MEASURING SOFTWARE ENGINEERING ESSAY

HOLLY MCEVOY 19334663

**Table of Contents**

**INTRODUCTION2**

**MEASUREABLE DATA2**

LINES OF CODE(LOC)………………………………………………………………………………2

CONSISTENCY OF CODE………………………………………………………………………….3

TESTING…………………………………………………………………………………………………3

DEFECTS AND BUGS……………………………………………………………………………….3

INFLUENCE MONITORING………………………………………………………………………4

**AVAILABLE PLATFORMS4**

GITPRIME/FLOW……………………………………………………………………………………4

HACKYSTAT……………………………………….…………………………………………………5

PERSONAL SOFTWARE PROCESS(PSP)…………………….…………………………….5

VELOCITY 2.0……………………………………………………………………………………….5

**ALGORITHMIC APPROACH6**

MACHINE LEARNING ANALYSIS………………………………………………………………6

ALGORITHMIC MODELLING………………………………………………………………….7

**ETHICS8**

**CONCLUSION10**

**REFERENCES…………………………………………………………………………………………....10**

**INTRODUCTION:**

In this report I will be accessing how software engineering can be measured using calculable data and how this data can be accessed. This can be a difficult process. There is a huge amount of data that can be gathered on individual software engineers. This report will be split into four sections, each section containing an in depth investigate how we measure software engineering. The First section will be on Measurable data and how we measure engineering activity. The second section will tackle the platforms that are available to us to measure software engineering. The third section will be on algorithmic approaches and how we can compute the data that we collect and finally the last section will discuss the ethics surrounding use of this personal data and the opinions I have formed based on my research.

We need to take a lot into account when measuring software engineering for example: Software Specification, Software development, Software Validation and Software Evolution. Ian Sommerville said that measuring software engineering includes those four fundamental activities, we need to investigate why we would want to measure this process and the steps involved. For many businesses measuring and accessing your software engineering process is an incentive to increase productivity as it can help to identify and communicate any issues to management. They can use this process to help communicate the status of projects, monitor, and improve workflow and to address any issues. This helps them to reduce costs and increase efficiency.

**MEASURABLE DATA:**

Since the beginning of software engineering, people have always tried to measure its process, some measurements are still around, even though they were created in the 1900’s and earlier. In this section I will discuss the history of the assessment of data and how methods have been developed upon to get more complex and subjective measurements.

* **LINES OF CODE (LOC):**

Lines of code is a commonly used metric that is often called kilo lines of code (KLOC). A major problem with LOC is that if it does not coexist with another data type it is basically useless and an extremely loosely based productivity measure. This has led to many metrics being developed that we still use today e.g., time per KLOC, defects per KLOC and errors per KLOC. These are extremely easy to measure methods but much like everything in life they do have some kinks and issues when we assess them in depth for example a fast programmer who has a low time per KLOC may make many errors or defects. We often see “effective” or “source” lines of code when we talk about measuring software engineering, this basically means how many lines are added or removed from the production version of the product. This is extremely useful as it only takes the relevant work done by the software engineer. Although again this does have its problems as code with many effective lines might cause more bugs or defects than code with less effective lines, yet it is still a huge improvement on the standard LOC measurements.

Since LOC is platform, specific this can cause quite a big problem as amount of code differs from one language to another based on their level, hence we would be unable to measure the effectiveness of their lines of code and compare them. Yet when we do have an effective measure of KLOC we can then access other factors based off these e.g., cost per effective KLOC. This can have a good general measure for a project as we can estimate how many KLOC the finished project should take and figure out the cost of the project based on this. Of course, like most things, this will be an estimate and there is still room for error, however it is still very effective and with increased experience and usage can lead to more accurate projections.

* **CONSISTENCY OF CODE:**

Consistency of code measure the performance of an engineer over time. We can use LOC to measure the quality of the code produced with respect to the time taken if we were to use LOC as a function of time. This can be useful, but it does not consider the difficulty of the work being done by the software engineer.

