***Measuring Software Engineering Report***

Contents

**INTRODUCTION2**

1. **Measuring Engineering activity2**

1.1 - Frequency of commits……………………………………………………………….…2

1.2 – Cycle time………………………………………………………………………………….….2

1.3 – Testing …………………………………………………………………………………………3

1.4 - LOC……………………………………………………………………………………………….3

1.5 – Code complexity……………………………………………………………………….…3

1.6 - Deployment Frequency………………………………………………………………...4

1.7 – Sprint Burndown…………………………………………………………………….……4

1.8 - MTTR………………….…………………………………………………………………….…4

**2- Platforms Available4**

2.1 - PSP…………………………………………………………………………………………….…4

2.2 - GitPrime………………………………….…………………………………………………….5

2.3 - Velocity ……………………………….…………………….……………………………….5

2.4 - HayStack……………………………………………………………………………………….6

**3 - Computation over Software Engineering Data 7**

3.1 – Unsupervised Machine Learning……………………………………………….7

3.2 – Supervised Machine Learning……………………………………………………7

**4 - ETHICS9**

**CONCLUSION10**

**REFERENCES…………………………………………………………………………………………....11**

**Introduction**

Software engineering has been on a rise in recent decades, and with this there have been various progressions in the viability of measuring software engineering progress, be it from and individual or organizational standpoint. We already know the huge impact software has had, is currently having and will have in the future as it steadily leads to the progression of humanity and industry as a whole.

The ability to measure software engineering is hugely important, and the requirement of this had led to the creation of platforms to accommodate teams of developers and different metrics to view the progress and current work being carried out by individual od teams of developers.

In the following report I will be going through four separate sections all of which are based about software engineering and how to measure it. The four sections are split as such, and each is pertaining to the following. The first section will be on how people can measure engineering activity, such as through the cycle time metric as well as the commits made. The second part of the report will focus on the platforms on which you can gather and perform calculations over data sets such as GitPrime and Velocity. The third section is on the various kinds of computation and finally the fourth section is about ethics of data such as personal information.

1. **- Measuring Engineering activity**

Throughout recent history there has always been attempts to measure software engineering and as such there are numerous ways of measuring engineering activity, some of which I shall discuss in this following section.

* 1. **- Frequency of commits**:

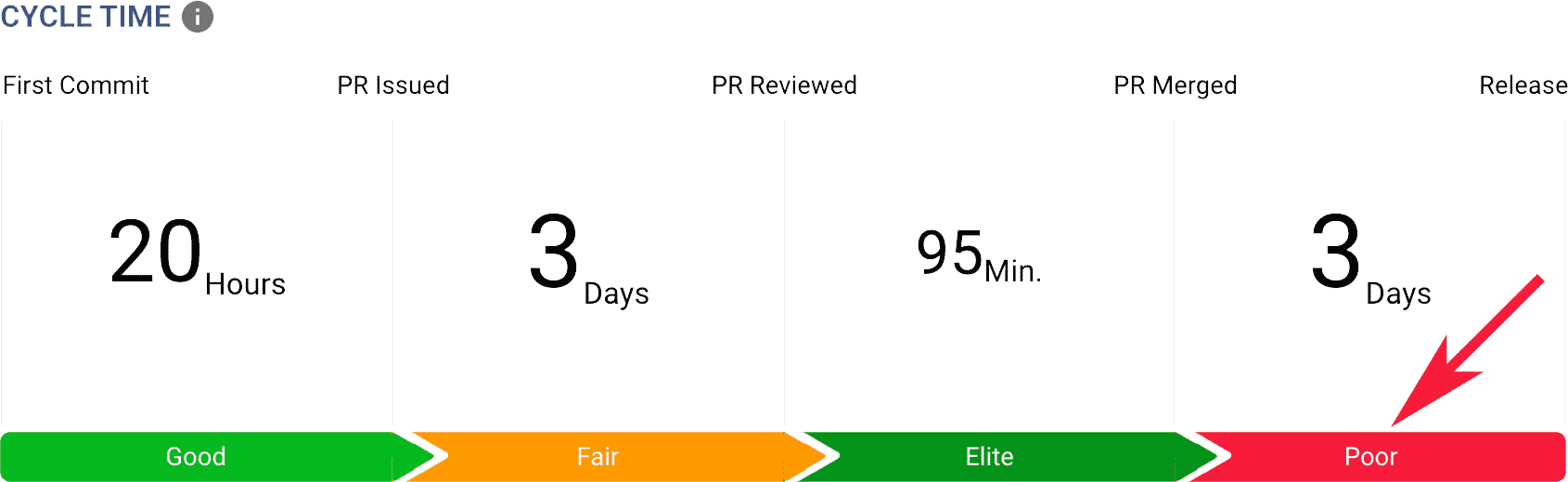
This is one of the simplest ways of viewing how consistent and productive a software engineer is, it could also be rather useful to see if a software engineer is facing problems within their work as they may not have made a commit in a few days.

