Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33012 Software Engineering

Measuring Software Engineering Report

Name: Emer Murphy

Student Number: 18340180

Introduction:

It's beneficial for an organisation to be able to measure software engineering activity. It allows companies to track progress, determine resource allocation, motivate the workers and reward high performers. However it is difficult to measure software engineering activity accurately and effectively. This report discusses:

1. How software engineering can be measured
2. The platforms that can be used to gather and perform calculations on data
3. The various kinds of computation that could be done over software engineering data
4. The ethical and legal issues surrounding the processing of this kind of personal data.

Software engineering process:

The metrics discussed in this report measure different phases of the software engineering process. The Software Development Life Cycle (SDLC) done correctly is the way in which high quality software is developed in the most efficient way possible. It usually has 6 steps: requirement analysis, planning, software design such as architectural design, software development, testing and deployment.

The **requirement analysis** is about identifying what the application is meant to do by communicating with stakeholders, developers, industry experts and customers. It also analyzes the resources that will be needed and the best use of these resources to fulfill the requirements.The Requirement Specification document puts guidelines in place to prevent the project from moving away from its original purpose.

The **design** step is about using the knowledge from the Requirement Specification document to produce the Design Specification document. Prototyping is used to show the basic plan of how the application will look and work, and allows stakeholders to give feedback. It's important to involve stakeholders in the design phase because it's much more economical to change the design in the design phase than in the next phases.

Software development is the stage where the building and coding of the application is done. Using the Design Specification document, guidelines are put in place and the work is usually divided up between the team of programmers. If the previous phases are done correctly developers know what they should build and why.

The **testing** phase is about checking to see if the application meets the requirements and works as expected. The testing phase can often be overlooked, however it is a very important phase which can fix bugs, lower cost and save time.
When the previously discussed phases are completed it is time to **deploy** the code to allow users to use the application. If there are any bugs or issues they are fixed before releasing the application.

Technology is always being updated so the **maintenance** phase is about keeping the application's software up to date with the new technologies, new stakeholder requirements and fixing bugs that weren't caught during the development process.

1. **How software engineering can be measure**

There are many different ways to measure software engineering productivity however it is difficult to measure it accurately and fairly. When measuring software engineering, usually productivity is defined as increased output that results in increased revenue. There are many different ways of measuring software engineering for example by the number of commits or the complexity of the code measured using the structure of the code.

Counting the **lines of code** is a metric used to measure the productivity of a programmer which is easy to implement. However, it is not a good way of measuring productivity as generally the less code needed to complete a task the better, as less code is easier to maintain and improves functionality. Measuring the developer's productivity by the number of lines often encourages programmers to write longer and less efficient code to complete a task with more whitespace and comments. It

encourages noise rather than productivity. Also different programming languages require a different number of lines to complete a task, for example CSS tends to be less compact then Kotlin meaning CSS programmers would appear to be more productive than Kotlin programmers if the number of lines of code are the metric used for measuring a programmer's productivity. Counting the lines of code also doesn't take into account the experience of a developer or complexity involved in performing the task.

The **number of commits** is another way of measuring the productivity of a software engineer. A commit saves the latest changes in the source code to the repository. Counting the number of commits to measure the productivity of a software engineer is more effective than counting the number of lines of code because it doesn't encourage as much noise. However commits can be different lengths for example one programmer could commit every time they change a single line of code or simply add a comment, whereas another developer could commit every time they solve a problem which takes experience and time. Counting the number of commits can reward the lazy programmers that commit more often to play the system instead of the programmers with lots of experience, who focus on writing efficient, good code that completes tasks.

Using **shipping velocity** (issues resolved) as a metric for software development is more effective than lines of code or commits because it encourages problems to be solved and tasks completed. However the problem is that not all tasks involve the same amount of complexity and time to solve. Some issues can take 5 minutes to

complete and other more difficult tasks can take the same engineer 2 weeks to complete. Shipping velocity as a metric is not effective when only looking at the number of tasks completed without looking at the complexity and experience involved. It can be more effective when the issues resolved are normalised for issue complexity although this is difficult to measure.

2. Platforms on which one can gather data and perform calculations

There are many different platforms for gathering and analysing data. This report will focus on three of these: The Personal Software Process, Hackystack and Github.

