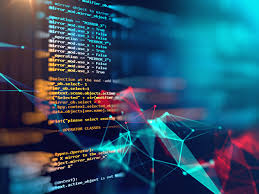
**Software Engineering Report**

Measurement & Assessment

****

David Berry

16334604

berryd1@tcd.ie

## Abstract:

This report considers the ways in which the software engineering process can be measured and assessed in terms of measurable data. It provides an overview of the computational platforms available to perform this work. The various algorithmic approaches available are analysed under software engineering metrics, and the ethical concerns and responsibilities involved in software engineering are also scrutinised.

## Introduction:

Let’s first begin by describing software engineering and its origins. Software engineering is a detailed study of engineering to the design, development and maintenance of software. Software engineering was introduced to address the issues of low-quality software projects. Software engineering first came about in the 1960’s due to the huge advancements made in the capabilities of computers from the 1950’s into the 60’s. These rapid advancements soon caused teething issues as not enough time was put into researching how best to develop programs that could match the improvements that had been made. This became known as the ‘Software Crisis’ as the individualistic approach to program development would not be feasible for larger and more complex projects. With the benefit of time, it has become apparent that larger projects require a greater number of programmers and a bigger emphasis on co-ordination and understanding as a result, aswell as adaptation which is vital when dealing with clients whose objectives can often be altering and unclear. Software engineering has helped shape the world around us and it plays a big role in how we interact with technology on a daily basis.

As stated by Ian Sommerville, software engineering is “critically important technology for the future of mankind”. Software engineering will always face issues even though it has evolved and improved remarkably since its beginning in the 1960’s. The main issues facing software engineering today are dealing with increasing diversity, demands for reduced delivery times and developing trustworthy software (Sommerville, 2011). Through accurate measurement and assessment of software engineering processes it is anticipated these issues can be resolved.

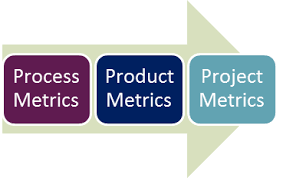
Measuring and assessing software engineering has long been debated, many feel it is only a waste of time as it is hard to quantify the work of a software engineer accurately. However, under strict principles and metrics, I feel strongly that their performance can be measured accurately when data is gathered concisely and interpreted correctly. In this report I will examine the software engineering process under four main headings:

1. Measurable data
2. Computational platforms available
3. Algorithmic platforms available
4. Ethical concerns

Each heading is equally important when comparing how software engineering processes can be measured and assessed to determine their respective performances.

## Measurable data:

As previously mentioned, it isn’t easy to measure the productivity and efficiency of a software engineer. To determine the productivity of a software engineer, we must begin by first choosing which ways to measure data. There is no one metric which is capable of measuring productivity conclusively. However, we can use several metrics to help paint a picture of the overall productivity of a software engineer. Unlike other industries, it can be hard to objectively measure productivity of software engineers hence it is important to have access to the appropriate data.

****Data is only useful however 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). A normal business will measure performance on time taken and monetary value added, which are two basics metrics that also fit software engineering but they will only provide a limited insight into software engineering performance. Other variables will also need to be considered as a result.

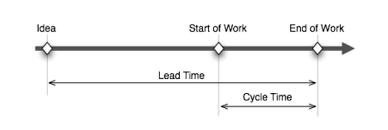
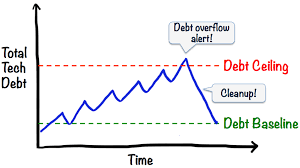
This is where we bring in software quality metrics to obtain the finer, more software specific details. We will examine product metrics, process metrics and project metrics which measure code quality and productivity.

