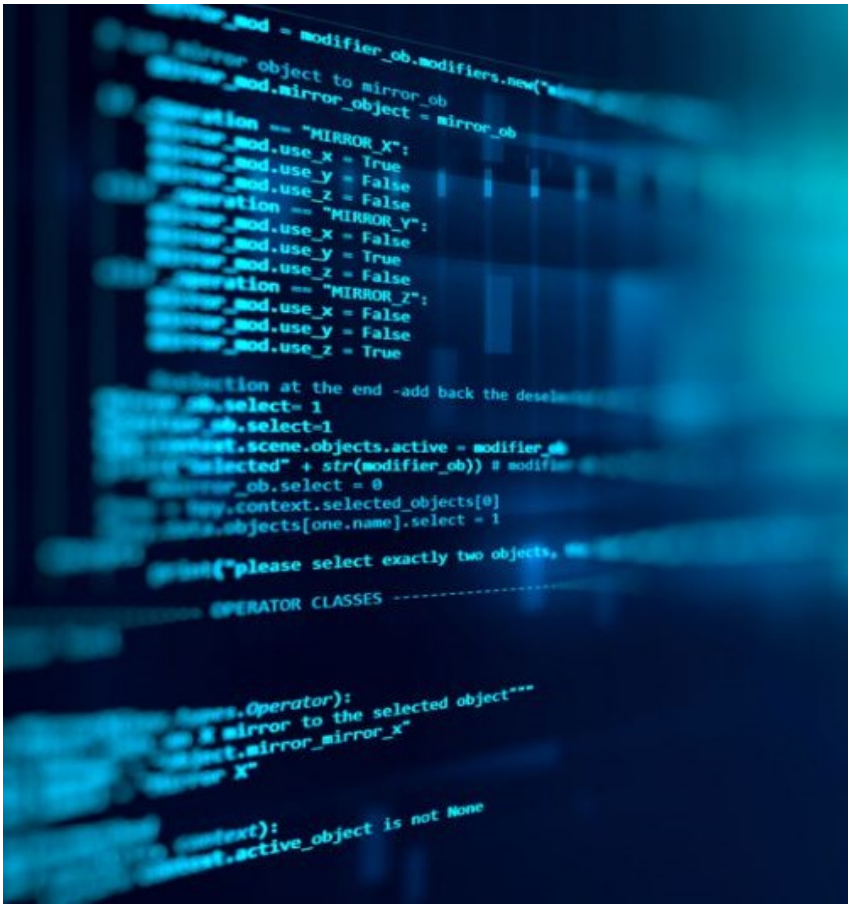
MEASURING SOFTWARE ENGINEERING



ABSTRACT

This report considers the way in which the software engineering process can be measured and accessed. It aims to do this by addressing the following topics which are; measurable data, available platforms, algorithmic approaches available and finally the ethics of such analysis.

WHAT IS SOFTWARE ENGINEERING?

Software engineering is the application of engineering to the development of software in a systematic method. Software engineers have to apply principles and techniques go engineering to design, develop, deploy, maintain and manage software systems.
Software measurement is vital in the software engineering discipline. The measuring itself is an extremely difficult task as modern software systems are very complex, this leads to the

complexity of engineering appropriate solutions to a given problem. As a result, the software engineer has to have a mastery of a wide range of skills with a deep understanding of the problem they are working on. Apart from that, they have to work closely and openly with team members to ensure optimal efficiency and a solution is reached.

