Measuring Software Engineering Report

CSU33012

Matthew Thompson #19334477

Introduction

There has always been some amount of debate surrounding the tracking and measuring of software engineering productivity. This is due to the fact that software engineering is not a simple development process and does not necessarily always have a tangible output. When someone employs a form of monitoring and attempts to collect data about productivity it can also be easy for a software engineer to game this metric and attempt to only achieve a positive result surrounding that one condition. Measuring software engineering is complex but it is certainly possible with the technology that exists in our industry today. In this report I will look at some of the data which can be monitored to track progress and how this data can then be used to paint a picture of the productivity that is being achieved.

Nowaday employers and individuals can employ a vast array of methods to measure the software engineering process and its progress.These can range from measuring factors that are completely unrelated to the amount of code or the time that has been placed into its development and to using different algorithms and methods to analyse these tangible factors.

To start with we will look at measurements drawn from Factors that are not directly measured.

Surveying Employee's Happiness

According to a study conducted by the University of Oxford (https://www.ox.ac.uk/news/2019-10-24-happy-workers-are-13-more-productive) workers are 13% more productive when they are happy with the workplace they are in. As measuring something like software engineering is a difficult task to achieve, this method will allow a

team lead to gain insight into how the engineering process is currently going and if those in the team are likely to be more productive.

Analysing workplace environment

Looking at the current environment that a software engineering team is working in can tell a great deal about the potential productivity that they may achieve. According to a study t on Science Direct the amount and lighting and its sources can have a large impact on the productivity and performance of a worker (https://www.sciencedirect.com/science/article/pii/S1877705811029730). If an office is poorly lit and at a cold temperature then the productivity of a worker may be less than if they were in a well lit office and with a more comfortable temperature.

Next we will look at the way in which the software engineering process can be measured by looking at the code or output of what a software engineer has produced.

Quantity of Code produced

One obvious metric to measure the software engineering process is to measure the quantity of code written. This is sometimes referred to as Lines of Code or LOC. In most instances it will be possible to quantify the amount of actual machine runnable code, this will ensure that a glancing eye is not misled when looking at the project itself if there is large amounts of white space or comments. Another method is to count the number of Logical Lines of Code. This method counts the number of 'logical statements' that exist in a project. This method may give a clearer indication of how much of the code written has a more solidified purpose behind it.

All of these methods can give a valid exploration of the productivity of a software engineer. However, in the majority of instances it may be a misleading metric as it is widely believed

that the best code is the most concise and least complex. (https://www.sciencedirect.com/topics/computer-science/code-metrics)

Time Dedicated to the project

Another easy approach to take is to measure the amount of time that an engineer has worked, or the amount of time that they have spent logged into their computer or coding environment. In most instances though this metric is not a very accurate representation of productivity as not all time spent at work is spent writing code. There have also been numerous studies that show that after a certain amount of time spent working a worker will become less productive with their time. (https://ftp.iza.org/dp8129.pdf)

Errors Existing in Code

Another method that is often used to assess the quality of work produced is to quantify the amount of errors or bugs that exist in a line of code on average. This can indicate the reliability and quality of the code that is being outputted.

If an engineer is working on a piece of code that has a large amount of unresolved bugs in it then finding and crafting a solution to remove these bugs will be a painstaking process which will consume large amounts of time. During this period the engineer will appear to have written no new code however their productivity may have been high during the time. (https://www.businesswire.com/news/home/20210216005484/en/Rollbar-Research-Shows-That-Traditional-Error-Monitoring-Is-Missing-the-Mark)

Commits through version control

If a software engineering team makes use of a version control system then it is possible to track the number of commits made by each individual. Managers can look at whether or not an engineer is active within the project and making a large number of commits to the repository. In some instances however a large number of commits does not tell us if the code quality is very good or if the commit is trivial or of large importance.

Code Churn

Code churn is a method of analysis that measures the quantity of code that is an edit to previous code. Code churn can be used to measure how much of the previous code written is of a high quality. If the code churn of a project has a sudden spike then it may indicate that the previous code was not of a high quality and needs more time and attention. (https://codescene.io/docs/guides/technical/code-churn.html)

Computational Platforms

There a wide variety of tools can be used to obtain relevant information regarding the software engineering process.

