Measuring Software Engineering

Introduction

The last couple of decades has seen a meteoric rise in the use of computing technology and software engineering, both of which are continuing to advance into new uncharted territories. As a result of this exponential increase in the number of people who use technology, the volume and usage of data has also risen rapidly.
This exponential growth of data has led to the need for data analysis on a large scale. It has become a massive industry and a full-time job for many people. Every piece of data is evaluated in the attempts to learn from it and maximise efficiency is every aspect of our lives. The processes we use in our daily life are now under scrutiny as we have the means to measure and assess them. These processes, whether it's a cooking process or a goal-setting method, often have problems which need to be identified and resolved. Here we will specifically be looking at the software engineering process.

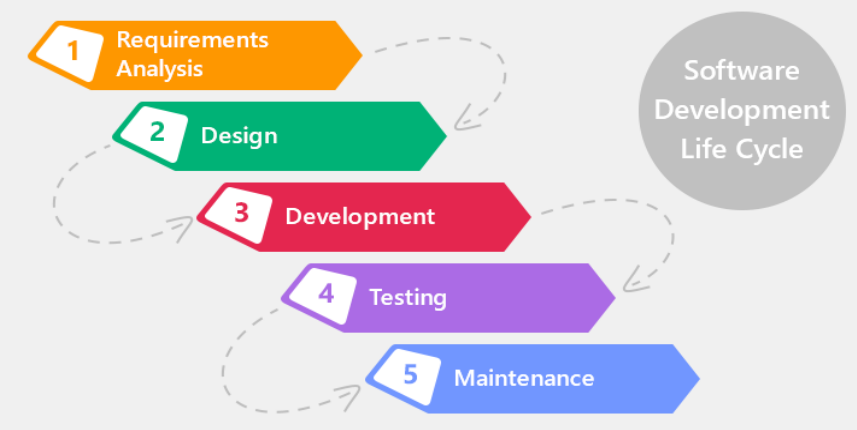
This report will look into the ways in which software engineering processes can be measured and assessed. The software engineering process can be measured in four specific ways.

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

Before we look at how software engineering is analysed, we must first look at what this process consists of.

Software Engineering Process

The software engineering process involves the division of work into several phases in order to maximise efficiency and increase output. This long, complex process used for developing a software project is called the life cycle.



Requirement Analysis: The first phase of software development involves extracting the requirements of the desired software product. This stage is the primary focus of the project manager. It can be compared to market research; a lot of questions are asked of the project in order to justify its production. Who is it for? What will it do? What input will it need? All of these questions are analysed and answered with all requirements accounted for. Then, specification occurs. This consists of putting together an exact description of the software which we require. This document will then serve as a guideline for the next phase of the process.

Design: The next step involves the preparation of system and software design from the requirement specifications which were outlined in the first phase. This phase is all about making sure the software system will meet the requirements of the product whilst ensuring it could address future requirements too. The system design specifications serve as input for the next phase of the model.

Development: This phase consists of the most obvious aspect of the process the coding. This stage is obviously the top priority of the software engineer or developer. A common misconception is that this stage requires the most work, however, this is generally not the case. If the first two phases were carried out thoroughly then the coding shouldn't cause much trouble. Reducing the product design to a block of code is easier when the guidelines from phase 1 can be assessed and the test strategy has already been devised.

Testing: The next phase of the software engineering process revolves around checking for bugs and errors in the code. This is particularly important when more than one programmer has worked on this project, as miscommunication can lead to bugs. The first prototype of the system will be beta tested by customers in order to establish any initial bugs. These errors must be dealt with before the final product in delivered to the customer.

Maintenance: The final phase of the software engineering process is surely the most important. After a system is up and running and people are using it, training and support are vital to the success of the product. Software can sometimes be difficult to use at first so people often stay away from new products. It is important to provide the necessary training to the users so that there is no confusion about how to use it. Maintenance ensures software can deal with newly discovered problems and requirements after the initial release of the software.

The software engineering process is long and can get very tricky if it's not carried out in an organised, structured manner. Every phase of the process is there to make the next phase easier for the next team working on it. There are many ways in which this process can be assessed in terms of *measurable data.*

Measurable Data

Why do we collect data?

The importance of collecting data lies in the ability to analyse the process in order to make the most of scarce resources. In simple terms, a company wants to measure time and money to ensure that their processes are maximising efficiency.

When analysing the software engineering process, the impact of data on these resources will be of much importance. Engineers collect and measure the data which is most likely to give insight into their processes and therefore help to improve them. This is very important as it establishes a pattern in their work, allowing them to streamline their process. In turn, better use of resources leads to reduced costs leading to business growth.