Unfortunately, commit count is similar to the Lines of Code metric as it is considered to not be a true representation of the efficiency of the developer as commits are generally nothing more than a save point in developers work.

**1.2- Cycle time**:

This metric measures the time spent on starting then completing a project. It has proven useful towards measuring a company’s development velocity. The cycle time metric takes use of an agile development approach and as such employs four general stages during its measurement, these four are coding, pickup, review and deploy.

This proves incredible useful for identifying trends or problems faced when coding. For example, say based off your measurements it generally takes one day to fix a bug, which then suddenly jumps to a week to fix them. This would allow you to view that some kind of error has occurred, as well as to pinpoint where and when said error began. Another use of employing cycle time while measuring software engineering is it allows you to view where the majority of delays occur, such as in time taken to peer review code or when testing occurs.



**1.3 - Testing**:

This is another useful metric often employed as it allows one to view the code coverage of the coding and commits already made. Using it determines the percentage of code validated under a test procedure which allows you to fix and bug that may occur as well as to detect them with ease. It also allows one to test the “resilience” of code should it fail or should an error occur.

Testing also offers one the opportunity to find the efficiency of the code created as well as to find its performance through the use of load testing or stress testing. It allows creators and business to set a “safety net” so that the code can handle certain required minimums to reduce the likelihood of errors in the future and it also offers one the opportunity to increase the security of the code which is a vital aspect especially for organizations and users.

**1.4 - Lines of Code**:

Lines of code is one of the more frowned upon metrics when used to measure software engineering activity, but it is still a viable metric when employed properly. One of the main reasons lines of code as a metric is disliked is because it can lead to inaccuracies. For example, two engineers can have the same task and one of them might write ten times the volume of code compared to the other despite them both working perfectly. The lines of code metric would measure these despite the other engineer with a lower volume of code probably having code that will face much less issues in the future as well as being easier to manage.

The other problems faced by employing the LOC metric is that is does not translate over different languages which can verily hugely in the amount of code required to get a working project. The usefulness of it comes from using it to estimate the cost of running a project or product as there is a correlation between LOC and this cost. Generally, the lower your LOC is the better and more efficient the creation will be.

**1.5 - Code Complexity**:

Similar to LOC code complexity is a formal measure of software engineering. The complexity of a piece of code relates to how unwieldy and complicated it is for a developer or viewer to understand. Code complexity is useful to rule out any overly complex segments of code or to at least allow one the opportunity to simplify it.

The more overly complex code is the higher the likelihood of bugs, errors, unreliable testing and other general code defects. Code complexity can also help highlight a developer’s skill and experience to allow one to review the general skills of a developer. The elimination of unnecessary complexity leads to more efficient, readable code.

**1.6 - Deployment Frequency**:

The metric of development frequency is, quite simply, the frequency at which code is produced. This can vary from bug fixes to improved capabilities. This agile delivery metric is extremely useful as you can vary to see the frequency of the organization as a whole to the various teams present.

It is often used to ensure that software is delivered early and can prove invaluable to those who have infrequent and unreliable deployments to allow them to finish the project in a timely manner. Overall deployment frequency is an extremely useful metric.

