**David Kavanagh – 18327890 – 25/11/2020**

# Measuring performance of software engineering

*"To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics."*

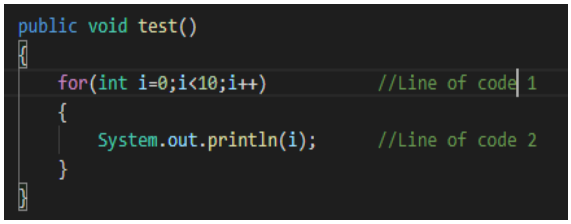
## Introduction:

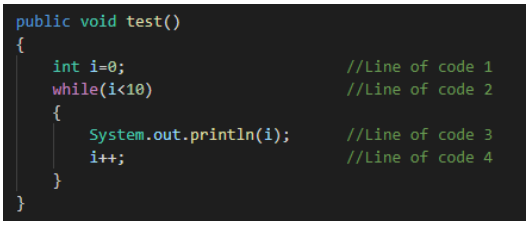
In today's modern world, there is an idea of perfection, and every company attempts to create this idea of perfection. To achieve perfection, every employee must perform to their maximum. The only way for a company to see if their employee is performing at their best is to measure their performance in the workplace. Software engineers are no different. However, I believe there is no "silver bullet" to accurately measure the performance of a software engineer. If several methods are combined, we may be able to have an estimate of their productivity but there is no guaranteed singular method to accurately measure how well a software engineer is performing in a company while considering external factors. In this report, I will describe potential methods used by companies to measure a software engineers' performance, and also explain the potential issues with each method. I will then proceed to discuss algorithmic approaches to measuring performance such as machine learning algorithms and then conclude with the ethics and potential problems of measuring the performance of an employee such as data gathering and analysis.

## Lines of Code:

One of the methods to measure the performance of a software engineer is to measure how many lines of codes (LOC) they write over a specified time period. This is a simple method and easy to understand. We calculate how many LOC the software engineer has committed in a period of time and compare that to their previous score or fellow software engineers. If a software engineer writes a lot of LOC, they are more productive and performing better than someone who writes less.

The main issue with this is that someone who writes more complex and longer code is seen to be performing better than someone who writes simple and concise code in less lines. The argument is that you are valuing quantity over quality which does not make sense if the same quality can be reached in a simpler fashion. Here we can see the first example is easier to understand and better code, but the second example has more lines of code so is seen to be more productive and contributing more to the team. LOC only measures the volume of code [1].




[Less LOC = less productive] [More LOC = more productive]

Another exploit with this potential measure of performance is that an employee can write code in a complex way as the complex implementation will more than likely have more lines of code compared to the simple implementation. This will falsely boost their LOC and make it appear like they are working harder and performing better. The LOC method can be easily exploited and will not produce 100% accurate results.

There is another method to measure lines of code known as logical lines of code (LLOC). LLOC is an improvement on LOC because it measures statements rather than lines of code [2]. Eg: the 'for' statement above is given a value of 3 as there are 3 statements in that one line. This causes the LLOC for both examples to have the same value, but there are still issues with LLOC.

LLOC does not consider external factors that an individual may be doing such as helping less experienced or new members of the team. This software engineer is contributing to the team and is performing well by helping other colleagues, but LLOC will not measure this contribution.

*"The best way to be a 10x developer is to help 5 other developers be 2x developers."* — Eric Elliott [3]

Or another issue with LLOC is that some languages require more lines to perform a task such as Java. When we compare Java against Python for example, we see that Java has more LLOC than a Python version despite them performing the same task. This is why LLOC cannot be the sole factor to determine the productivity of a software engineer.

```python
import random
import math
 # Taking Inputs
lower = int(input("Enter Lower bound:- "))

# Taking Inputs
upper = int(input("Enter Upper bound:- "))

# generating random number between
# the lower and upper
x = random.randint(lower, upper)
print("\n\tYou've only ", round(math.log(upper - lower + 1, 2)),"chances to guess the integer!\n")

 # Initializing the number of guesses.
count = 0

# for calculation of minimum number of
# guesses depends upon range
while count < math.log(upper - lower + 1, 2):
    count += 1

    # taking guessing number as input
    guess = int(input("Guess a number:- "))

    # Condition testing
    if x == guess:
      print("Congratulations you did it in ", count, " try")
      # Once guessed, loop will break
      break
    elif x > guess:
      print("You guessed too small!")
    elif x < guess:
      print("You Guessed too high!")

# If Guessing is more than required guesses,
# shows this output.
if count >= math.log(upper - lower + 1, 2):
   print("\nThe number is %d"%x)
   print("\tBetter Luck Next time!")

# Better to use This source Code on pycharm!
```

