then their employers have failed in some aspect.

CONCLUSION:

In conclusion, this report has outlined the key bases for the measurement of the software engineering process. This report has outlined how the software engineering process has developed over the past number of years due to metrics in place that have facilitated the growth of software development. For this surge to continue, it is necessary to identify what makes a software engineer productive and beneficial. I have identified what sort of data can be measured to be able to quantify the performance of a software engineer, and also the computational platforms available to assist in measuring the software engineering process. I also looked at an algorithmic approach to collecting data, but this brought up certain ethical concerns that I have dealt with. It is clear that the measurement of the process is possible on a number of levels but must be implemented in such a way to avoid the promotion of a bad internal atmosphere.

BIBLIOGRAPHY:

- (Wirth N, 2008) – "A Brief History of Software Engineering" – Available at: https://www.inf.ethz.ch/personal/wirth/Miscellaneous/IEEE-Annals.pdf [Accessed 18 Nov. 2018].
- (Kan, 2014) – "Metrics and models in software quality engineering" - [Place of publication not identified]: Addison-Wesley, [Accessed 18 Nov. 2018].
- (Sillitti et al, 2003) – "Collecting, integrating and analyzing software metrics and personal software process data" by A. Sillitti, A. Janes, G. Succi, and T. Vernazza, [Accessed 18 Nov. 2018].
- (Lethbridge, Sim and Singer, 2003) – Studying Software Engineers: Data Collection Techniques for Software Field Studies. Empirical Software Engineering, 10(3), pp.311-341 [Accessed 18 Nov. 2018].
- (Johnson P, 2013) - Searching Under The Streetlight For Useful Software Analytics IEEE Software (July 2013) by Philip M. Johnson [Accessed 18 Nov. 2018].
- (Humphrey W, 1995) - A discipline for software engineering - Reading, Mass.: Addison-Wesley [Access 18 Nov 2018].
- (Codeclimate.com, 2018) - *About | Code Climate*. [online] Available at: https://codeclimate.com/about/ [Accessed 18 Nov. 2018].
- (YouTube, 2018) - *The future of the professions: how technology will transform the work of human experts*. [online] Available at: https://www.youtube.com/watch?v=Dp5_ 1QPLps0 [Accessed 20 Nov. 2018].
- (Siddique, N. and Adeli, H., 2013) - Computational intelligence. John Wiley & Sons
- (Dataprotection.ie, 2018) - *GDPR - Data Protection Commission - Ireland*. [online] Available at: https://www.dataprotection.ie/docs/GDPR/1623.htm [Accessed 19 Nov. 2018].
- (Humanyze, 2018) - *Humanyze - Analytics For Better Performance.*. [online] Available at: https://www.humanyze.com/ [Accessed 20 Nov. 2018].
- (Heath, T, 2016) - This employee ID badge monitors and listens to you at work — except in the bathroom. [online] Washington Post. Available at: https://www.washingtonpost.com/news/business/wp/2016/09/07/thisemployee-badge-knows-not-only-where-you-are-but-whether-you-aretalking-to-your-co-workers/ [Accessed 20 Nov. 2018].