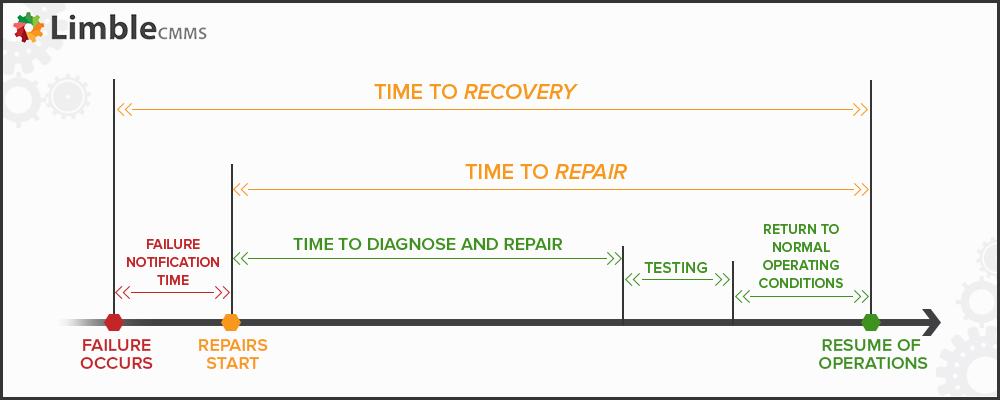
**1.7 - Sprint Burndown**:

Sprint burndown is a highly useful agile metric which can be extremely useful in measuring how teams get their work done consistently as well as highlight teams who were not given a large enough workload and hence manage to finish consistently early, or similarly those who have too large a volume of work can be shown by those who unable to get the work done in time. This metric can allow you to make sure your work is done consistently and allows you to measure the overall workload decreasing in increments rather than a large drop off.

**1.8 - MTTR (Mean Time to Repair)**:

The final metric ill speak of is mean time to repair. This metric is an essential failure metric which is used to represent the average time take for the restoration of a system or its components. MTTR is the time taken to detect, diagnose and fix an issue that has occurred.

It is useful as it allows an organization to view how well prepared, they are and how well they respond for unplanned maintenance and also allows said organization to view places where improvement is necessary to remove said errors from occurring again. This information allows the organizations to reduce their downtime which can prove quite expensive generally. Fast response time is an essential customer service and a lack of such can prove to be a liability for an organization, and monitoring MTTR can help with this issue.



**2 - Platforms available**

In this section I will highlight the platforms available on which one can gather and perform calculations over data sets.

**2.1 - Personal Software Process** **(PSP):**

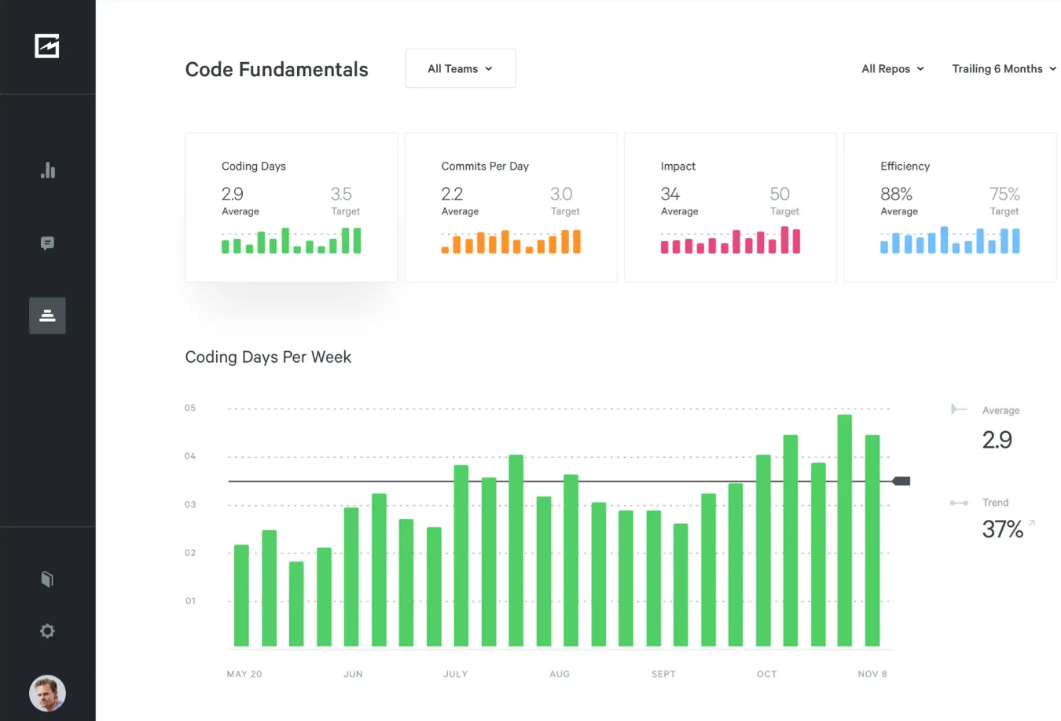
PSP is the structure that assists software engineers in measuring and developing their way of working. It does so by assisting the engineers in planning, helping them manage the standards of their projects, and increase the consistency and standard of their work as a whole. PSP is split into four levels, and these range from PSP0 to PSP3.