MEASURABLE DATA

To try and determine a software engineer's productivity, the first measure we must overcome is picking what data used in studying their work. I will be looking at different ways of measuring data, such as:

- **Number of lines in code**
- **Cyclomatic complexity**
- **Code coverage**
- **Time taken per task**
- **Emotional state**

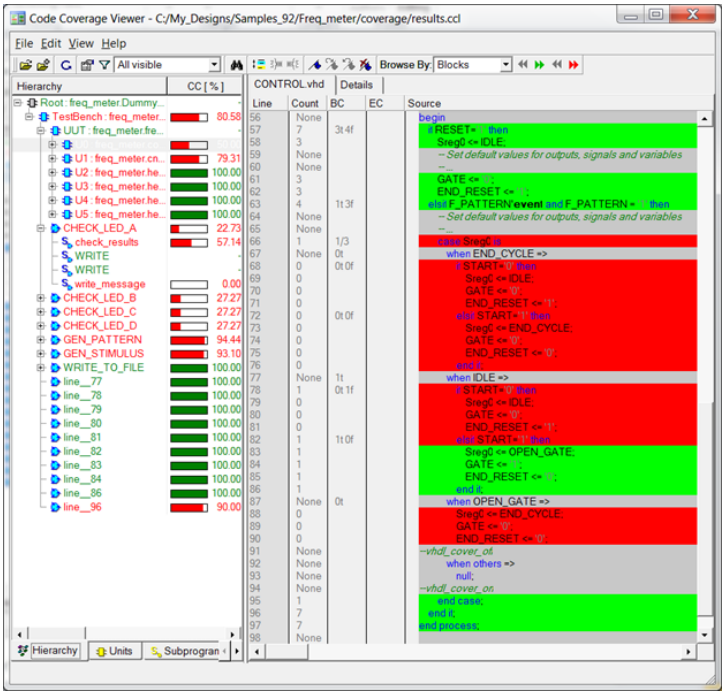
Number of lines in code

This is a deeply flawed way in trying to measure the productivity of a software engineer. In simple terms, an engineer with the most lines of code is considered as the most productive in the team. In a situation like this, software engineers are placed in a situation where they are competing for the most amount of lines of code written. Immediately, there are issues created from this.

Software engineers can falsely make their code appear longer than it actually is, in order to look better in comparison to other engineers. For example, things can be done like using more line space or using longer versions of different productions to inflate their line count. For example, in Java, using the operator '?' in place of an 'if' statement for conditional expressions would suffice. This can lead to a situation where attempts to measure productivity has led in the reduction of code quality. As Bill gates said, "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs".

Having more than an optimal amount of code in a solution is often undesirable. Shorter solutions are preferred cause they are visually concise and are usually more efficient. In

comparison, shorter lines of code does not always mean they are the better option. From these observations it is clear another way of measurable data is needed.



Cyclomatic complexity

Cyclomatic complexity is another data measurement that could be used in measuring software engineering. Cyclomatic complexity is measures the complexity of some code obtained, by seeing how many different ways of arriving to the solution that one can take. The less complex solution is seen as the better solution. Arguments for using cyclomatic complexity is that the more complex a piece of code is, the more chance it leads to a higher probability of bugs and other undesirable factors. However, cyclomatic complexity increases naturally with the size of the program, as do the errors.

Code coverage

Code coverage is a valuable form of data that can be used for the measurement of software engineering. It is a measure of how much a programme is ran when testing for it is carried out. It is an essential part of measuring data if the development in question being used is test driven. The idea is that 100% of the code is executed at least once during the test. We can use this measurable data to say that a software engineer with certainty has thoroughly investigated and tested the code they have written.

Comparing code coverage for different engineers can give insights into how well they investigate their code. A shortcoming is that high code coverage does not necessarily mean that the code is bug free. If the tests are insufficient, then full code coverage won't translate to good code coverage. This particular problem must be addressed when using this as a metric.

Figure 1: example of code coverage.

Time taken per task

Time taken to complete a project/task is another potential data measurement. Using this as a measurement, the software that gets their task completed first would be the better employee. While meeting deadlines for a given project is a critical part of any job, it is not always the case that quicker implementation is always the case for the optimal solution. A carefully planned, well though out piece of code will always be better than a quicker produced, but worse solution. Encouraging software engineers to quickly rush to a solution could lead to problems down the line, where issues or bugs arise that could have easily been detected and sorted at an earlier stage. It could also lead to less concentration and burnout amongst the team. While completing tasks on time is important to the software engineering process, it should not be considered as a standalone solution.