1. **Process metrics** − These characteristics can be used to improve the development and maintenance activities of the software. Process metrics involve constructing specification, detailing design and testing. Ie. Effectiveness and quality of the processes that produce the software product.
2. **Product metrics** − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level. Product metrics are concerned with specification, design, coding and testing. Ie. Size, complexity, design features, performance, efficiency, reliability, portability etc.
3. **Project metrics** − This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity. Project metrics describe product quality. Ie. Quantify defects, cost, schedule, productivity and estimation of various project resources and deliverables.

I will now give a brief outline of some useful metrics which can yield worthwhile information:

1. **Code churn** is the percentage of a software engineer’s own code that represents an edit to their own recent work. It’s typically measured in Lines of Code (LOC) that were modified, added and deleted over a short period of time.

When code churn is suddenly higher than expected, it can be an early indicator that issues are occurring with the software engineering process. Detecting code churn early, is critical to ensuring early diagnosis of potential issues with the project.

1. **Lead time** is the time elapsed between the commencement of a project’s development and its delivery to the customer. Lead time can help management predict how much work a software engineer is capable of completing in a particular time frame and is a very good metric.
2. **Cycle time** is the length of time taken to make changes to the software system and convert the change to production. It is quite similar to lead time only cycle time starts when the software engineer begins work on the project rather than when the initial request is made.
3. **Number of commits** refers to the number of contributions that a software engineer makes to their source code. This can vary greatly depending on the developer. It provides us with a very good idea of if a developer is active, when they are most active and essentially, which developers are doing the most work.
4. **Technical debt** describes the additional development work programmers have due to implementing a solution which it is simpler in the short term but is not the overall optimal solution. Although it is inevitable for projects to hold some level of technical debt, management should endeavour to keep it minimal.
5. **Bug fixing** and bug handling does not specifically refer to the amount of bugs in the code but rather the time a software devotes to bugs each week. Both fixing issues once they have been identified or troubleshooting issues when they arise is what is entailed in bug fixing and handling.
6. **Test coverage** is a measure used to determine whether programmer’s tests cases are actually covering the source code and how much is executed when these test cases are run. It will include gathering information about which parts of a program are executed when running the test suite to determine which branches of conditional statements have been taken. It is a technique to ensure that your tests are testing your code or how much of your code you exercised by running the test.

## Computational platforms available:

Having considered the measurable data that should be collected in order to assess and measure the engineering process, we must now focus on the various platforms available to compute such data. There are a number of computational and software packages that allow us to do this. Code review is a crucial part of the development process as it makes you code stronger and reduces the risk of bugs.

There is a huge number of platforms available to analyse data. I will now look at some of the most prominent platforms available and compare their usefulness for software engineers in allowing them to interpret and develop their software engineering process.

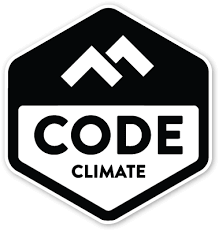
**Personal Software Process (PSP):**

In 1995, the Personal Software process (PSP) was created by Watts Humphrey to provide a platform for individual software engineers to acquire a disciplined and effective approach to writing programs. Personal software process (PSP), is designed to assist software developers in using sound engineering practices. PSP shows software developers how to plan and track their projects, use a measured and defined process, establish goals, and track their performance against these goals.

PSP assists software engineers to:

* Improve their planning and estimating skills.
* Make commitments and schedules they can keep and meet.
* Reduce defects in their projects.
* Manage the quality of their plans.

PSP assists engineers in managing software quality from the start of a project right through to completion, analysing the results of each task and using the results to improve the software process of the next project. However, PSP has been subject to some criticism and is not universally accepted in recent times. Some software engineers are of the opinion that PSP is adequate in theory but not in practice, as it is too inflexible and time consuming.