[Python – Guessing Game] [4]

```java
public static void
guessingNumberGame()
{
    // Scanner Class
    Scanner sc = new Scanner(System.in);

    // Generate the numbers
    int number = 1 + (int)(100
                           * Math.random());

    // Given K trials
    int K = 5;

    int i, guess;

    System.out.println(
        "A number is chosen"
        + " between 1 to 100."
        + "Guess the number"
        + " within 5 trials.");

    // Interate over K Trials
    for (i = 0; i < K; i++) {

        System.out.println(
            "Guess the number:");

        // Take input for guessing
        guess = sc.nextInt();

        // If the number is guessed
        if (number == guess) {
            System.out.println(
                "Congratulations!"
                + " You guessed the number.");
            break;
        }
        else if (number > guess
                 && i != K - 1) {
            System.out.println(
                "The number is "
                + "greater than " + guess);
        }
        else if (number < guess
                 && i != K - 1) {
            System.out.println(
                "The number is"
                + " less than " + guess);
        }
    }

    if (i == K) {
        System.out.println(
            "You have exhausted"
            + " K trials.");

        System.out.println(
            "The number was " + number);
    }
}

// Driver Code
public static void
main(String arg[])
{

    // Function Call
    guessingNumberGame();
}
```

[Java – Guessing Game] [5]

## Pull requests Count:

We can measure the performance of a software engineer by looking at their pull request count. A pull request allows a software engineer to tell their team members about changes they've pushed to a branch and allows them to discuss the changes made with other members from their team. Pull request count shows who from the team is engaging with the project but it can also be exploited.

The major disadvantage for measuring performance using pull request count is that small changes can be made to the code for each pull request. Pull request count yet again values quantity over quality and thus measuring performance can be difficult and inaccurate. A software engineer could make several pull requests a day by changing the code ever so slightly. Since we are only measuring a software engineer's performance using pull request count, we do not consider the difficulty involved with each pull request which can cause inaccuracies. A software engineer who commits code weekly but to a high standard is seen as performing worse when compared to someone committing code daily but to a low standard. Here we can see this method is flawed. Pull requests count, like LOC, encourages negative behaviours [6].

## Throughput:

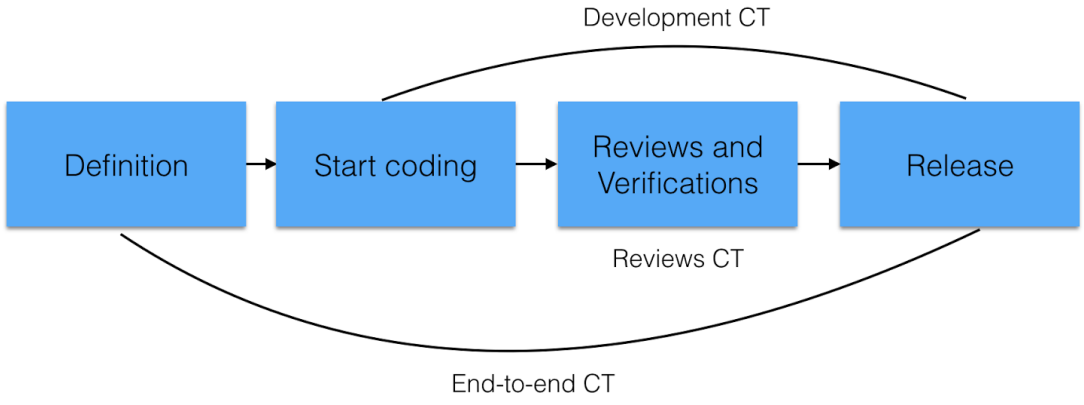
Throughput is a measure of total work output. This includes features, bugs and tasks that are completed and ready to test and ship. After completing one of these, the individual is given a ticket. The higher an individual's ticket count, the more productive they are in theory.