Emotional state

An alternative to the above approaches is looking away from data derived directly from the code, and to look at the software engineer instead. It has been stated that in general, a happy worker is a more productive worker. Happy workers are 37% more productive than those who aren't – *S.Anchor, "Positive Intelligence", Harvard Business Review (2012).*
Can this form of measurement be applied to a software engineer? How can happiness me measured and quantified? One solution is to give workers a sensor that measures physical activity taken by the wearer. The argument here is that the more physical activity leads to increased happiness and productivity. By following this logic, management may assign special days where employees perform physical activity in a fun way. While this does not give accurate measurements of the efficiency of a software engineer, it can lead to the overall well-being and productivity of the team.

As demonstrated above, there are a number of measurable sources of data that can be used when measuring the quality and efficiency of a software engineer. It is up the management of the company to decide which method best suits their needs and strategy, while also trying to be appropriate.

AVAILABLE PLATFORMS

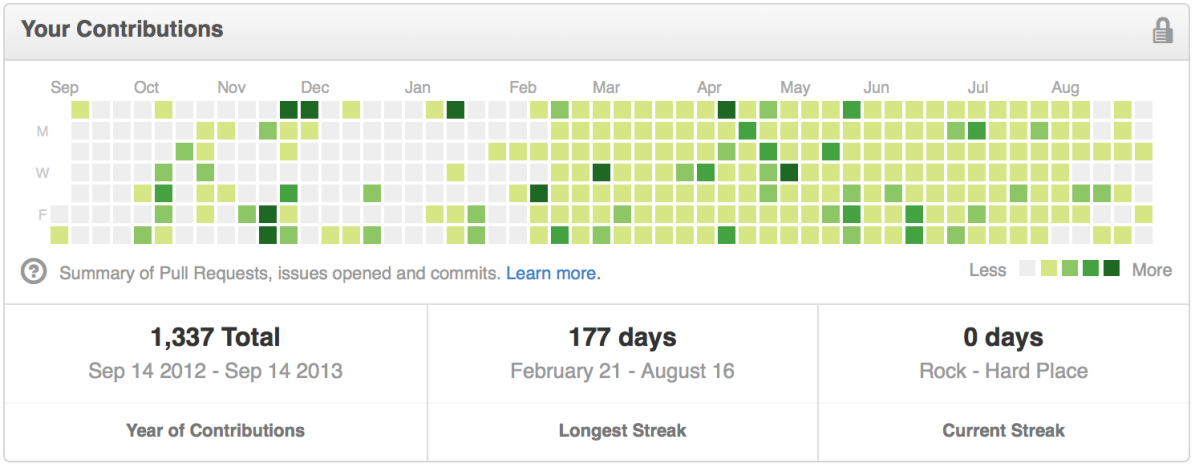
Platform overview

It is difficult to define tools to measure metrics when discussing platforms available to measure the software engineering process. The majority of time, metrics are internally defined and differ from project to project. Process metrics are usually defined based on the nature of the company and the products that they are creating. The pipeline to carry out the work will usually determine the basis from which the process metrics can be carried out.

Platforms for measuring the process

There are a number of platforms available on the market aiming to measure the software engineering process. Providing a product to companies, they enable companies to measure the software engineering process at different levels. GitHub is an example of a platform where we can see how data can be measured at a free, open-source, large scale level. GitHub is an online environment where software engineers can create repositories containing code, design specifications, mechanisms and everything else in between. It is a web based repository hosting service, offering distributed version control and source code management. GitHub provides organisations with the ability to measure key metrics relating to the software engineering process. Examples of metrics available on GitHub through the use of the REST API include:

- Number of code commits
- Number of contributions
- Frequency of contributor output



Above is a visual example of this. The heat map details the level of contributions over a period of time. While this visualization may look basic, it can provide meaningful insight into the level of contributions of certain individuals. For example, in this software engineering module, the students level and timeline of commitments is analysed as part of the decision on their final grade.

