MEASURING SOFTWARE ENGINEERING

1. ABSTRACT

In this report I will discuss the ways in which software engineering process can be measured and assessed. I will do this through the following topics: measurable data, platforms available to do this, algorithmic approaches and the ethics surrounding this type of analysis.

This report will be compiled using a variety of academic material on the subject of software engineering as well as the content discussed in the lectures of CS3012.

2. INTRODUCTION

In order to understand how to measure and assess software engineering, we must first understand what it means. In 1968 at a NATO conference, the term “software engineering” was first used. It was at this conference that it was decided that a systematic approach, similar to that of physical engineering, should be applied to the development of software. Software engineering encompasses all aspects of the development of software, including, designing, constructing, testing and maintaining software systems. (Rouse, 2016)

Since 1968, software engineering has developed immensely. Software engineering has helped to form the world around us today. It helps to determine how we engage with technology on a daily basis. It is with this rapid growth in software engineering that the need arose to measure and assess the process of software engineering. In this report I will analyse how this process is measured under four main headings:

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

In order to look at how the software engineering is analysed, I must first look at the current software engineering process.

3. SOFTWARE ENGINEERING PROCESS

The software engineering process is a set of steps that carry out related activities that lead to the development of software. In any software engineering process there are four activities that must be carried out regardless of the project.

1. Software specification
2. Software design & implementation
3. Software verification & validation
4. Software maintenance

Each of these activities is a broader umbrella for sub-activities, such as unit testing, however, the any type of process must include the above activities in order to properly develop a software system. These activities can be implemented in a variety of processes such as, waterfall, agile and iterative. For example, the waterfall process follows a sequential order and is a plan driven process. Comparatively agile processes are a flexible approach which takes aspect of the iterative and incremental methods. Agile methods have a set of principles which outline the purpose of the process: (Gabry, 2017)

- Customer collaboration
- Incremental delivery
- People not process
- Embrace change
- Maintain simplicity

Two common types of agile methods used in industry are Scrum methods and Extreme Programming. Extreme Programming's core values of simplicity, communication, feedback, respect and courage tie in with the outlined values of agile methods. Extreme programming aims to provide adaptability over predictability by carrying out work in small iterative steps in order to allow customer collaboration. (Beck, 2000) The Scrum method follows a similar ideology by carrying out a list of prioritised tasks over the course of a sprint, a set timeframe in which tasks are completed, and regular meetings occur over the course of the sprint. The scrum method, similarly to extreme programming, breaks a project down into a set of iterative steps. (Littlefield, 2016)

Agile methods are most commonplace in the software engineering industry. In the 12th Annual State of Agile Report, 97% of companies surveyed used agile methods in software development with 56% of these companies using the Scrum method. The current software engineering landscape shows a heavy focus on agile methods and the use of iterative development and daily communication. (VersionOne, 2018) As a result of this, I will mostly focus on how companies assess and measure the software engineering process based on agile methods.

4. MEASURABLE DATA

A. WHY DO WE COLLECT DATA?

> *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. (A. Sillitti, 2003)*

The importance of collecting data lies in the ability to analyse the process in order to make the most of scarce resources. At the most basic level, any company wants to measure time and money to ensure that their processes are maximizing efficiency in these areas. When analyzing the software engineering process, the impact of data on these resources will be of much importance to a team.

Gathering data will allow more adaptability for a business. As discussed earlier, agile methods place a much larger focus on the human element of software development. As a result, software engineers are very focused on the human effort required to develop and maintain systems. Collecting and measuring data which looks at this human effort will allow them to improve their software development process. This is a vital as it will allow software engineers to establish patterns in their work, allowing them to streamline their process. All of this leads into business growth due to better use of resources.

B. WHAT TYPE OF DATA DO WE COLLECT?

In the early days of software engineering, one of the biggest issues that companies were faced with was how to collect data and what data to collect. At this time manual data collection was commonplace. It was time consuming and prone to human error. As the industry developed so did the ideas around collecting data. An important ideology that developed was that data collection should provide information on the product, process and project but the process of collecting data should not cause a strain on resources, such as, time and money. (Kan, 2003)

As Kan laid out, there are three main measures required in order to analyse the software engineering process: product metrics, process metrics and project metrics. I will look at these metrics in further detail and discuss how this data is collected in order to analyse the software engineering process.

These bases can be measured in terms of external and internal attributes. External attributes are concerned with metrics that can be measured only with respect to how the product relates to its environment. On the other hand, internal attributes are concerned with only the metrics that can be measured in terms of the software product aside from its behavior.