Commit frequency can also be used to see a developer consistency of code. We can look at the number of commits in a day as well as the frequency of the commits over time and the pattern in which the developer commits can be used to analyse performance, for example we can see if the developer’s performance changes throughout the course of the week.

We can also use the metric “churn rate”, this is the percentage of the developer’s code that is an edit to their recent work i.e., how many times they rewrite their code and how much of it they must rewrite. If a developer has a high churn rate this can indicate a poor code quality where if they have a low churn rate this can indicate high quality code. Churn rate can also show how consistent and clear the project specification is as a changing specification can lead to a higher churn rate.

* **TESTING:**

Testing software is a commonly measured process, the common metric here being the code coverage i.e., the percentage of code being covered, this is found by an automatic test. This is a good metric as it helps prevent failures occurring, it also allows errors to be easily found and fixed. Often the business will set up a “safety net” in which the code needs to pass a certain percentage before it can be pushed in order not to cause failures.

* **DEFECTS AND BUGS:**

Measuring defects and bugs in code can be very challenging. Fixing bugs can often be more of an involved process than writing the code. From a numerical standpoint the developer with less defects per effective KLOC would be preferable however the severity of the bugs must also be considered, this would be up to management to judge as it is always impossible to measure this as one hundred bugs in a simple code can be fixed easily whereas one bug in a difficult code can be extremely hard to find and fix which can create errors for the whole business. The process of removing bugs requires an understanding of the problem code and how it interacts with other parts of the software. The longer and more difficult it is to fix a bug the harder it is to measure performance as the code quality would be quite low until the bug is fixed. Therefore, it is extremely hard to get a good data measurement when bugs and defects are involved especially if they lead to a failure.

* **INFLUENCE MONITORING:**

Influence monitoring measures the effect and importance a certain employee has based on several metrics: how connected the employee is, the size of their immediate circle, how connected they are and how business critical they are. Their influence is then measured based on these four metrics to see how valuable this software engineer is to the company. This would be more of an environmental factor i.e., the working environment can be measured and used to access its impact on its software developers.

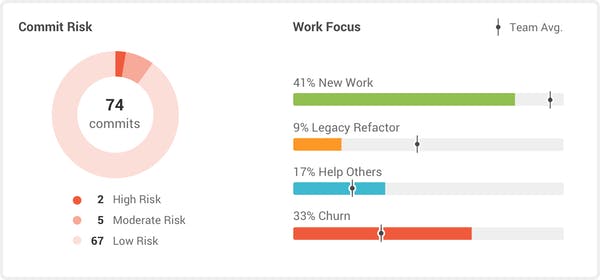
**AVAILABLE PLATFORMS:**

There is a vast number of available platforms that help us to measure and analysis software engineers and their process. These platforms can manually or automatically scrape metrics from VCS depending on what is required. In this section I will talk about a few of them

* **GITPRIME/FLOW:**

GitPrime would fall under the category of a cloud service, which is a service which is offered by companies that perform analyses. It is one of the largest companies in performance measurement and has clients such as Tesla and Disney. It analyses data gathered from the version control system being used by the company, some examples of these control systems are GitHub, BitBucket and SVN. It is a very easy to use platform as almost everyone in a technology-based industry has a least one version of git as their VCS. GitPrime record and analyses data in many different metrics, this includes personal metrics such as number and times of commits which is handy for measuring data methods discussed above such as consistency of code. It also takes organised metrics into consideration such as levels of activeness in a team and progress on current projects. This is a huge help to managers being able to identify developers who are not working effectively and why as it looks at analysis rather than actual code. Since May of 2019 GitPrime is now a part of Pluralsight and has been renamed Flow.