Other platforms that can measure metrics are integrated development environments (IDEs). Various IDEs contain their own testing suite, for example, Microsoft Visual Studio. As part of test functionality, Visual Studio contains a tool which can be used to calculate code coverage for each tested method. This can be ideal as it provides direct feedback to managers and engineering teams as they write and design their tests. Other IDEs also provide this feature. For example, a code coverage plugin exists for the Eclipse IDE called ‘EclEmma’, which I use & still use for some college assignments. These tools can be used by software engineering teams to ensure the process is being supported by proper unit testing. Such tools exist for the most popular programming languages, the general concept for representing code coverage is the same.

Another platform which can be used to measure the engineering process is the open source Hackystat reference framework. A “software project telemetry” facility is what Hackystat describes itself as. Shortly put, Hackystat automatically collects a large series of measurable data, which can then be accessed easily by the project team. The creators claim it can be used to “identify the source of problems added to a project”. They provide the example that if an increase in defect density is matched by an increase in complexity, then action should be taken to reduce complexity and see if his makes an improvement.

ALGORITHMIC APPROACHES

In collecting measurable data, there are many algorithmic approaches to consider. These include algorithms for calculating the data being measured, to algorithms changing the engineering process to allow measurement.

Process Quality Index

One way which in which software engineering can be measured is the process quality index. The PQI has 5 components and they are outlines in the following table.

Design table	Minimum {1.0, time spent in detailed design divided by the time spent in coding.}
Design review quality	Minimum {1.0, 2 times the time spent in detailed design review divided by the time in detailed design}
Code review quality	Minimum {1.0, 2 times the time spent in code review divided by the time in spent in coding}
Code Quality	Minimum {1.0, 20 / (10 + Defects / KLOC in compile)}
Program Quality	Minimum {1.0, 10 / (5 + Defects / KLOC in unit testing)}

The PQI will give a value between 0 and 1.0, and is a product of the five components. Values that are below 0.5 indicate that the product is likely to be of low quality – the lower the value, the lower the quality, (Pomeroy-Huff, 2008). This process is a simple algorithmic metric measuring the software engineer process, yet it can prove to be very insightful. Collecting data from the platforms mentioned above, it leverages that data and provides a detailed breakdown of product sections. Through implementation of this algorithm managers and software engineer team members alike can access and analyse the key areas of success and failure, improving the overall process moving forward.

Advanced data analytics

When assessing and trying to solve an analytical problem, it is extremely hard in the modern day not to come across the field of data analytics. Living in a technology heavy world today, with ever increasing amounts of data, we can see how data analytics is being leveraged into almost all corners of society today – from our home to our place of work. Relating to the

workplace, we can see that more frequently performed data analytics is providing a basis for a better output in work quality for software engineering. Previously having discussed the different types of data that can be collected and the platforms available for this data collection, we need to define and understand the analytical techniques that can be utilized to provide a real insight out of the data available.

To understand methods available, first we must define the nature of the data that we have collected. As detailed in the section above – measurable data, we identified key attributes that are available to measure. There are a multitude of different variables when measuring the software engineer process, as a consequence – we are discussing multivariate data analysis in contrast to one dimensional data analysis.

When dealing with multivariate data analysis there are two key groups of techniques used, which are clustering and classification. When dealing with these techniques, it's important to understand the notable differences between supervised and unsupervised methods.

Supervised methods – The majority of practical machine learning uses supervised methods. Supervised methods consist of when you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

Unsupervised methods - Unsupervised methods consist of taking in input data (X) and no corresponding output variables.
The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

In general we can assume that our data will define one of two states, bad engineering practice or good engineering practice. Following from this, supervised methods that can be used to measure software engineering include; kth-nearest neighbours, linear discriminant analysis, and logistical regression to name a few. The method which I believe can be leveraged the most effectively is discriminant analysis, which I will be exploring below.