What data do we collect?

In the early days of software engineering, companies were often faced with the issue of how to collect data and what data to collect. Most data collection was done manually so there was much less volume of data available. Furthermore, data collection was time consuming and prone to human error.

One important ideology that came from this time was that data collection should provide information on the product, process and project but the process of collecting the data should not strain scarce resources(time and money). (Kan, 2003) According to this paper, there are three main measures required in order to analyse the software engineering project:

1. Product metrics
2. Process metrics
3. Project metrics

I will analyse these metrics and discuss how this data is collected in order to analyse the software engineering process. These can be measures in terms to internal and external attributes.
Internal attributes are concerned with only the metrics that can be measured in terms of the software product aside from its behaviour. External attributes, on the other hand, are concerned with how the product relates to its environment.

1. Measurement of Product: Software Engineering metrics allow software engineers to gain a better understanding of their processes and asses the quality of the software. Product metrics focus on four main areas: Specification, Design, Coding and Testing. Using the software product metrics we can gain a real-time insight into the software development. To ensure software has been properly developed and tested, it should include all four of these entities. The table below lists the entities alongside their internal and external attributes. (Xenos & Stavrinoudis, 2008)

Entities	Internal Attributes	External Attributes
Specification	Functionality, Size, Modularity, Redundancy	Comprehensibility, Maintainability
Design	Functionality, Size, Modularity, Coupling	Complexity, Maintainability, Quality
Coding	Size, Algorithmic Complexity, Functionality	Maintainability, Usability, Reliability
Testing	Size, Coverage	Quality

Product metrics can be very complex to define. A simple example of measuring testing data for a product is the number of errors it produces.

2. Measurement of Process: The process of software engineering involves looking at the steps involved in creating a software product. These include setting a specification, detailing the design and testing the product after its been coded. The table below lists the steps alongside their attributes, both internal and external.

Entities	Internal Attributes	External Attributes
Developing Specification	Effort, Time, Volume of Changes	Cost, Quality, Stability
Detailing Design	Number of Faults Found, Time, Effort	Cost, Effectiveness
Testing	Time, Effort	Cost, Stability

Process metrics can be very easy to measure. For example, time, cost and number of changes are all generally easy to track.

3. Measurement of Project: The easiest metric to measure is that of the projects. This encompasses looking at the resources of a project. The way a software engineering project is measured is just like the way in which normal projects found in the common workplace are measured. The three main entities which are analysed in project management are personnel, tools and environment. The table below lists these entities with their attributes.

Entities	Attributes
Personnel	Age, Experience, Salary
Tools	Price, Speed, Usability, Reliability
Environment	Cost, Size, Comfort

Computational Platforms Available

Once data has been collected, it must be analysed. Rapid analysis facilitates faster decision making which keeps a business' competitive edge in the market.
With the ongoing advancements in technology, there is now an array 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 would be infeasible to create tailored platforms for the development process as it wouldn't be efficient use of time. These platforms would be single 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 asses 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 a Collaborative Software Development Laboratory(CDSL) at the University of Hawaii have looked at 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 passed. According to them "the easier as analytic is to collect and less the 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"(1995). This platform required the user to fill in a form, inputting data such as a number of lines of code, time taken to complete a task and the quality of the code produced.

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. Also, the LEAP toolkit provided a lot of analysis that PSP did not provide such as regression analysis. 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 the data in themselves. This is obviously not very time-effective and they could have spent this time writing code. As a result, LEAP Toolkit was developed to deal with these inefficiencies.

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 developer 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) In order to carry out unbiased analysis,

manual input is not the way forward and this realisation lead to the development of more automated tools such as Hackystat.

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. It unobtrusively collects data, so users shouldn’t notice when data is being collected. This raises concerns about the ethical nature of the software and we will discuss the ethics later in the report. It allows for offline data collection and doesn’t require the client to log in every time they wish for data to be collected. The third feature 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. Finally, it allows for both personal and group-based development. It can handle projects that are being worked on collaboratively.

Unfortunately, certain issues have arisen in Hackystat that have stopped it from being fully adopted in industry. Developers have fought against the unobtrusive nature of data collection used. “Many developers were uncomfortable with the accessibility the platform have others into their work”. (Johnson. P, 2013) They are also uncomfortable with the power that the management have on them, as they can see the work they are doing at every minute of the day.

Other companies have filled the gap left by Hackystat, as the industry still longed for a data collection service with the features of Hackystat but also one which allowed control over what data was collected. Then along came Code Climate.