PSP0 refers to improving the individual, such as coding standards and personal management. PSP1 then refers mostly to time managements and prior planning. PSP2 relates to personal quality management such as code reviews, and finally PSP3 is personal process evolution. This ensure that the engineers plan, measure and track the quality of their products and allows the engineer to fucus on the overall quality from the beginning of the project to its end.

**2.2 - GitPrime**:

GitPrime is one of the largest companies in performance measurement and caters to numerous large clients. GitPrime is extremely useful analyses data gathered from the version control system which are being employed by said companies. GitPrime employs various Git of product management tools such as GitHub, Bitbucket and GitHub enterprise. GitPrime also uses two main reports for coaching developers. One of these is the player card, which can allow a manager to see a contributors percentile performance, a work log of the projects currently being undertaken and also how the review is being handled as a group. The other is Snapshot, which plots contributors over a quadrant based upon their average throughput and churn.

GitPrime also has the ability to display a project timeline which displays the work progress in terms of commits, impact and velocity. This hugely beneficial to find and isolate any outliers or issues which may have occurred which would allow one to rectify the problems.



**2.3 - Velocity:**

Velocity is somewhat similar to GitPrime but varies in a couple of ways. Velocity offers an overview of a developers work using developer360. In this it includes a snapshot of what a developer is currently working on and the impacts of said work, an activity piece which details what the developer has been working on for the past month which allows one to view the workload of a developer over time and make appropriate adjustments, foundations which allows one to view how a team is according to the critical velocity metrics, and finally skills which can allow one to view the languages a developer has been working with.

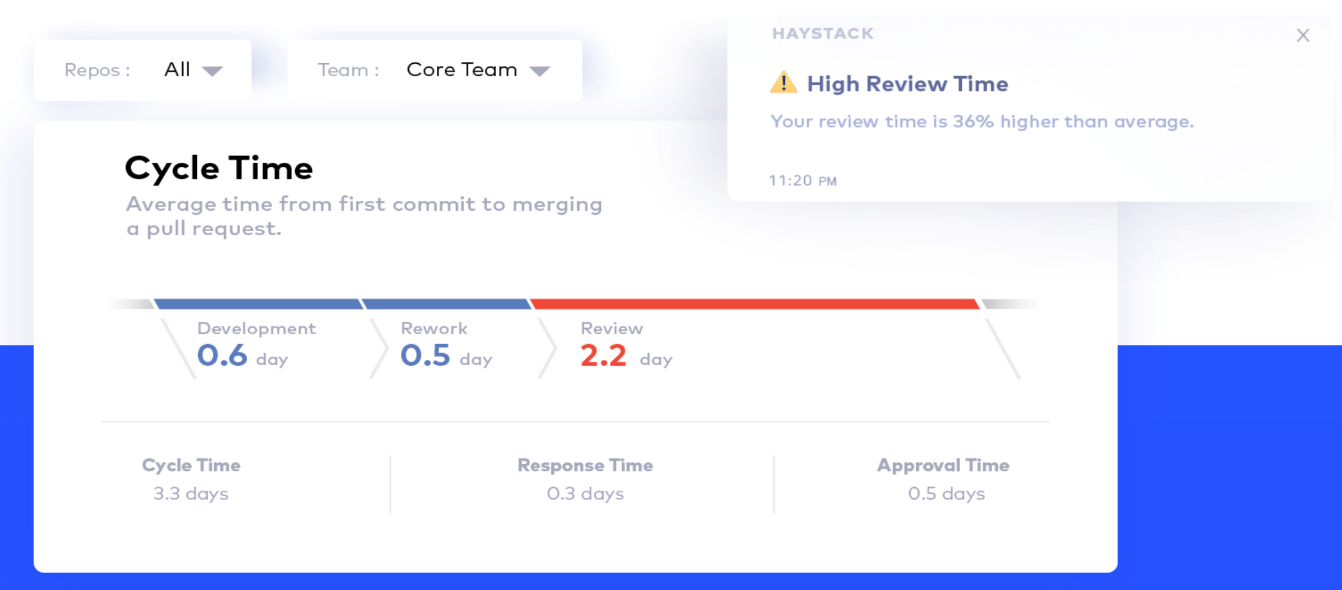
From a tracking progress standpoint velocity offers two main features, Overview and Analytics. Overview allows one to view a summary of the progress made by a team bashed on various metrics. This allows one to view the trends across the various metrics and observe which areas are in a decline while the other is at an incline. Analytics meanwhile offers managers data so that they can create reports specifically tailored to individual fields or areas within a project.