Discriminant Analysis & Classification

Linear discriminant analysis is a classification technique, allowing you to leverage your multivariate data with the goal of assigning entities into a classification group. For example, the process quality index is calculated from a wide range of variables collected in the software engineering process.
For example purposes, at a basic level, we define two groups from our data; good practice/product & bad practice/product.
Linear discriminant analysis is essentially the process of defining a classification bound, e.g. 0.5, then classifying multiple instances of data by this boundary, thus creating a classification group.
Statistically, this is a fundamental method and today – it is seen to be modern practice with machine learning and artificial intelligence.
By using classification techniques in multivariate data analysis, it is evidently clear that they can be leveraged to as algorithmic process to gain a deeper insight and understanding into the large amount of data that is collected in the software engineering process. These advanced analytical methods are used on a wide scale in process optimization during production, across a vast array of industries from banking to agriculture.

ETHICS OF ANALYSIS

Measuring and monitoring employee performance metric is a common part of any sizeable business. Whether it be a food business or a technology firm, many companies take part in this practice – mainly to gain insights into the productivity and contributions of their employees, so they can reduce inefficiency or improve efficiency, if needs be. Of course, there are a number of ethical concerns which must be considered when measuring software engineering. These can range from measurements infringing on the privacy of software engineers to concerns over too much information being stored. No matter the scope of the work or measurement options being considered, these ethical concerns should always be raised. Even more critically, they should be sufficiently addressed before carrying out any monitoring of employees is started. This is important in protecting the rights of employees or teams working on a software engineering project. If not addresses, it could affect employees in a negative light. The company itself may even be compromised, losing talented staff or even taking on an unwanted court case. Therefore, it in both the interests of employees and management to ensure these concerns are addressed to avoid such problems for both parties.

From a legal standpoint, companies must be extremely careful when it comes to the protection of data. With the introduction of the new EU wide data protection regulation (GDPR), it is important companies abides to the rules to avoid consequences. This regulation states that firms in breach of it can be fined €20 million or 4% of annual turnover, whichever is the larger amount. GDPR caused waves across industries as companies ensured that their data collection and measurement practices aligned with regulation contents.

Personally, I believe it's important for companies wanting to improve/expand their measurement processes that they are tactful in the way they do so. In general, when new metrics re being defined in the workplace, it is extremely important to take into consideration the opinion and thoughts of the workforce. Failing to do so may result in a hostile working environment for both employees and management. When people feel like they are being monitored and scrutinized, there tends to be backlash – therefore, it is important that communication is a priority when considering this important topic.

In a nutshell, when a company is implementing measurement and analytical techniques to measure the software engineering process, it's important that they communicate this with their employees to ensure everybody is on the same page, and more importantly – everybody is content with decisions taken. After all, a happier workforce leads to increased productivity, so it's in a companies best interest to do this.

CONCLUSION

To conclude, this report has outlined the different ways the software engineering process can be processed. This was outlined in terms of software engineering topics such as; different types of measuring data, platforms that can be used for measuring this data, algorithmic approaches and ethical concerns that should be considered. It is evident that the measurement of software engineering can be performed on a number of levels, but organizations must take care in their approach in this implementation.

BIBILOGRAPHY

- Searching Under The Streetlight For Useful Software Analytics IEEE Software (July 2013) by Philip M. Johnson
- Brooks, F. and Kugler, H.J., 1987. No silver bullet (pp. 1069-1076). April.
- Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. 2004. How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Trans. Softw. Eng. 30, 12 (December 2004), 889-903
- Johnson, Philip M., et al. "Improving software development management through software project telemetry." IEEE software 22.4 (2005): 76-85
- Cockburn, A et al. Agile Manifesto, http://agilemanifesto.org
- Houlding, B. 2017. Introduction to Multivariate Linear Analysis. Available at: https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides1.pdf
- Pfleeger, S.L., Jeffery, R., Curtis, B. and Kitchenham, B., 1997. Status report on software measurement. IEEE software, 14(2), pp.33-43.
- Hassan, A.E. and T. Xie, Software intelligence: the future of mining software engineering data, in Proceedings of the FSE/SDP workshop on Future of software engineering research2010, ACM: Santa Fe, New Mexico, USA. p. 161- 166