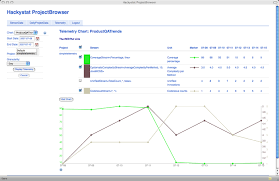
**Code Climate:**

Code climate is a software product that offers static program analysis tools for developers and was founded in 2011 by Bryan Helmkamp as a privately held company. Its focus is on helping software developers to improve the quality of their source code. Code Climate states on their website they specialise in ‘Providing meaningful and actionable engineering insights for the entire engineering organization.’(Code Climate, 2018). Code Climate now analyses billions of lines of code per day which shows the success and popularity of the company. Code Climate allows analysis of codebases written in Javascript, Ruby, PHP, Python and CSS, etc. Code Climate is a well-developed and very stable solution with a great number of features. It has many advantages over its competitors and many big players recommend it as the best option.

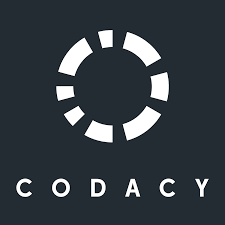
* A great number of supported languages, technologies and frameworks
* Very stable
* Nice new UI
* Well-maintained test coverage feature gem
* Browser extensions

Code Climate helps developers control the quality of their code, by combining fully-configurable test coverage and maintainability data throughout the development workflow. However, Code Climate is one of the most expensive tools, have unpredictable API and seem to have little interest in extending the tool the way the customer may suggest.

**Hackystat:**

Hackystat is a data collection tool created by The University of Hawaii due to increasing difficulties with other tools such as PHP and Leap. Hackystat describe themselves on their website as ‘A framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data’. Hackystat implements a service oriented architecture in which sensors, attached to development tools, gather process and product data and send it to a server, which other services can query to build higher-level analyses. 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 Sensor Base for storage. “The goal of Hackystat is to provide an extensible mechanism that can radically reduce the overhead associated with collection of a wide variety of software engineering data, along with a sophisticated toolkit of analyses that can facilitate useful interpretation” outlines how Hackystat filters and condenses data efficiently(P. M. Johnson, 2007). Hackystat has four key design features namely server-side data collection, unobtrusive data collection, fine-grained data collection and personal/group-based development. Unfortunately, Hackystat isn’t as popular as it once was as it has faced ethical concerns from the public as Hackystat collected data from developers without their prior knowledge. As a result, Hackystat went from being a revolutionary invention to an unethical, invasive platform.

**Codacy:**

Codacy was founded in 2012 in Portugal by Jaime Jorge with the simple mission statement to “help developers ship better code, faster”. Codacy is used by over a thousand companies for code review and analysis to show not only code quality, but improvement over time as well. Codacy is known to have a clean user interface which makes it easy to find important information. Codacy have many notable clients including PayPal, Deliveroo and Adobe. Codacy provide many cutting-edge statistics such as churn, complexity, duplication and number of lines of code. Here are Codacy’s main features:

* Code review automation.
* Code quality analytics.
* Security code analysis.
* Cluster installation/multiple instances.

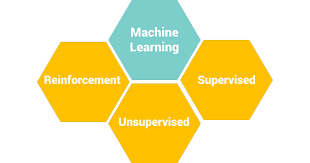
Codacy supports many languages including Java, Python, Javascript, Ruby, etc. Codacy has many benefits such as intuitive UI, time to fix estimation for each issue, delivering fresh features frequently and frequently check security issues.

## Algorithmic approaches available:

In this section of the report I will discuss the process of measuring software engineering through various algorithms which analyse the metrics discussed in the ‘Measurable Data’ section of this report. What does Algorithm mean? An algorithm is a step by step method of solving a problem. It is commonly used for data processing, calculation and other related computer and mathematical operations. An algorithm is also used to manipulate data in various ways, such as inserting a new data item, searching for a particular item or sorting an item. I will look at how different machine-learning algorithms aim to improve the structure of data and produce coherent, efficient results.

**Machine-Learning Algorithms:**

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. Machine-learning is playing a huge role in developments of artificial intelligence (AI) which is constantly being improved, taking technology to new realms as a result. Machine learning has become much more accessible in recent times. There is still a human element to algorithmic analysis but the accessibility to machine learning in the current climate allows for a whole new world of possibilities. Machine learning provides many advantages to global companies as it can greatly enhance an organization’s chance of detecting lucrative opportunities. Machine learning algorithms can be split into three general categories: supervised learning, unsupervised learning and reinforcement learning.