I. MEASUREMENT OF PRODUCT

Software product metrics allow software engineers to gain a better understanding of their processes and assess the quality of the software. Software product metrics focus on four main areas: Specification, Design, Coding and Testing. By measuring the software product metrics, we are able to gain a real-time insight into the software development. All good software should include these four entities as they ensure it has been properly developed and tested. The below table outlines the internal and external attributes associated with the four main areas of product. (Xenos & Stavrinoudis, 2008)

Entities	Internal Attributes	External Attributes
Specification	Functionality, Size, Modularity, Redundancy	Comprehensibility, Maintainability
Design	Functionality, Size, Modularity, Coupling	Complexity, Maintainability, Quality
Code	Size, Algorithmic Complexity, Functionality	Maintainability, Usability, Reliability
Test Data	Size, Coverage Level	Quality

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

II. MEASUREMENT OF PROCESS

The process of engineering involves looking at the steps involved in creating a software product. These include developing a specification, detailing the design, implementing code and testing the product. The internal attributes of a process include effort, time and quantity of changes. The below table outlines the internal and external attributes associated with the processes involved. (Xenos & Stavrinoudis, 2008)

Entities	Internal Attributes	External Attributes
Constructing Specification	Effort, Time, Quantity of Changes	Cost, Quality, Stability
Detailing Design	Number of Faults Found, Time, Effort	Cost, Effectiveness
Testing	Number of coding areas, Time, Effort	Cost, Stability

It is very easy to track process metrics such as time, efforts and number of changes made.

III. MEASUREMENT OF PROJECT

The easiest metric to measure is that of the projects. This encompasses looking at the resources of a project. The way in which a software engineering project is measured is very similar to normal

projects found in the common workplace. The three main entities which are analyzed in relation to project management are personnel, tools and environment. The below table outlines the attributes associated with measuring project performance. (Xenos & Stavrinoudis, 2008)

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

5. PLATFORMS AVAILABLE

A. WHAT IS A PLATFORM FOR?

Once data has been collected, it must be analysed. Rapid analysis facilitates faster decision making which keeps a business' competitive edge in the market. Over the years a number of computational platforms have been released which collect and analyse software engineering metrics. While the software engineering process is easily measured, it is much more difficult to measure software engineering product metrics. As a result, a vast majority of companies the definition of product metrics is dealt with internally and will vary greatly between companies. Product metrics are measured with the nature of the company in mind and due to this specific nature it means there are few products available in the market to carry out this work. As a result, I will only be looking at platforms which allow for the analysis of the software engineering process metrics.

B. BREAKDOWN OF PLATFORMS

There is a huge numbers of platforms available to analyse data. I will now look at some of these available platforms and compare their usefulness for software engineers in allowing them to understand and improve their software engineering process.

I. PERSONAL SOFTWARE PROCESS (PSP)

The PSP platform was developed by the Software Engineering Institute (SEI) to look at potential improvements for individual software developers. In the 1990s, Watts Humphrey wrote a book outlining the PSP. Humphrey believed that a structured development process as well as constant tracking of progress would allow the engineer in question to streamline their process, making it more efficient.

PSP is mainly focused on manual input, where the engineer fills out a variety of forms which require personal judgement. The main drawback of this analysis is that manual input has a greater risk of human error. This resulted in incorrect conclusions being drawn. PSP's manual input meant that for

many it would be more beneficial from their own experience than the platform itself. As a result, Leap toolkit was developed to deal with these inefficiencies. (Johnson, 2013)

II. LEAP TOOLKIT

Leap toolkit aimed to deal with the data quality issues presented by PSP. It did this by automating the data analysis process. The new technique still requires a level of manual input from the developer, and is therefore, still prone to human error. The Leap toolkit provided a lot of analysis that PSP did not provide such as regression analysis. After the development of the Leap toolkit and its longstanding presence in the industry, it soon became clear that there would never be a way to fully automate the PSP as manual input would always be required in some form. 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. (Johnson, 2013)