Below is an example of a data visual from flow:



* **HACKYSTAT:**

Hackstat is a framework that allows developers to collect and analyse PSP data automatically. It is an open-source framework for collection, analysis, interpretation, annotation, visualization and dissemination of software development process and product data. They claim to “unobtrusively” collect data by attaching sensors to the developers’ tools (such as their ide), which then sends that raw data to a server to interpret. Since the framework is designed to gather data from development tools, it also allows the engineer to analyse work other than software such as documentation. It mainly focuses on individual data and its uses; this structure emphasizes privacy. It is also lightweight as it stores as XML files rather than a backend database.

* **PERSONAL SOFTWARE PROCESS (PSP):**

The personal software process is a structured software development process that is designed to help software engineers better understand and improve their performance by bringing discipline to the way they develop software and tracking their predicted and actual development of the code. There are many variations available, but most involve manual data collection which is analysed and continuously monitored this a substantial effort to collect data, so we have developed platforms to aid us and make this process more effective. An example of this is called PROBE, which uses the estimated size of a system to find an estimated time of development based on past projects.

* **VELOCITY 2.0:**

Velocity is a product by Code Climate, it is quite like GitLab, which also analyses data from a repository and draws information from this. Velocity transforms data from Jira and your DevOps tools into insights that drive engineering operations from stand ups to board meetings. Velocity is aimed at finding process constraints in departments rather than individual developers. They also focus on visualising raw engineering data so it can be understood by people outside the software engineering industry. They use this information to provide a clear view of progress over time.

**ALGORITHMIC APPROACHES:**

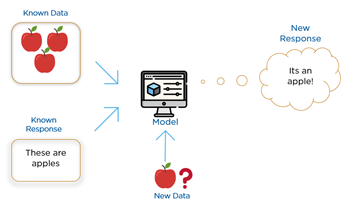
Measuring productivity accurately has been in demand for quite some years now and because of this there has been a numerous number of different algorithms for software developers created in the past few years. Some of these algorithms are extremely useful and easy to use. I will talk about a few of them in this section.

* **MACHINE LEARNING ANALYSIS:**

Machine learning (ML) algorithms are particularly useful in analysis of data in the software analysis process. Complex ML algorithms are often used for fault prediction in large software projects. These methods have proven to be very useful in tackling large problems such as translation, image recognition and self-driving vehicles, there are 3 types of machine learning: Supervised learning, Unsupervised learning, and Reinforcement learning.

Supervised learning is used to describe a mother of training an algorithm by providing it with a large label dataset. This algorithm learns from this and after it has learned from this data the resulting model uses it when asked to analyse other data.  The supervised approach is indeed like human learning under the supervision of a teacher.

Below is an example of how Supervised learning works:



Unsupervised learning involves providing the algorithm with a large amount of unlabelled data and allowing it to discover correlating factors on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

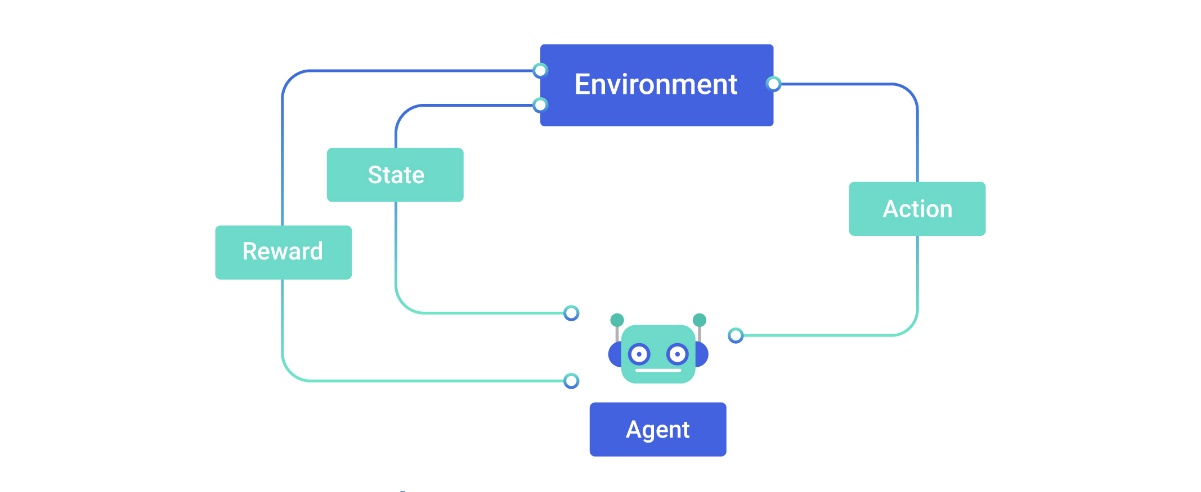
Below is an example of how Unsupervised learning works:

Diagram

Description automatically generated

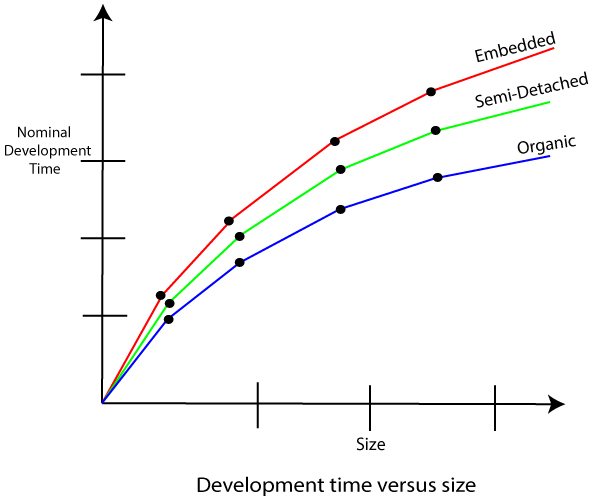
Reinforcement learning is quite like unsupervised learning in the sense that the algorithm is provided with unlabelled data. However, this data is accompanied by positive or negative feedback depending on the output of the algorithm. This is like learning by trial and error. Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves.

Below is an example of how Reinforcement learning works:



* **ALGORITHMIC MODELLING:**

This is the use of relatively easy to implement algorithms to predict variables such as cost. There are many models available but one of the most used for cost is the Constructive Cost Model or COCOMO. It is a regression model based on LOC. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was developed by Barry Boehm. The model is based off two parameters, effort, and schedule. Effort is the amount of labour required to complete a task i.e., people, whereas schedule is the amount of time required to complete the job measured in time i.e., days, weeks.



COCOMO has three increasing accurate models: basic, intermediate, and detailed. Each is useful for different complexities of projects. Boehm also defines three kinds of systems: organic, semi-detached, and embedded of different complexities. There is a different formula for each making it a very versatile model. It is one of the most researched and used models, this makes it extremely effective.

Below is the formula for the estimation of effort for a basic model:

**Basic Model –**

   E= a(KLOC)^b 

  time= c(Effort)^d 

  Person required = Effort/ time 

**ETHICS:**

Ethical concerns over how companies gather and analyse data has increased tenfold over the past decade, especially due the Facebook data breach scandal in 2018, in which millions of Facebook users’ data was collected without their knowledge by Cambridge Analytica in 2010 and was used for political advertising. Our data is constantly being collect and analysed to market us specific products and services. Therefore, we must consider a huge number of ethical concerns when approaching the topic of measuring data. In this section I will discuss a few major concerns and my opinions on them.