**Supervised Learning:**

Supervised learning is where 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. Supervised learning problems can be further grouped into regression and classification problems.

* **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
* **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

The idea behind supervised learning is that the output variable can be predicted when corresponding new data is inputted. An example of a supervised learning algorithm is Decision tree analysis which use tree-like modelling with nodes, branches and leaves. There are two phases in decision tree construction algorithms, firstly a very large tree is constructed and then pruned to prevent over-fitting. The pruned tree is then used for classification purposes.

**Unsupervised Learning:**

Unsupervised learning is where you only have input data (X) and no corresponding output variables. Unsupervised learning problems can be further grouped into clustering and association problems.

* **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
* **Association**:  An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

The unusual thing about unsupervised learning as it uses information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance, hence we are completely dependent on the algorithm to uncover the data correctly. Unsupervised learning can carry out more complex tasks than supervised learning systems but it can be unpredictable as it may include irrelevant clutter instead of ordering the data efficiently.

An example of an unsupervised learning algorithm is K-means clustering. K-means algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It works by specifying number of clusters, initializing centroids and randomly selecting K data points for the centroid without replacement and iterating until there is no change to the centroids.

**Reinforcement Learning:**

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. Hence, rewards are made when observations are made and actions are taken as a result. It is a beneficial form of algorithm as it can produce the perfect model to solve a problem, can solve very complex issues and can outperform humans in many tasks. However it can encounter some issues, such as it isn’t ideal for simple problems, it assumes the world is Markovian which it isn’t and it can be quite expensive to use.

An example of a reinforcement learning algorithm is the Markov Decision Process (MDP). MDP is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker (Howard, 1960). Each time the algorithm makes a poor decision, it learns from it and will not repeat the mistake. This process repeats until the solution is found.

## Ethical concerns:

Ethics, also called moral philosophy, the discipline concerned with what is morally good and bad and morally right and wrong. The term is also applied to any system or theory of moral values or principles. More importantly, we must now focus more precisely on ethics in relation to software engineering specifically. I will first look at the legal aspect of collecting and analyzing data which have become increasingly strict in recent years and I will then evaluate the steps which need to be taken in the software engineering industry to prevent the breach of these laws and the ethical responsibilities the industry must have.

Legal issues have become increasingly common for big organizations as the data protection laws have become very strict. Organizations across Europe have felt the effects of General Data Protection Regulations (GDPR) which came into effect in May 2018.  The fines for violating the GDPR are very high.  Companies are now required to be far more transparent and stringent when it comes to data collection and usage. “If you process the personal data of EU citizens or residents, or you offer goods or services to such people, then the GDPR applies to you even if you’re not in the EU. There are two tiers of penalties, which max out at €20 million or 4% of global revenue (whichever is higher), plus data subjects have the right to seek compensation for damages.”(Citizens Information, 2018).

There is good reason for these increasingly stringent data protection laws as it has come to light in recent years of several cases where companies broke the trust of customers and users in relation to the data they collect. Not so along in 2015, news broke that Volkswagen had their software engineers program their cars so that they would cheat emission standards. It was revealed up to 11 million cars were programmed in this way. Another front age story was the case of Cambridge Analytica who manipulated the Facebook users to help with President Trump’s 2016 election bid. They gathered data from Facebook users and built personality profiles on these people so that they could target specific people, which is coined ‘psychographic targeting’. The firm is accused of using and secretly storing the data of almost 50 million Facebook users without their permission.