However, like pull request count, this does not consider the difficulty or complexity of achieving a ticket. A small change is seen as equal to possibly changing the underlying data type of a program as they both fix one problem, but we know that one is much harder to achieve compared to the other. The throughput method also may cause a software engineer to rush their code in attempt to gain more tickets by completing them quickly. Yet again we have the problem where a performance measure values quantity over quality which is not ideal.

Throughput does not measure external factors either. For example, we might see an engineer's number of commits decrease and conclude his / her performance is down. However, when we analyse it closer, we see that his commits are down but only because he was pairing with team members, speeding them up, mentoring them and adding long-term value [7].

## Cycle Time:

Cycle Time is the time it for a bug, feature or task to move from one status to another [8]. Within the cycle time of a project, there can be several smaller elements that have their own cycle time as well.




[Cycle Time][9]

Here we can see the development cycle time is a smaller element of the end-to-end cycle time. To measure the cycle time for any element, we record the time for each user story, bug or feature and see how long each one takes to move to

the next stage. If the time period is shorter than expected, the software engineer is performing well. However, if the time period is longer than expected, the software engineer could be performing better.

Similar to LLOC, cycle time does not consider external factors that may cause time spent on one cycle to increase. Factors such as increased complexity of the initial task or possible system changes mid-way through could cause an increase in cycle time. Cycle time can provide useful data when a client asks how long a feature can take to implement. The software engineers can try and compare it to a similar feature they previously implemented and return an accurate estimate for the cycle time of this new feature. Using cycle time to measure a software engineer's performance remains challenging.

## Velocity:

Velocity is a metric for work done in a single sprint [10]. Velocity can be used for planning sprints and measuring team performance. A team assigns a value to each task representing the expected difficulty and time required to complete it. If a team is completing tasks in a quicker time frame than the velocity value indicates, then that team or individual is over performing. This sounds like a solid performance measuring tool, but problems arise quickly with it.

Velocity can lead to an over-estimation of time required for a task as the individual will attempt to achieve tasks in a quicker time frame than the velocity value would suggest increasing the appearance of their performance. This would cause a boosted result for that person's performance and hence does not accurately measure a software engineer's performance.

Another possible issue with velocity as a performance measurement method is that it does not take quality into account. Due to an individual or team wanting to complete a task in a quicker time-frame, it may cause the code to become rushed or cause more bugs to occur in testing compared to their performance if there was no velocity score in their mind. The software engineer will have completed their task, but the quality of the code may cause additional time to be spent on it in the future. Velocity sounds like a good idea initially but again there are issues that cause an over-estimation of performance or else it causes a worse product to be released.

## Code Analysis:

Visual Studio offers code analysis to measure the performance of a software engineer. The idea is that the team decide on coding standards and practices at the start of the project to follow and adhere to. Visual Studio then allows these standards and practices to be set as rules [11]. If a software engineer follows these rules and guidelines, we can conclude their code is of a high quality and they are performing well in their role.

When code is published, VisualStudio checks if it conforms to these rules previously set. Using code analysis can improve the quality by helping to find common problems and violations of good programming practice [12]. It searches for specific code patterns that have been known to cause problems rather than compiler errors and warnings.

Code Analysis can be helpful for a team but rather than measuring performance, it instead checks to see if the developer if adhering to the coding regulations set at the beginning of the project. One could argue that there is a correlation between good performance and following coding rules but there is also the possibility the software developer has done little work, but that little amount satisfies the code analysis rules. We have the possibility of the speed of a software engineer being reduced as they are focused on making the quality of their code perfect. This is not an ideal scenario in the fast-paced environment that is software engineering where new code and algorithms are being released daily.