Finally for setting targets or end goals velocity offers a feature which allows one to view their various goals and offer percentile-based gaols encourage a greater standard of code and such.

**2.4 - HayStack**:

This is another hugely popular platform and is known to have good quality metrics and is extremely developer friendly. HayStack offers various metrics such as cycle time, deployment frequency and potential burnout, offering a view at both the development of the project as well as helping to ensure that developers are not overworked which could lead to more errors later on. It is also extremely developer friendly offering team level insight throughout.

It is also extremely easy to use offering a simple clean dashboard and a user-friendly experience showing only key alerts and performance metrics rather than filling the screen with an abundance of hard to read data, and it is also quite secure which is a necessity for developers. Below is an example of HayStack.

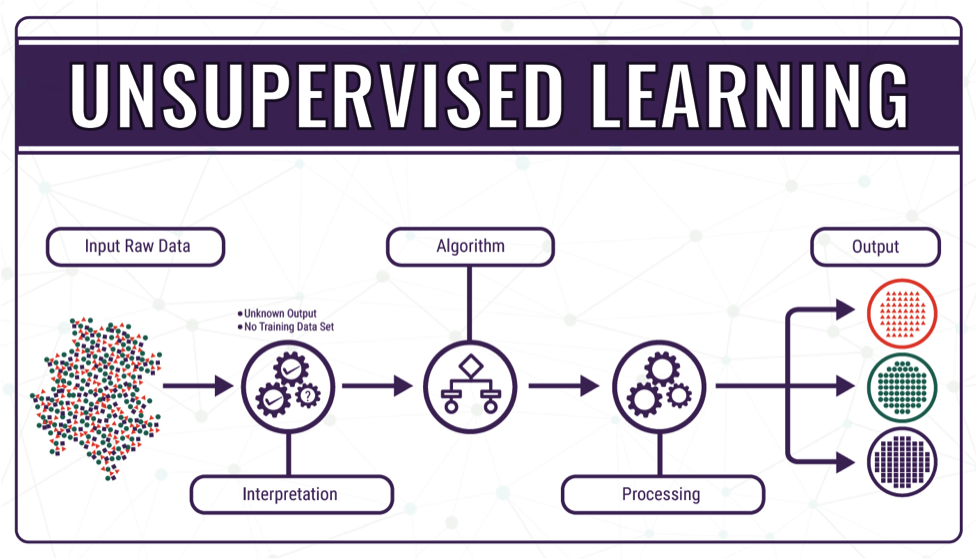


**3- Computation over Software Engineering Data**

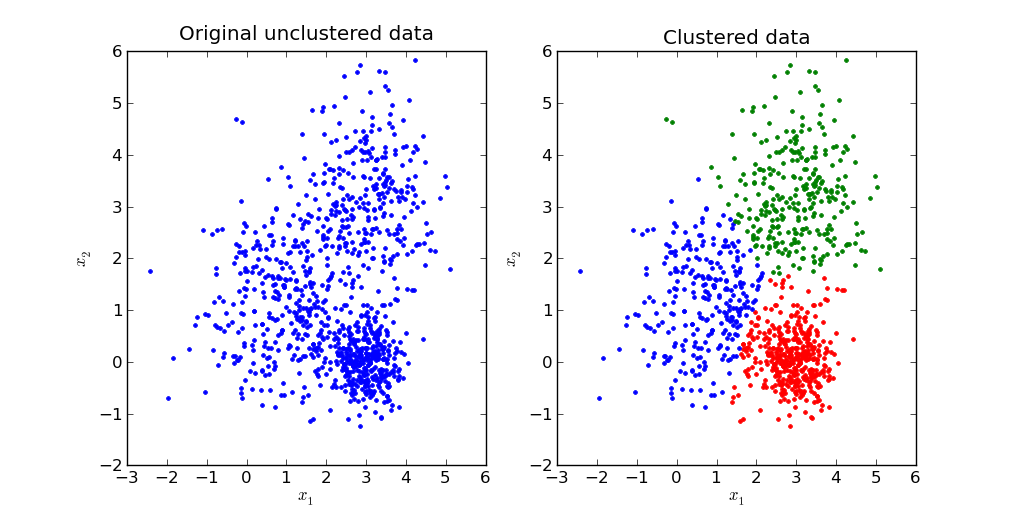
In this section of the report, I will be explaining how data computation can be carried out in order to profile the performance of software engineering, through the use of various algorithms.