Personal Software Process

The Personal Software Process or PSP is a process of software development which has been designed to help software engineers to improve their own performance by tracking the projected development of their code versus the actual development. The main idea is that engineers should be writing code of a good quality to start with instead of relying on tests. More recently it has been shown that utilising the PSP may lead to sizeable benefits for an engineer. One drawback of the PSP is that it requires a large amount of forms to be completed which can be very time consuming and be affected by human error. (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283)

Pluralsight

Pluralsight is a tool which is used to analyse code which has been committed onto a version control system. It is a cloud service which will collect data and then analyse a number of variables concerning the code which has been produced. It will also assess the pull requests and other factors concerning the engineering process such as code churn and efficiency. Managers and engineers will then be able to receive a comprehensive review of all the projects activity on a single dashboard in a visual way.
(https://www.pluralsight.com)

HackyStat

HackyStat is an automated data tracking, data collection and data analysis tool. Software engineers install sensors to their development tools which unobtrusively collect and gather data surrounding the software engineering process and then send it to be analysed on a remote server. HackyStats main selling point is that it collects data in real time throughout the day, in order to reduce any overhead in the collection of data. This allows the tool to monitor whether the engineer is applying good practises to their code output.
(https://hackystat.github.io/)

Code Climate

Code Climate is another automated tool that analyses and collects data regarding code produced. The tool performs an analysis on code coverage and a progress report. After its analysis it will then provide the engineer with recommendations on how to improve the current program.
(https://codeclimate.com)

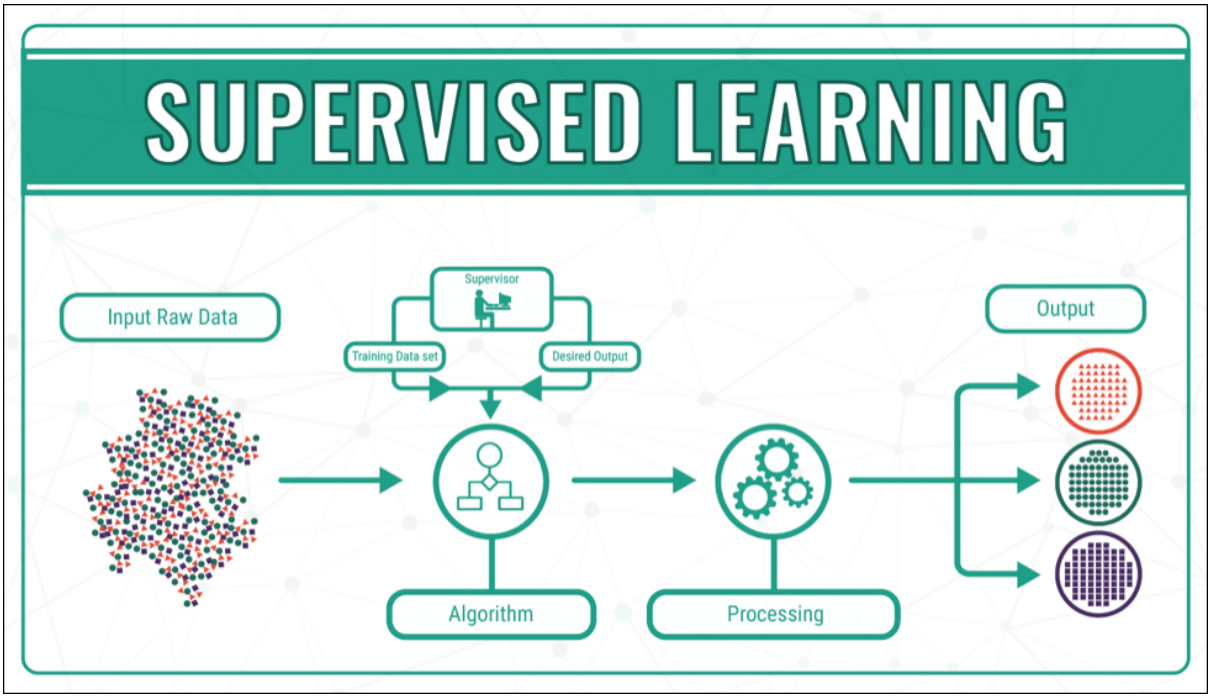
Computation of software engineering data

A number of algorithms can be applied to any data which has been collected regarding the software engineering process. Some of these are machine learning algorithms. We can break these into 3 types of machine learning, unsupervised learning techniques, supervised learning and reinforcement learning.