## Churn Rate:

Churn rate is how often a software engineer rewrites their code shortly after it has been committed or checked-in to a project. This is seen as a natural part of the software engineering process but having a high or low churn rate can indicate that a software engineer is not performing to their best. GitPrime's data team found that code churn levels frequently run between 15-30% of all code committed [13].

The obvious indication with churn rate is that if it is at a high percentage, then that software developer is writing sub-standard code, and is spending valuable time re-writing code when they should be focusing on new features or user stories. This indicates the software engineer is performing at a low level compared to their normal productivity.

However, we can also see a decrease in productivity if a software engineer has a low churn rate. A low churn rate indicates that the code committed is to a high standard and requires very few changes to it. This seems like a positive and would suggest that software engineer is performing well. But if we compare this churn rate to their average or previous churn rate and it is significantly lower, it suggests this developer is spending too long on a feature or task. Low levels of churn rate can indicate that a software engineer is sacrificing speed for quality which is can cause performance to decrease.

Churn rate is relative to each software engineer and there is no healthy rate to suggest that one software engineer is performing better than another. Churn rate can be useful to compare one software engineer's performance to their performance on an old project, but it cannot compare two software engineers working on the same project. Churn rate can be useful, but we can see the errors if we use this as the sole method for measuring a software engineers' performance.

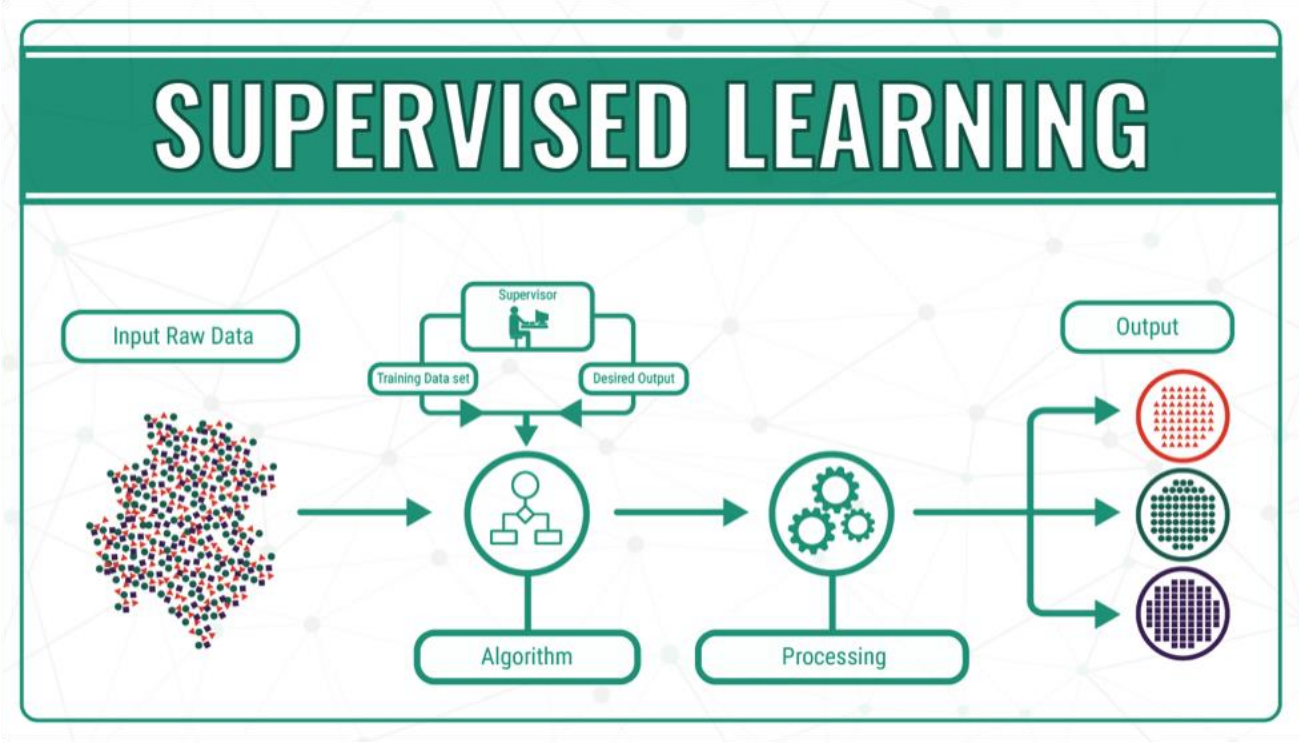
## Health/Exercise:

Having a healthy lifestyle is known to improve a person's work performance and it is no different for a software engineer [14]. A healthy lifestyle is known to reduce stress and thus allow an employee to focus more on their job rather than be distracted by other areas when working. Regular exercise has been shown to help with stress management and promote proper brain function. Another way to measure how productive an employee is their sleep time. On average, a person should aim for 7-8 hours of sleep to function at their best. If a company could monitor these areas of a software engineer, they may be able to estimate how well he/she are performing and if there is room for improvement. Although these performance measures may be helpful, they may also raise ethical concerns which I shall discuss later.

## Machine Learning:

Machine Learning can measure the performance of a software engineer using an algorithm or a statistical method to measure performance. Machine learning is a computer system that is able to learn and adapt without following explicit instructions, using algorithms and statistics to analyse and draw conclusions from patterns in data [15]. I have decided to focus on both the supervised and unsupervised learning algorithms for machine learning. These are considered some of the most common machine learning algorithms currently [16].

1: Supervised learning:



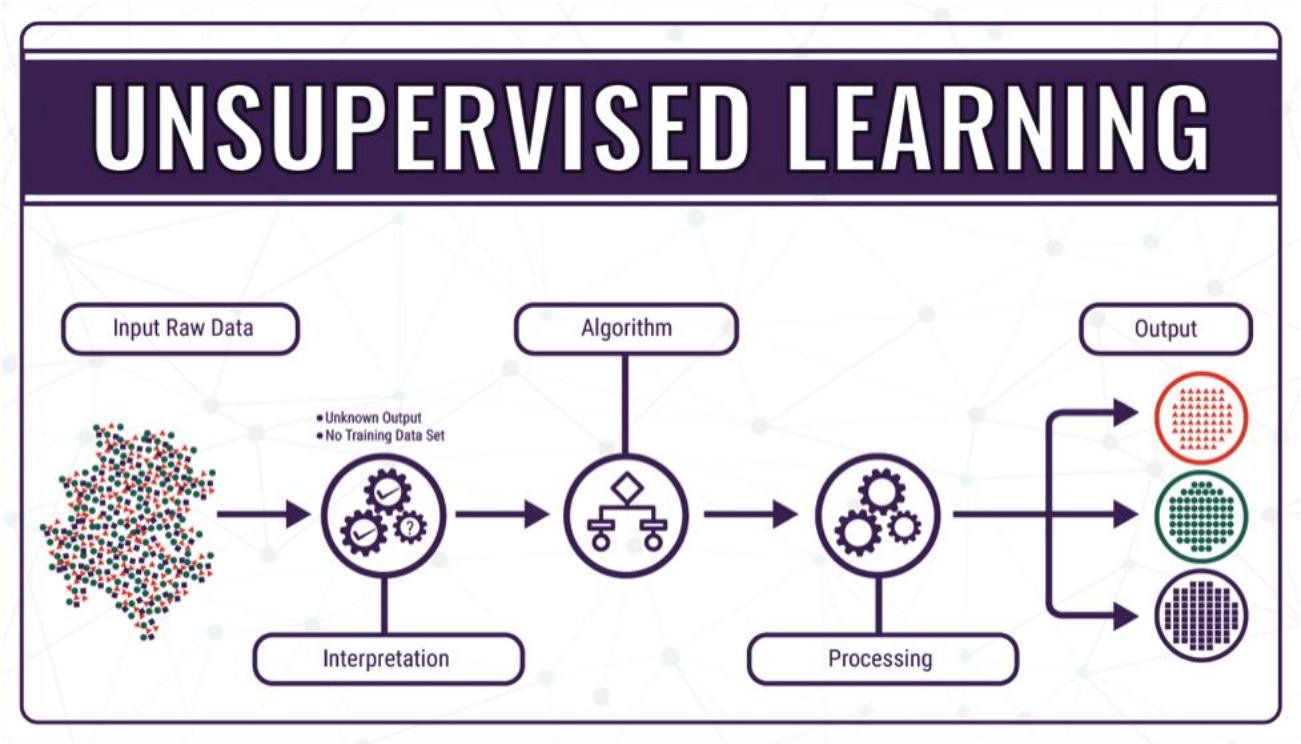

[17]