The Personal Software Process (PSP) provides developers with a disciplined personal system for doing software development. The PSP has four levels:

1. Level 1 includes personal measurement, basic size measurements and coding standards.
2. Level 2 includes the planning of time and scheduling
3. Level 3 introduces the personal quality management, design and code reviews
4. Level 4 is for the personal process evolution.

The different levels improve the developers' approximation and planning skills and can reduce the number of bugs in their project. However PSP involves manual data collection meaning it is very time consuming and requires commitment.

Hackystat is a framework with automatic data collection. Software ‘sensors’ are attached to the engineer's tools which collect and send the data to a web server called the Hackystat SensorBase. Hackystats collects data in real time to track

developers as they work on code. It allows the developer to analyse documentation and is lightweight as the data is stored as XML files in the SensorBase repository. The repository can be queried to form higher level abstractions of the data. Hackystat, although being revolutionary for its time, was not popular with developers because many felt it crossed the line in terms of privacy.

Github is a code hosting platform for version control and collaboration. Github has more than 100 million repositories thus it contains an abundance of data about the software development process. It is very popular and widely used. Github can be used to count the number of commits by a developer and this can infer some knowledge about the programmer. However as already discussed in this report the number of commits does not provide much information about the developers abilities. The commits can describe which developers have worked on which parts of the code therefore identify who would have the best understanding of that particular code thus is more valuable to the team.

3. Computation that can be done over software engineering data

There are various kinds of computation that can be done over software engineering data in order to profile the performance of software engineers, such as machine learning approaches and Halstrad's complexity metrics.

As discussed above it's simple to measure the number of commits or the number of tasks completed by a programmer, however the problem is it is difficult to measure the complexity of the individual tasks. **Halstrad's complexity metrics** are used to

measure complexity. A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands. The indicators defined to check the complexity of a module are:

Parameter	Meaning
n1	Number of unique operators
n2	Number of unique operands
N1	Number of total occurrence of operators
N2	Number of total occurrence of operands

The metric report shows:

Metric	Meaning	Mathematical Representation
n	Vocabulary	n1+n2
N	Size	N1+N2
V	Volume	Length * Log_2 Vocabulary
D	Difficulty	(n1/2)*(N1/n2)
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000

T	Testing Time	Time = Efforts/18

Halstead metrics are simple to calculate and can be used to predict the rate of error and maintenance effort. However the metrics were developed in the 1970s and hence are now a little outdated. It can be inaccurate for some of the new programming languages that are more efficient at writing programs. It can be a useful tool but similar to the methods discussed above it should be taken in context.

The two **machine learning** approaches that are discussed are **supervised machine learning** and **unsupervised machine learning**.

Supervised learning algorithms use labeled training datasets to classify data points. The algorithms can then assess its accuracy using the labeled data. The **K-nearest neighbours** algorithm is a supervised learning method which can be applied to large datasets as a classification technique. KNN classification is a non-parametric method of assigning group membership to unlabelled observations. KNN makes no underlying assumptions regarding the spread of data within each class. To utilise this classification method, there must exist a subset of the data which is labelled with respect to the observation's group membership. The objective of applying KNN to the dataset is to utilise the information related to the labelled data set in order to classify unlabelled data points and assign them group membership. The labelled subset is split into three; Training Set, Test Set and Validation Set. The information obtained from the labelled points in the training set is used to classify unlabelled observations. The test set is then used to find the appropriate value of k. This is done by testing a range of values of k and finding the appropriate value

based on minimizing the misclassification rate. The validation set is then used to find the misclassification or error rate based on the chosen value of k. An alternative method for ascertaining an appropriate value for k is cross-validation. This method involves the removal of a single datapoint from the dataset and determining if the point would still be correctly classified with the knowledge of all the other observation's labels. This is done for each observation in the dataset for all values of k. The value for k with the highest classification rate is subsequently selected.