Code Climate

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 allows for code to be reviewed to ensure that is it easily understood it doesn’t repeat itself and it can be modified and reused with ease. It allows organisations to properly analyse their code quality through full test coverage and by reducing code complexity. It also tests code for maintainability, an important measure of produce discussed above. Code Climate allows companies to have an analysis of their software engineering product and process metrics. Code Climate ensures that any updated to code add increased value and are adequately tested(Code Climate, 2019).

The type of data you wish to collect should influence your choice in which platform you should use. If developer behaviour and practices is sought after, then the PSP method might be most applicable. If you are looking for a more automated system, then Code Climate is the better choice, but it does not provide as comprehensive an analysis as the PSP.

Algorithmic Approaches

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 the use of Artificial Intelligence(AI).

AI is intelligence displayed by computers. In contrast to natural intelligence displayed by humans, these machines mimic human thinking functions such as learning and problem solving. There has been a huge push in recent years to implement AI into everyday life. 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 he said "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, 2019) There is only a certain amount of human nature that can be replicated in machines. Susskind identifies creativity, judgement and empathy as the main human characteristics that machines are not yet capable of mimicking. However, due to the fast nature of the learning curve of Artificial Intelligence, the potential for machines learning human behaviour is limitless. With machines becoming increasingly capable of taking on more and more responsibilities, it will not be long until AI is an inescapable part of everyday life.

As machine capabilities grow, their use in the assessment of the software engineering process grows too. Computer intelligence is the ability of a machine to learn a specific task from a set of data and/or from experimental observations. It 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 used in instances where the process is of a random nature and where there is an amount of uncertainty. Therefore, it is also suitable for measuring the software engineering process.

According to Siddique and Adeli(2013), Computer Intelligence has five main components:

1. 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.

2. Neural Networks have three components that work together. Firstly, they process the information, then they send the signals received from processing and finally they control the signals that are sent. It is a distributed information system that learns from experiemental data.

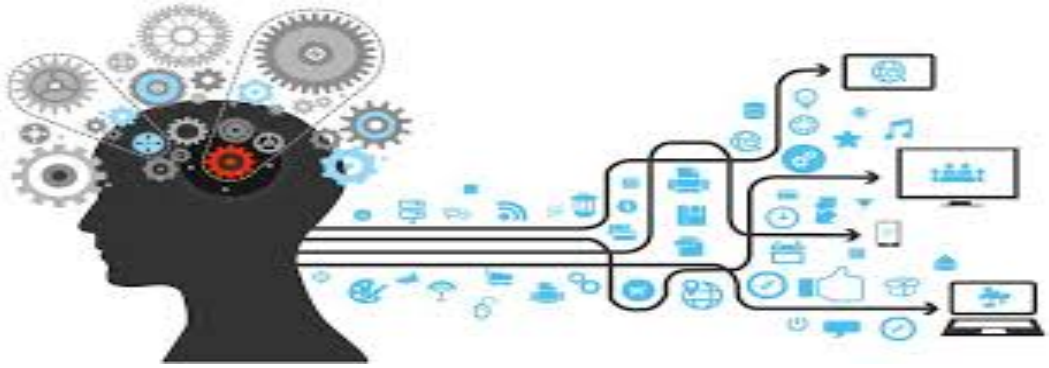
3. Evolutionary Computation is based on the theory of natural selection where only the strongest survive. Applications of this kind are found in the area of optimisation,

where the problems are generally too complex for traditional mathematical techniques to be used , so CI must be used to compute a solution.

4. Learning Theory in Computer Intelligence tried to approximate as closely as possible the reasonings of humans. It helps understanding how cognitive, emotional and environmental effects and experiences are processed. Predictions can then be made based off of these findings.

5. Probabilistic Methods utilise a more probabilistic approach to problem solving and try 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 application of CI is its use in estimating the cost of a future project where cost is based on a model built on historical project prices. In some cased the databases will be missing information – that's where fuzzy logic comes in. The neural network component connects all the needed data together and improves the overall result as more and more data is added. The more data the better it can be trained to give precise results. Evolutionary computation can be applied to a problem in order to find the optimal solution. This could be optimising the number of developers needed for a project or even finding the right budget. Computer intelligence tries to approximate human behaviour and reasoning through learning theory. This component allows the model to learn from what is has been working on and aid it in making informed decisions going forward. It also allows the computer to learn from a mistake and to not repeat a failed experiment without making changes. The probabilistic methods account for the natural variance and errors in every model which is an everyday occurrence and of course happens to humans too. This component allows us to 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 of information. 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 the lack of creative abilities but in terms to raw computational power they are unmatched.