Supervised learning

The most common computation performed on software engineering is that using supervised learning methods. These methods assume a given structure exists in the data. The machine can be trained by receiving a set of inputs where the desired output is known. The goal of this is to create a mapping function that when we receive a new set of data, as close an estimate for the output is calculated.

One of the supervised learning algorithms is K-Nearest Neighbours (KNN). KNN will classify data points which are similar into a single test set and will then make estimates of where unclassified points should be placed.

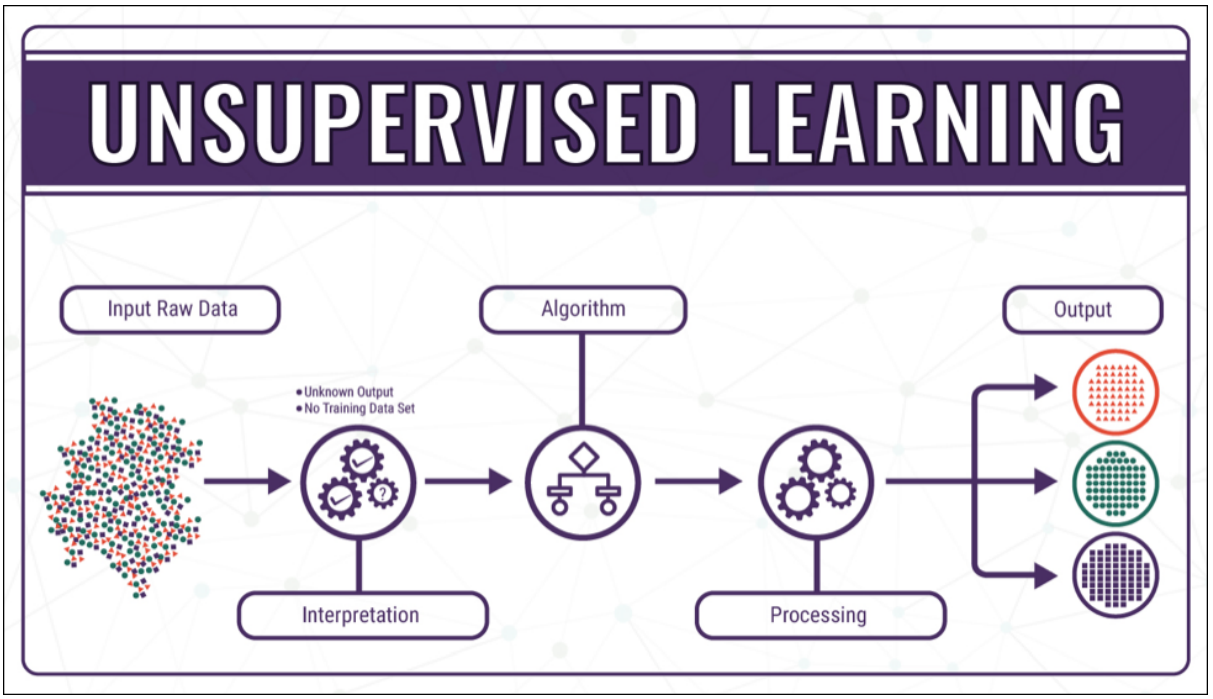


(https://technative.io/why-unsupervised-machine-learning-is-the-future-of-cybersecurity/)

Unsupervised learning

The purpose of unsupervised learning methods is to uncover a hidden structure of the data and to discover any patterns that may exist. The algorithm tries to discern what is being inputted.