Gathering data based on code is of little ethical concern as this data is previously being made available by an engineer. Since this data is most likely provided to the employer it is ethically sound to analysis however this data can be manipulated by the software engineer to appear more productive, which is although unethical can be easily solved. On the other hand, gathering of personal or health data is in my opinion extremely unethical, gathering data such as heart rate can be hugely invasive and provide an extreme insight into someone’s lifestyle and I believe employers should not have access to this data. It gives the employer too much insight into their employee’s lifestyle outside of the workspace i.e., if they are fit or not and can be extremely invasive. To expand on this, I believe employers should not be allowed to record employees. This type of monitoring is not only invasive but creates a hostile atmosphere in the workplace as employees feel as if they constantly need to say the “right” thing instead of honest feedback. It is also up to the employer to store this data correctly. Since 2019 there have been over 2000 data breach across government departments. Statistics from the 15 departments show the Departments of Social Protection, Foreign Affairs and Justice experienced the highest numbers of breaches in recent years. More than 400 breaches have taken place so far this year. With the sensitivity of the data that is accessible it is extremely important for companies to securely protect their employee’s data and to take the necessary steps to prevent mass breaches. This can include careful selecting who has access to this information in the company itself. Data permanence is another concern as many companies store many backups of their data to prevent data loss. However, this does raise several concerns, the main one being that if a company has personal data on an employee and said employee is no longer employed by the company, should the company still be allowed to keep their personal data? I personal believe it is unethical.

There are also many concerns about how measured data can be interpreted, the people interrupting the data may not understand the data or the context of the data. It could also be presented in a misleading way and this error can cost someone their job as the data may not show their actual performance to the correct extent. Should there be measures put in place to prevent this? Such as having the direct superior of the employee analysing their data as they would interact with the employee on a regular basis.

We also must consider the ethical implementation of analysing data. There are many different issues with techniques we use to analyse data with many of them surrounding machine learning. We must ask the question “do we trust a machine to fairly and accurately analysis a person?” Machine learning is known for not being able to guarantee a correct result. There is also a known gender bias in machine learning. While a machine's ability to process large volumes of data may address this in part, if that data is laden with stereotypical concepts of gender, the resulting application of the technology will perpetuate this bias. Gender balance in machine learning is therefore crucial to prevent algorithms from perpetuating gender ideologies that disadvantage women. As a woman in STEM, this is an extremely important ethical concern to me. I must ask the question constantly that if I am being judged by a machine will I be seen as less productive because of my gender? Software engineering is a predominated male-based field, essentially can be seen as a “boys club” and even thought the differences in productivity scores many be only slightly lower it can still have a significant impact on how woman is perceived in the workforce.

**CONCLUSION:**

In conclusion the collection, measuring and analysis of data in software engineering is extremely advanced. There is plenty of measurable data, but it is clear there is no definitive way to measure productivity. We have an abundance of platforms and algorithms to help us predictive and analysis data, but they all have their downfalls and faults. Confident Software engineers are a valuable resource to have, they are a need for them in the current market. That is why companies should be careful how they handle their employee’s data, if a software engineer feels as if their data is being misused or if they are being unfairly analysed, they can easily find another place of employment. This luxury is not very common in today’s world. Therefore, employers should be more concerned about their employee’s happiness with the company to increase productivity rather than their data. They should work with a computational platform they feel happy with, many use FLOW/GitPrime which is also my favourite. We must continue to try and ethically use data to provide a safe and comfortable environment for everyone.

**REFERENCES:**

<https://en.wikipedia.org/wiki/Source_lines_of_code>

<https://www.softwaretestinghelp.com/software-test-metrics-and-measurements/>

<https://blog.devgenius.io/why-code-consistency-is-important-9d95bdebcef4>

<https://www.scitepress.org/papers/2017/63144/63144.pdf>

<https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2425392>

<https://stackify.com/measuring-software-development-productivity/>

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

<https://www.pluralsight.com/product/flow>

<https://hackystat.github.io/>

<https://en.wikipedia.org/wiki/Personal_software_process>

<https://codeclimate.com/velocity/>

<https://www.dummies.com/programming/big-data/data-science/3-types-machine-learning/>

<https://www.geeksforgeeks.org/software-engineering-cocomo-model/>

<https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal>

<https://www.irishtimes.com/news/politics/state-departments-hit-by-2-000-plus-data-breaches-since-2019-1.4577281>

<https://dl.acm.org/doi/abs/10.1145/3195570.3195580>