III. HACKYSTAT

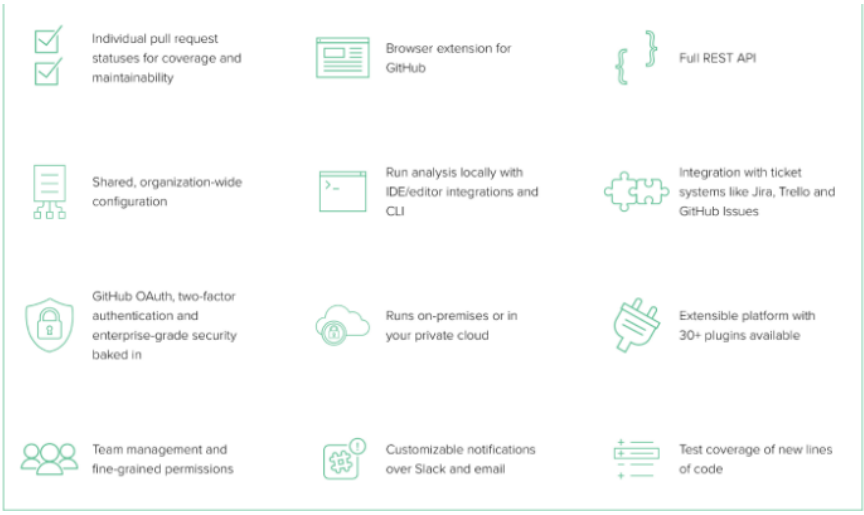
Hackystat operates by attaching sensors to development tools which gather raw data and send this to the Hackystat server. Hackystat collects data from both client and server side data. This allows for a more comprehensive analysis of the data. Hackystat also collects data unobtrusively meaning that users are not aware of when data is being collected. This lead to a distaste among developers as many did not want data being collected without them being aware of it. Hackystat allows data to be collected on a minute by minute basis and allows both personal and collaborative efforts to be tracked. The nature of this collection allows managers to have a deep understanding of the work that is being done, however, many developers were uncomfortable with the accessibility the platform gave others into their work. (Johnson, 2013)

Hackystat took on the problem that PSP had by trying to automate the input and prevent distractions of development. It fell into problems when developers disliked the sheer volume of data that was being collected by the server.

IV. CODE CLIMATE

Code Climate is a specialized software that allows for the analysis of the software engineering process. On their website, they state “Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous, and ubiquitous.” (Code Climate, 2018)

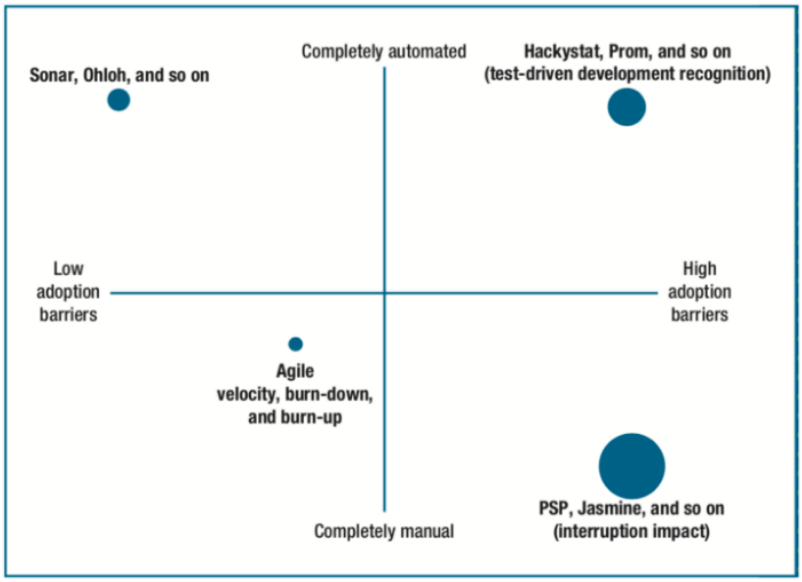
Code Climate allows for code to be reviewed to ensure that it is easily understood, it doesn't repeat itself, and it can be modified and reused with ease. It allows organizations to properly analyse their code quality through full test coverage and by reducing code complexity. It also tests code for maintainability, an important measure of product as discussed above. Code Climate allows companies to have an analysis of their software engineering product and process metrics. Code Climate ensures that any updates to code add increased value and are adequately tested. (Code Climate, 2018)



The image shows some of the services that Code Climate offer to customers such as Intercom, Salesforce and Heroku.