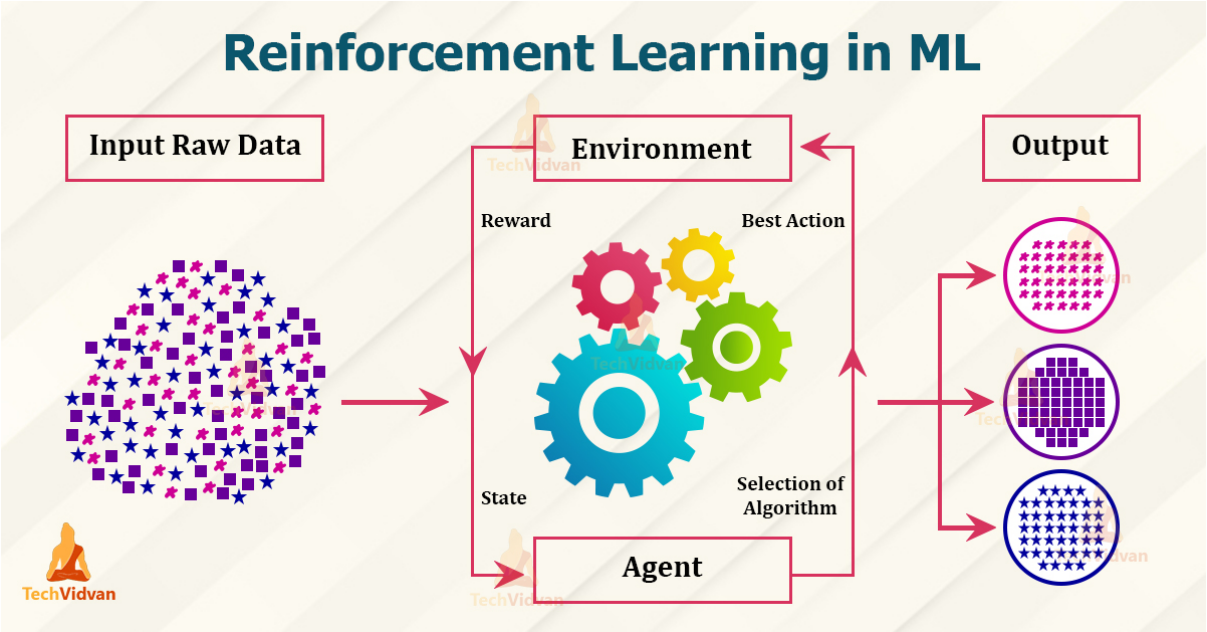
Clustering is the simplest and most important unsupervised learning algorithm. Clusters can be modelled through the use of a similarity measure. Once we have assigned data points to a cluster via an algorithm such as K-means clustering we can analyse its output to identify any patterns in our data.



(https://technative.io/why-unsupervised-machine-learning-is-the-future-of-cybersecurity/)

Reinforcement learning

Reinforcement learning is a machine learning technique which trains a machine to learn models and its behaviour from feedback received through its environment. This behaviour can be adapted over a period of time. Artificial intelligence may face a simulation based scenario and then use a trial and error approach to solve it. If the machine performs an action that is relevant then the programmer will reinforce that behaviour and remove any undesirable behaviour.



(https://techvidvan.com/tutorials/reinforcement-learning/)

Ethical Concerns

It goes without saying that there is a lot of ethical concern around the collection of data and what it will ultimately then be used for and this goes for all areas, not just surrounding software development. Collecting meaningful and useful data often requires intrusive technologies and a lot of monitoring about an individual. This no doubt leads to an uncomfortable feeling for the individual who is being monitored.

The latest regulation which surrounds the collection and usage of data produced by an individual is the General Data Protection Regulation(GDPR) which was released by the European Union in 2018. GDPR requires that any organisations which are collecting data on citizens living in the EU must meet a certain standard. This provides some level of protection towards an individual however I am of the opinion that it is not enough. (https://gdpr.eu/)

A large ethical concern is where does the boundary start and end surrounding the collection of data in a work environment. Information about an employee's health and financial information may be useful for an employer to make an informed business decision but how

many employees would be comfortable with this information being collected and then being used.

The next question then arises as to what this data is then being used for. Is the data being used to help improve the engineers performance or is it being used to inform a decision of whether or not their contract should be terminated. Both of these usages have different implications for the software engineer at hand and without transparent and thorough communication about what the data will be used for it is easy to see why an individual would not want this data to be collected.

Conclusions

I am of the opinion that any employee should not be monitored or tracked as they go about their work. Therefore I think that any performance related monitoring should not occur in the workplace. I am of the view that there should be an element of trust between the team leader and a team member. If this trust does not exist and the more senior member of the team resorts to installing sensors at an employees desk or goes out of their way to survey and analyse every move of a workers day then I would consider this to be a toxic work environment, and one which would not lead to any increased productivity.

Other tools which are used to monitor the performance of a team and the quality of their work are more favourable in my opinion, however they don’t come without their downfalls.