Ethics

After discussing the software engineering process, how to measure the data and where it all comes from we're left with a very important question; is it all ethical? To be ethical means to be in line with moral principles. Because of the huge role computers play in every industry in today's society, software engineers. Those responsible for implementing the programs utilised by computers have significant opportunities to do good but also to cause harm.

To ensure software engineers do not cause harm in their work, they commit themselves to make software engineering a beneficial and respected profession. This is done by following the 8 principles of ethical software engineering which I will outline. They include an obligation to the general public, the client and employer, judgement, the product, the profession, colleagues, the engineer themselves and management.

The obligation to the public: This is perhaps the most obvious obligation of the software engineer. Software engineers must act in the interest of the public. After all, the public is who their software will affect the most and therefore care must be taken to ensure that they accept the full responsibility of their work. Developer are expected to balance the interest of the employer, client and public. If they were to focus on the interest of one party it would be detrimental to their product. Software should only be made accessible to the public when it is believed to be safe, meet specifications and pass relevant tests. A very interesting aspect of the engineers obligation to the public involves considering factors such as economic disadvantage, disability, and the allocation of resources and how this can affect access to the software.

The obligation to the client and employer: This obligation includes no knowing use of illegal or unethical software, the proper authorised use of client's property and ensuring relevant documents that need to be approved are done so by someone qualified. It is important for the engineer to keep any confidential information obtained in their professional work confidential. This is to ensure everything is kept legal. With GDPR being a huge buzzword nowadays, software engineers must ensure that nothing breaches this strict policy.

The obligation to their judgement: The part of the Software Engineering Code of Ethics seeks to see software engineers maintain integrity and independence in their judgement. This includes focusing on only promoting documents prepared under their supervision and within an area which they feel competent in.

The obligation to the product: The developer's obligation to the product is clear in that they must strive to ensure they meet the highest professional standards possible. Areas of this obligation include ensuring they are qualified to produce the relevant product and that they are themselves capable of understanding the specification for the software it requires. The software engineer should strive to fully understand the specifications for software on which they work.

The obligation to the profession: Software engineers are obligated to help develop an organisational environment which is favourable to acting ethically including promoting public knowledge of software engineering. Software engineers should strive to support each other

in striving to follow this code, In the best interest of the profession, software engineers should not act in their own interest at the expense of the profession.

The obligation to their colleagues: This involves software engineers being fait to and supporting their colleagues. This obligation is similar to that of any good co-worker in any field, in that they should assist colleagues in professional development, review the work of others were relevant and give a fair hearing to the opinions and concerns of their colleague.

The obligation to themselves: In regards to this obligation, software engineers shall continually endeavour to further their knowledge of development in the analysis, specification, design, development and maintenance and testing of software, together with the management of the development process.

The obligation to management: Software engineering managers should promote an ethical approach to the management of software development and maintenance. In particular, those managing or leading engineers should ensure that those developing products are informed of standards before beginning work on the project. The obligation of management ranges from ensuring work is only assigned after taking into account appropriate contributions of equation and experiences tempered with a desire to further that education and experience.

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 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. In the report I have identified the kinds of data that can be measured to be able to analyse the performance of a software engineer, and also the computational platform available to assist in measuring the software engineering process. I also looked at the algorithmic approach to collecting data but this brought up certain ethical concerns. I outlined the code of ethics that developers must follow in order to remain ethical.

Bibliography

Code Climate (2019) “About Us: Code Climate” Available at: https://codeclimate.com/about/ [Accessed 1 Nov 2019]

Ethics.org (2019) “Software Engineering Code” Available at: https://ethics.acm.org/code-of-ethics/software-engineering-code/ [Accessed 1 Nov 2019]

Humphrey, W (1995) “A discipline for software engineering” Addison-Wesley [Accessed 1 Nov 2019]

Johnson, P (2013) “Searching under the Streetlight for Useful Software Analytics” IEEE Software [Accessed 1 Nov 2019]

Kan, S (2003) “Metrics and Models in Software Quality Engineering” – Addison-Wesley Accessed 1 Nov 2019]

Siddique, N and Adeli N (2013) “Computational Intelligence” John Wiley & Sons [Accessed 1 Nov 2019]

Wirth, N (2003) “A Brief History of Software Engineering” – Available at: https://www.inf.ethz.ch/personal/wirth/Miscellaneous/IEEE-Annals.pdf [Accessed 1 Nov 2019]

Xenos, M and Stavrinoudis D (2008) “Comparing Internal and External Software Quality Measurements” Greece: ResearchGate [Accessed 1 Nov 2019]

YouTube (2019) “The future of the professions: how technology will transform the work of human experts Available at: https;//www.youtube.com/watch?v=Dp5_1QPLps0 [Accessed 1 Nov 2019]