The supervised learning algorithm is where the machine receives raw unfiltered input and makes predictions about the output value. The machine makes these predictions using a learning algorithm and is only seen as functional once the machine predicts the correct output to a satisfactory level. If the machine predicts an output incorrectly, a human will be able to correct it with the predicted or correct output. The human is seen as the supervisor of the machine.

An issue with supervised learning is that because the machine is using a learning algorithm made by a human and is also being corrected by a human, there is the possibility of bias. We have seen bias in machine learning with the AI system COMPAS which I shall discuss later in the ethics section. While this machine may produce the correct output for the human who created the learning algorithm, the output may be less desirable for someone who does not share that possible bias.

Another possible issue with supervised machine learning is what data do you give the machine? As outlined above we have seen some of the data we can collect from software engineers that could in theory measure their performance. However, we have also seen the problems with all of these methods. Data quality is essential for machine learning algorithms to function as intended [18]. Without clear data and results, it is unlikely machine learning will be able to evaluate the performance of a software engineer accurately or consistently.

2: Unsupervised learning:




[19]

The unsupervised machine learning algorithm does not involve human confirmation for the output. The machine is given raw and unfiltered data and produces an output based on the machine noticing patterns itself. This algorithm is mainly used when we do not know the expected outcome of the data set, and we hope a machine may find a new pattern to give a clear result. This eliminates the bias problem with the human supervisor mentioned above but it also includes new problems

The primary concern is will the output accurately measure the performance of a software engineer? Because we do not know what the output will be, the machine may not produce a useful or helpful output to accurately measure the performance of a software engineer. There is no feedback system for someone to correct the machine as we do not know what output we should expect, and we also lack the human supervisor.

It is difficult to imagine how a machine could solve the problem of how to measure software engineering performance without including external factors such as helping junior team members or such that are unlikely to be represented as data. Although the supervised method may involve human bias, I believe it would produce a more accurate output for measuring performance compared to an unsupervised machine learning algorithm. However, I still believe there are better methods to measure a software engineer's performance compared to machine learning. I believe there are too many external factors that a machine learning algorithm will not take into consideration.

## Environment:

Another way we can measure the performance of a software engineer is by looking at their surrounding environment when working. Studies have shown that the performance of employees increase when they are in a comfortable environment. Factors such as office temperature, desk type such as stand up desks and chairs designed for ergonomics can increase the performance of an employee [20] [21] [22]. These factors can be easily improved or introduced and the increase in performance may be greater than the cost of implementing them.

With the increase of people working from home due to COVID-19, a software engineers' environment could be a determining factor in measuring their performance. At home, there is an obvious increase in distractions which could have a negative effect on an employee's performance. There is a possibility that communication between software engineering teams could decrease due to not meeting face-to-face on a weekly basis as is the norm for most teams. This may lead to a decrease in performance for all software engineers in a team due to a decrease in effective communication. Although there are additional distractions, there could also be an improvement due to a person being in a more comfortable environment like their own home and feeling less stressed which could lead to an increase in performance.

## Ethics:

Ethics are the moral principles that govern a person's behaviour or the conducting of an activity [23]. Ethics allows a private and work life balance when measuring the performance of an employee. I shall look at the ethical issues of gathering data and the analysis of this data to measure the performance of a software engineer.

## Gathering Data:

For the majority of the performance methods mentioned, they do not cause an ethical issue. Measuring performance involving code submitted by a software engineer does not cause any ethical concerns as this is submitted willingly by an employee and the code is the company's property if the software engineer has worked on it during company time. However, some performance measurements mentioned previously such as measuring a person's average heart rate or level of exercise using data is seen as a breach of privacy and raises an ethical issue.