Unsupervised machine learning uses machine learning algorithms to analyse and cluster unlabeled datasets. **Hierarchical clustering analysis (HCA)** is an unsupervised learning method which aims to identify groups of observations such that the observations within the groups are similar and the observations within the other groups are dissimilar. HCA begins with each of the observations presented as a separate group. The two closest groups are then joined together to form a single group. This continues until all of the observations in the data set are joined to form one group. There are many dissimilarity measures which can be used in HCA. Some of which include Euclidean, Manhattan or Maximum. By determining the dissimilarity of the observations, one can determine which observations are the least dissimilar or conversely, the most similar. When joining groups, one must consider the linkage methods being used. Take two groups of observations A and B. Some of these methods include single linkage which joins the groups based on the minimum distance between an observation in group A to the group B. Complete linkage calculates dissimilarity based on the maximum distance between group A and B. Average linkage takes the average distance between each of the observations in group A and B to ascertain dissimilarity between A and B. When visualising the

results of hierarchical clustering, one could produce a dendrogram. Dendrogram presents a tree-like structure whereby the lower the groups are connected on the diagram, the closer they are together. The vertical scale which accompanies the dendrogram shows the distance at which the groups were joined.
When deciding how many groups exist within the dataset, there is no fixed rule. A proposed rule of thumb is to take a line traversing the dendrogram at the mean height plus 3 standard deviations of the height. From there, the groups can be observed.

To assess the performance of the clustering algorithm, it can be useful to apply it to artificial data with known structures and groups. Therefore, the groups and structures identified by the algorithm can be verified or compared.

4. Ethics of processing of personal data

The ethics, legal and moral questions surrounding the processing of personal data are ambiguous and difficult to answer. Collecting and analyzing data can be beneficial in many different ways from personalised newsfeeds on your social media page to advancing research for cures in healthcare. The Data Protection Regulation was put in place to protect people's privacy, however this was in 2018 and technology developed quickly. Laws and regulations are not being put in place at the same rate as the rate of advancements in technology. This puts the regulation in the hands of the companies developing the technologies, the same companies that are making money from the data they are collecting putting the individuals privacy at risk.

For rich and powerful tech companies such as Facebook, measuring the software engineering productivity of its employees is about increasing revenue and squeezing the most amount of money as possible out of its employees for the gain of the people on top who are already very rich. In today's capitalist society in many cases profit and power is put before peoples rights and wellbeing, for example Cambridge Analytica was implicated in a massive data breach, the privacy of millions of facebook users was compromised while benefiting the millionaires and billionaire at the top of these companies, Mark Zuckerberg the CEO of Facebook is worth over 114 billion USD.

As discussed in this report, it's very difficult to use data to accurately profile the performance of software engineers. If the collected data is inaccurate and unreliable this leads to incorrect insight and faulty predictions. Instead of potentially compromising employees' privacy, creating a work environment where employees are happy is a much better way of improving productivity. When employees feel fulfilled and happy in work they are more engaged, creative and offer more ideas. The rights and privacy of employees should be the number one priority of any organisation, if there are doubts then the employee should be priorities.

Conclusion:

In conclusion, this report discusses how software engineering can be measured, by the number of commits or the complexity of the code measured using the structure of the code, the platforms that can be used to gather and perform calculations on data, The Personal Software Process, Hackystack and Github, the various kinds of

computation that could be done over software engineering data, such as machine learning approaches and Halstrad's complexity metrics, and the ethical and legal issues surrounding the processing of this kind of personal data. As discussed in the report it's difficult to accurately profile the performance of the developer, in addition the ethics around the processing of personal data are ambiguous thus employees' privacy can be put at risk which is crossing the line.

Sources

https://phoenixnap.com/blog/software-development-life-cycle

https://www.geeksforgeeks.org/software-measurement-and-metrics/

https://www.getclockwise.com/blog/measure-productivity-development

https://www.pluralsight.com/blog/software-development/5-software-delivery-metrics

https://www.geeksforgeeks.org/personal-software-process-psp/

https://hackystat.github.io/

https://towardsdatascience.com/githubs-path-to-128m-public-repositories-f6f656ab56b1#:~:text=There%20are%20over%20128%20million%20public%20repositories%20on%20GitHub.

https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides5.pdf

https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides7.pdf

https://ieeexplore.ieee.org/abstract/document/7474681

https://product-help.schneider-electric.com/Machine%20Expert/V2.0/en/CodeAnly/CodeAnly/D-SE-0095969.html#:~:text=The%20Halstead%20complexity%20metric%20is,to%20a%20sequence%20of%20tokens.

https://www.dataprotection.ie/en/who-we-are/data-protection-legislation

https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html

https://www.businessinsider.com/facebook-mark-zuckerberg-net-worth-priscilla-chan-2017-10?r=US&IR=T