V. OTHER PLATFORMS

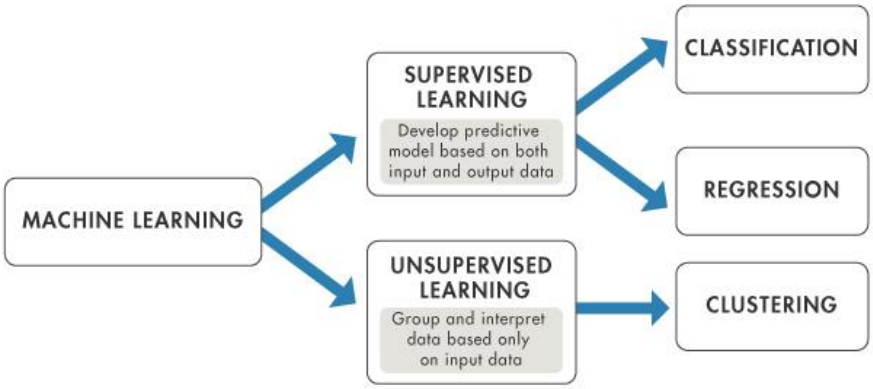
In the past few years, a number of specialized firms have developed in the area of software analytics as the importance of measuring the development process grows. Aside from the companies I discussed, there are a number of other companies offering similar products. They are Codacy, Codebeat, Sonas, DevCreek, CAST and many others. The below diagram illustrates some of the differences between the current platforms available on the market.



(Johnson, 2013)

6. ALGORITHMIC APPROACHES

As the nature of analysis has changed over the past 10 years from manual to a more automated approach. The rise of machine learning has enabled computers to analyse information and make decisions themselves. This has greatly speeded up the decision making process. In this section I will look at some types of machine learning algorithms that enable developers to automate their analysis of the software engineering process.



A. TYPES OF MACHINE LEARNING ALGORITHMS

There are two main subsections to machine learning algorithms: supervised and unsupervised.

I. SUPERVISED LEARNING

Supervised learning occurs when there is an input and an output variable, and an algorithm is used to map the input to the output, ie. $Y = f(x)$ where y is the output variable and x is the input variable. The belief underlying supervised learning is that, after some trial and error, the output value, y, will be so well approximated that it will be able to compute output not only for the current input values, but new input values as well.

Supervised learning gets its name from the fact that we know the answer expected from the original input data. As a result we are able to monitor that the machine algorithm is predicting correctly and if not changes can be made to fix it. (Brownlee, 2016)

II. UNSUPERVISED LEARNING

Unsupervised learning is where an input variable has no set output result. As a result, unsupervised learning involves modeling the underlying dataset to learn more about the data. This process can't be monitored as there are no correct or incorrect results. The data must be analysed by an appropriate algorithm but there is no guarantee what structure the data will look like. (Brownlee, 2016)

B. SUPERVISED LEARNING ALGORITHMS

I. LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis (LDA) is a classification technique that allows you to place your multivariate data into classification groups. LDA is the process of defining a classification decision bound, say 0.25, and classifying multiple instances of this data by the boundary. This method is widely used with machine learning and artificial intelligence. (Houlding, 2018)

II. K-NEAREST NEIGHBORS

K-Nearest Neighbors is a non-parametric method of classification. This algorithm makes no assumptions about the spread of data within each class. We implement this algorithm by receiving a new point we are looking to make a prediction for and classifying this point using the data we already have. We will have to decide the number or K nearest neighbors that will help classify the point, ie. K = 3, and then the machine will choose the variable which dominates the choice of the 3 nearest observations to the new point. (Houlding, 2018)

C. UNSUPERVISED LEARNING ALGORITHMS

I. K-MEANS CLUSTERING

Clustering algorithms are very useful in data analysis as any data can be divided into a set of clusters based on their characteristics. The first step in the k-means is to specify the number of k clusters you want to analyse and then randomly assign a data point to each cluster. Once the clusters are created, you must compute the central data point in each cluster and then re-assign the data points to each cluster centroid it is closest to. You then recalculate the central data point in each cluster and re-assign the data points. This process is repeated until there are no improvements possible. (Houlding, 2018)

D. CONCLUSION

Machine Learning has become much more accessible over the past number of years. All the algorithms I have discussed are computational algorithms and do not require a computer to use them. However, automating these algorithms will enable faster analysis and quicker decision making. These algorithms are not suited however to softer data, such as social interaction, and it is for this reason that machines are not taking over analysis as a whole. There is still a human element to algorithmic analysis but the growth of machine learning has opened the door to a range of new possibilities.

7. ETHICS

The ethics and legality of collecting and analyzing data are complex at best. I will discuss the laws surrounding this and then look at the different measures being taken within the software engineering industry to protect employees.