**3.1 -Unsupervised Machine learning:**

Unsupervised machine learning uses algorithms to analyze and cluster unlabeled data sets, which discover hidden data or data groupings without the requirement for humans. Using this it has the innate ability to notice patterns or differences in information makes it a valuable computational method. In this form of machine learning only input is required and it then creates its own output. Below is an example.

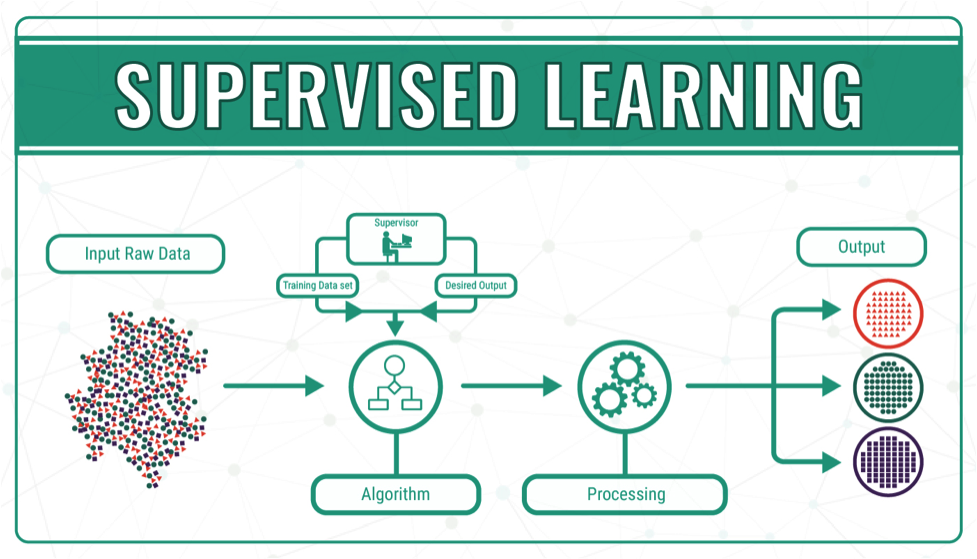


One such example of this unsupervised machine learning is clustering. Clustering is a data mining technique which groups data and is used to process raw data into groups through their patters or differences. There are numerous clustering algorithm types such as exclusive clustering and overlapping clustering.



**3.2 -Supervised Machine learning:**

This form of machine learning is slightly different from unsupervised machine learning. Similarly, it is a subcategory of machine learning which uses labeled datasets to train algorithms which can then sort data accurately as well as make accurate predictions based off these algorithms. It includes inputs and correct outputs, which allows the model to learn over a period of time.



This kind of machine learning is used to assist in dealing with various real-world problems and has proven to be an asset to various organizations. A few examples of supervised machine learning are Neural networks, which process training data to mimic the human brain through employing nodes. It passes data through the nodes and learns how to map this data using supervised learning. When the cost function is near zero the model can be considered to be quite accurate.

Diagram

Description automatically generated

Another example of supervised machine learning is linear regression. This is used to identify the relationship between a dependent variable and other independent variable which will allow the machine to attempt to predict future outcomes. If there is a number of independent variables it is known as multiple linear regression. This method uses the least squares method to plot a line of best fit.