To prevent such huge scandals and breaches of trusts of millions of people worldwide in the coming future it is of upmost importance that the software engineering industry acts accordingly. All software engineers must abide by the ‘Software Engineering Code of Ethics and Professional Practice’ and receive the sufficient education to act in a socially responsible manner when dealing with data. Information gathered must also directly help to improve software engineering process, while software engineers must only collect data that is linked to their work. It is evident that the software engineering industry must only collect and use data to help improve process which will increase productivity, rather than the exploitation of individual’s private information. It’s essential for clearly defined boundaries, laws and regulations to be established. GDPR within Europe has gone some way to reduce the misuse of data with increased transparency and heavy fines now being imposed.

## Conclusion:

To surmise, measuring and assessing the software engineering process is a complex and difficult task. I have outlined in this report the different ways the software engineering process can be measured and assessed by discussing different ways of measuring data, platforms and algorithms that can be used and the ethical concerns that exist. I believe that measuring the performance of software engineers can be extremely beneficial when carried out in the correct manner, however organizations must be cautious in the approach they take.

## Bibliography:

Bjorn W., 2019. Personal Software Process. [Online] Available at: https://explainagile.com/agile/personal-software-process/

Brownlee, J., 2016. *Supervised and Unsupervised Machine Learning Algorithms.* [Online]   
Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Carpenter, W., 2018. Code Climate: How it Works and Makes Money. [Online] Available at: https://www.investopedia.com/articles/investing/022716/code-climate-howit-works-and-makes-money.asp#ixzz5XCtdZq26

Citizens Information, 2018. Overview of the General Data Protection Regulation (GDPR). [Online] Available at: http://www.citizensinformation.ie/en/government\_in\_ireland/data\_protection/overview\_o f\_general\_data\_protection\_regulation.html

Codacy, 2019. About. [Online] Available at: https://www.codacy.com/about

Codebeat, 2018. *How is Codebeat Different.* [Online]   
Available at: https://hub.codebeat.co/docs/how-is-codebeat-different

Codeclimate.com, 2018. About | Code Climate. [online] Available at: https://codeclimate.com/about/

Crunchbase, 2018. *Codacy.* [Online]   
Available at: https://www.crunchbase.com/organization/codacy#section-overview

Dataprotection.ie, 2018. GDPR - Data Protection Commission - Ireland. [online] Available at: https://www.dataprotection.ie/docs/GDPR/1623.htm

Encyclopedia Britannica, 2019. Ethics Philosophy [Online] Available at: https://www.britannica.com/topic/ethics-philosophy

Fenton, N., 1999. Software Metrics: successes, failures and new directions.

Gwyn Topham, S. C. C. L. P. S. a. M. F., 2015. *The Volkswagen emissions scandal explained.* [Online]   
Available at*: https://www.theguardian.com/business/ng-interactive/2015/sep/23/volkswagen-emissions-scandal-explained-diesel-cars*

Hackystat, 2018. About. [Online] Available at: https://hackystat.github.io/

Howard, 1960. Markov Decision Process. [Online] Available at: https://en.wikipedia.org/wiki/Markov\_decision\_process#cite\_note-FOOTNOTEHoward1960-2

Johnson, P., 2003. Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. 25th Annual Conference on Software Engineering.

Johnson, P. M., 2013. Searching under the streetlight for Useful Software Analytics. IEEE Sofware, 30(4).

Kan, S. H., 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kan, 2014. “Metrics and models in software quality engineering” - [Place of publication not identified]: Addison-Wesley

Psaila, S. B., 2018. *Cambridge Analytica explained: The facts, implications, and open questions.* [Online]   
Available at: https://dig.watch/trends/cambridge-analytica

Sillitti, Janes, Succi and Vernazza, 2003. Collecting, integrating and analyzing software metrics and personal software process data.

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

Techopedia, 2018. *Software Engineering.* [Online]   
Available at: https://www.techopedia.com/definition/13296/software-engineering

Tutorials Point, 2019. Software Quality Metrics. [Online] Available at: https://www.tutorialspoint.com/software\_quality\_management/software\_quality\_management\_metrics.htm