A. LEGAL ASPECT

In May 2018, the EU released the General Data Protection Regulation (GDPR) to replace the previous Data Protection Act (DPA). GDPR has greatly strengthened the protections surrounding data collection. All companies must now require consent for holding any person's data and record of this consent must be held by a company. Companies are now required to be much more transparent about the data they hold, how it is stored and how it is used. This applies to any company processing data of customers who live in Europe, which has a major implication for a number of companies. Any company found to be in breach of GDPR can be fined 4% of their revenue or €20 million. As a result of this regulation, any firm looking to measure their software engineering process must do so in a way which complies with current regulation. (Citizens Information, 2018)

B. DATA COLLECTION

There are a number of ethical concerns surrounding data collection. These come about as a result of unobtrusive data collection. As spoken about earlier in regards to Hackystat, developers are often uncomfortable with their data being collected without their knowledge. The type of fine-grained data collection that Hackystat engaged in made a number of engineers very uncomfortable with the transparency it gave management into how they work. It is clear that the employees in a company must have access to the way in which their information is collected in order to avoid further conflict. It is important that employers keep in line with GDPR to avoid the stigma associated with data collection. (Johnson, 2013)

Another important aspect of data collection is the type of data they are collecting. While an employer may have access to an employee's life through a variety of technologies, it is important that they are only collecting data which is directly linked to the work they are doing. It is important that employers are only collecting data which will help improve business decisions and help support employees. In relation to this report, all data collected should directly help to improve the software engineering process.

C. DATA USAGE

There are a number of ethics surrounding how data is analysed. Most of this is concerned with keeping analysis impartial and fair. For example, rewarding a developer for writing the most lines of code will only encourage developers to write as many lines as they can regardless of their value. It is important that data is analysed in such a way that it will help motivate workers.

It is also important that data is not used in a discriminatory manager. For example, data collected hints towards that someone applying for a job is of a certain ethnic background. This information should not be used to stop them from having an opportunity. There is a very fine ethical line when using data to make decisions.

In regards to software engineering, it is very important that measuring a software engineer's performance doesn't stifle their creative process. As we discussed throughout the course of this

report, there are a number of processes used to develop software. Often these are very individual. It is important when management are analyzing data and looking at the productivity of one developer against another they keep in mind the individuality of the process. It is clear that the main use of data in the software industry should be to help the improvement of the individual process to increase productivity rather than pitting people against each other.

8. CONCLUSION

In brief, this report has outlined the way in which the software engineering process has and is measured. We have looked at measurable data, platforms available, algorithmic approaches and the ethics surrounding this type of analysis. There are a number of ways available to measure the progress of software engineers but this measurement must be carried out in such a way that internal conflict is not created. The main focus of any companies involved in the measurement of the software engineering should be how the analysis will help their employers. It is vital that this is kept transparent to avoid any issues down the line.

9. BIBLIOGRAPHY

A. Sillitti, A. J. (2003). *Collecting, integrating and analyzing software metrics and personal software process data.* Proceedings of the 29th Euromicro Conference: IEEE.

Beck, K. (2000). *Extreme Programming Explained.* Addison-Wesley.

Brownlee, J. (2016, March 16). *Supervised and Unsupervised Machine Learning Algorithms*. Retrieved from Machine Learning Mystery: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. Accessed 12/11/2018.

Citizens Information. (2018, September 5). *Overview of the General Data Protection Regulation (GDPR)*. Retrieved from Citizens Information: http://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_of_general_data_protection_regulation.html

Code Climate. (2018). *About Us: Code Climate*. Retrieved from Code Climate: https://codeclimate.com/about/

Gabry, O. E. (2017, March 17). *Software Engineering — Software Process and Software Process Models*. Retrieved from OmarElGabry's Blog: https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc. Accessed 12/11/2018.

Houlding, B. (2018, November). *ST3011: Notes .* Retrieved from SCSS TCD: https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides8.pdf

Accessed 12/11/2018.

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

Kan, S. (2003). *Metrics and Models in Software Quality Engineering.* Addison-Wesley Professional.

Littlefield, A. (2016, Septemeber 12). *The Beginner's Guide To Scrum And Agile Project Management*. Retrieved from Trello: https://blog.trello.com/beginners-guide-scrum-and-agile-project-management. Accessed 12/11/2018.

Rouse, M. (2016, November). *Software Engineering Definition*. Retrieved from WhatIs: https://what.is.techtarget.com/definition/software-engineering

VersionOne. (2018). *12th Annual State of Agile Report.*

Xenos, M., & Stavrinoudis, D. (2008). *Comparing internal and external software quality measurements.* Greece: ResearchGate.