The main ethical issue with these performance measures is how the data is gathered by the employer. In practice, the only consistent way to gather this data is to constantly monitor a software developer's heart rate and level of exercise per day and compare that to their productivity level. If the head of a software engineer team or company can see that one of their employees has a higher average heart rate or is not exercising daily, then they might conclude that this could lead to a decrease in performance. This could cause the employer to set expectations outside of work, which I believe to be highly unethical.

Although this monitoring may help measuring a software developers' performance, it raises a valid ethical issue.

## Analysing data:

If we imagine that there are no ethical concerns about gathering the data and the employer can gather personal information about one of their employee's, the employer must derive a plan on how to use this data to measure performance. As seen above with supervised machine learning, there is the possibility of bias when judging performance using this method. This hypothetical algorithm developed by a human could produce incorrect results which may not be noticed initially but could cause issues of unfair judgement in the future. We have seen an example of this bias before, where the AI system COMPAS produced twice as many false positives that previous convicts of African American descent were likely to reoffend [24]. This shows that bias can be included in any machine learning method.

Another potential issue we have with analysing this data is that we do not have the ability to see the inner workings or 'thought process' of the machine if it follows the unsupervised machine learning algorithm. Due to a human not being able to correct a machine using an unsupervised algorithm, the data recorded may be of no use at all. This causes us to have a lack of control over how the

data is used and may cause it to be worthless. We also have the potential issue where the machine could develop a bias towards male or females skewing either way. 25% of software engineers are female [25] , so using this data and unsupervised learning, the machine could hold a bias towards female software engineers where it could measure them as under or over performing depending on the reduced data it will receive [26] .

**References:**


1: https://wiki.c2.com/?LinesOfCode

2: https://www.aivosto.com/project/help/pm-loc.html#:~:text=A%20logical%20line%20is%20a,not%20a%20line%20of%20code

3: https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0

4: https://www.geeksforgeeks.org/number-guessing-game-in-python/

5: https://www.geeksforgeeks.org/number-guessing-game-in-java/

6: https://www.usehaystack.io/blog/software-development-metrics-top-5-commonly-misused-metrics

7: https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/

8: https://www.indexcode.io/post/best-kpis-to-measure-performance-success-of-software-developers

9: https://www.klipfolio.com/blog/cycle-time-software-development

10: https://en.wikipedia.org/wiki/Velocity_(software_development)

11: https://www.devteam.space/blog/how-to-measure-developer-productivity/

12: https://docs.microsoft.com/en-us/cpp/code-quality/quick-start-code-analysis-for-c-cpp?view=msvc-160

13: https://www.pluralsight.com/blog/tutorials/code-churn#:~:text=In%20benchmarking%20the%20code%20contribution,churn%20(75%25%20Efficiency)

14: https://www.mentalhealthfirstaid.org/external/2018/01/healthy-lifestyle-better-job/#:~:text=Adding%20just%20a%20few%20healthy,just%20be%20better%20off%2C%20too

15: https://www.oxfordlearnersdictionaries.com/definition/english/machine-learning?q=machine+learning

16: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

17: https://medium.com/@michaelgraw/putting-supervised-and-unsupervised-learning-to-work-for-your-business-c7bb68f50efa

18: https://www.provintl.com/blog/5-common-machine-learning-problems-how-to-beat-them

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

20: https://indoor.lbl.gov/sites/all/files/lbnl-60946.pdf

21: https://www.independent.co.uk/news/uk/home-news/standing-desks-work-office-tiredness-productivity-health-benefits-research-nhs-study-sitting-a8578561.html

22: https://www.interstuhl.com/vintage/ia-en/news.php?nr=2780

23: https://languages.oup.com/google-dictionary-en/

24: https://aibusiness.com/document.asp?doc_id=761095

25: https://leftronic.com/women-in-technology-statistics/#:~:text=25%25%20of%20software%20engineers%20are,than%20those%20born%20before%201983.

26: https://www.internationalwomensday.com/Missions/14458/Gender-and-AI-Addressing-bias-in-artificial-intelligence