Chart, scatter chart

Description automatically generated

**4 – Ethics**

Ethics is a hugely important issue especially when dealing with a huge amount of personal information of varying individuals. The breach of such trust can have a detrimental effect on the companies affected and the individuals involved in said leaks. Unfortunately, there have been many unfortunate events which have led to the personal information of individuals being used in unethical ways, such as when Facebook was involved in the theft of half a billion users’ personal data which was then used by Cambridge Analytica to display targeted political adverts. This led to a major questioning of Facebooks credibility.

As a general rule, the gathering of personal information is quite unethical, but the gathering of the data produced by software engineer about their work is a very different story. The availability of the information allows managers to properly and effectively support their teams as well as smoothly complete projects while decreasing the risk of bottle necks or bugs. It is vital though that these metrics will only collect information on the developers work and not any personal data or data on outside projects as this would be a breach of privacy and would prove to be extremely unethical, especially if the team manager or peers had access to this private information which is not at all relevant to the developers work. As such it is vital that there be measure in place to ensure that the data collected is the correct kind so as to avoid any unethical breaches.

It is also extremely important that the organizations store the employee’s information safely to stop any possible leaks of information, as well as for those who are not employed but the organization may have the personal data of for other purposes. I believe that this is imperative as there have been numerous large data leaks by organizations or attacks made on these organization to steal data which has led to a huge amount of personal private information being leaked online or stolen. One such example of this is the HSE data breach recently, which led to the information of numerous people being stolen, including their PPSN details and contact details. Things like this led me to believe that a huge emphasis must be placed in security in order to prevent these kinds of things from occurring again.

It also needs to account for the possible errors in the data collected which could portray one employee as working less than others despite them doing the same if not more, merely due to a certain metric, for example a developer who makes a commit every hour, even if no changes have been made, versus one who commits only when a change of some significance has occurred. In this scenario the first developer will have a larger number of commits but will have done much less work than the second developer, which in turn could lead to this developer feeling underappreciated despite the amount of work they are doing.

Overall, it is my belief that as long as barriers and regulations are in place so as to safeguard data and stop the abuse of personal data, the analysis of software engineers’ data on their work is reasonable.

**Conclusion**

In conclusion, the measuring of software engineers’ activity, the platforms available and computational methods of the data sets is progressing extremely quickly and is constantly being built upon to make it more streamlined and effective. The platforms and computational methods available make it much easier to view and plot the work of software engineers as well as helping managers engage their teams more effectively and offer them the ability to optimize their team’s workflow and strategy, as well as quickly and effectively work through any errors that occur as they can easily view where these errors and bugs first began to appear.

I also believe that collecting the work data of software engineers is perfectly reasonable, so long as there are rules in place to stop any leaks of personal data as well as to avoid breaches that could occur which would lead to personal information being stolen.

Overall, I believe that there has been significant progress in the metrics used to measure software engineers’ activity, the platforms available to graph and observe this data and the algorithms available to cluster and display the information and variable entailed within them, all of which I have discussed in my report.

***References***

* <https://waydev.co/software-development-metrics/>
* <https://www.klipfolio.com/blog/cycle-time-software-development>
* https://stackify.com/track-software-metrics/
* <https://www.pluralsight.com/blog/teams/lines-of-code-is-a-worthless-metric--except-when-it-isn-t->
* <https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/>
* <https://linearb.io/blog/what-is-code-complexity/>
* <https://humanitec.com/blog/deployment-frequency-key-metric-in-devops>
* <https://plandek.com/2021/04/29/metric-of-the-week-deployment-frequency/>
* <https://medium.com/@infopulseglobal_9037/top-10-software-development-metrics-to-measure-productivity-bcc9051c4615>
* <https://nextservicesoftware.com/news/mean-time-to-repair-mttr/>
* <https://www.geeksforgeeks.org/personal-software-process-psp/>
* <https://codeclimate.com/blog/velocity-vs-gitprime/>
* <https://www.usehaystack.io/blog/top-alternatives-to-gitprime-in-2020>
* <https://technative.io/why-unsupervised-machine-learning-is-the-future-of-cybersecurity/>
* <https://www.ibm.com/topics/unsupervised-learning>
* <https://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/>
* <https://www.ibm.com/topics/supervised-learning>
* <https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/>
* <https://medium.com/the-21st-century/machine-learning-a-strategy-to-learn-and-understand-chapter-3-9daaad4afc55>
* <https://limblecmms.com/blog/mttr-mtbf-mttf-guide-to-failure-metrics/>