Measuring Software Engineering

Ralph Swords Student Number: 19335541

Introduction

Merriam-Webster defines software engineering as a branch of computer science that deals with the design, implementation, and maintenance of complex computer programs. The measurement of the software engineering process can give us vital insights into the time and resources required to complete a project, how much each engineer has contributed to the project, and the progress made on the project. Analysing these metrics can improve the efficiency and quality of a team's work, as well as the final program they deploy.

Measurable Data

If the measurement of software engineering is important, then we must think about what it is we can measure. Luckily, due to the use of source control frameworks, software engineers effectively leave a trail of data, that describes their every action. As such, there are a variety of metrics that are used to measure software engineering. I feel that the exploration of some of these metrics, not only highlights how software engineering can be measured but also how difficult it is to quantify an engineer's contributions.

One of the earliest software metrics is called source lines of code (SLOC). This metric attempts to quantify the effort made by an engineer, by counting the number of lines that the engineer contributed.

In theory, this metric provides a clever way to measure the productivity of an engineer and the size and maintainability of a project. I believe SLOC, however, has fundamental flaws. Firstly, if one wished to infer the performance of an engineer by counting the lines of code they have contributed, what would be considered "good"? One person may argue that many lines would suggest a productive contributor, as they have provided a large body of work to the project. It is, however, true that often a better programmer can write more concise code, which is easier to understand and maintain. Therefore, it might be better to have a smaller line count.

A metric often used in conjunction with SLOC is the measurement of bugs per line of code. This is meant to give an idea of the quality of the code being produced. It is certainly preferable to produce code with fewer bugs, as this will less time and resources will be used to fix them. Relying on these two methods for the measurement of software engineers, however, can lead to counterproductive workplace habits. To boost the number of lines that they have committed, an engineer might intentionally write less concise code than they are capable of. The use of bugs per line of code may also encourage engineers to avoid tackling tricky problems.

SLOC and bugs per line could be seen as the most basic software engineering metrics. They are easy to measure and not too difficult to understand. It is for these reasons that these metrics were utilised in software engineering's formative years. The issue with them is that they are not aligned with the general goals of a software engineer; that is to reduce the lines of code, reduce the number of bugs, and reduce the time of completing tasks. There are, due in large part to the quantity of data that software engineers leave behind as they work, several metrics we can use.

Ideally, one would want engineers to be efficient in their work. By efficient, I mean what percentage of their code is productive. This could be measured by balancing the coding output against the code's longevity. If an engineer has high efficiency, then their code provides monetary value to their business for a longer period.

A useful metric for calculating is called "code churn." This is a measurement of how many lines of code were added, deleted, or modified in a specified period. If an engineer consistently has a high code churn, that might indicate that engineer is not efficient.

Code churn can give a variety of insights into the progress of a project. In the preliminary stages of a project, there will be a period of prototyping and exploration. This naturally will lead to a high code churn. As a project progresses towards release, one would expect to see the code churn to decline.

When engineers encounter a difficult problem, a large amount of backtracking and exploring will occur. It is almost inevitable that a team will encounter such problems during developing a project. These problems will result in a spike in the code churn.

A high rate of churn could also be caused by unclear requirements and issues surrounding uncertain external stakeholders. These situations can cause a large amount of frustration within engineering teams. If these situations arise, it is important to define concrete requirements.

The most important thing a software engineer can do is to make code contributions. Often, engineers have undergone years of train and acquired university degrees dedicated to providing them with the skills to build and solve difficult conceptual problems. It is for these skills that many companies are willing to pay high salaries.

Software engineering involves process overheads. Tasks such as planning, meetings, research, and gathering specifications are essential to the process of engineering and require time. Ideally, however, an engineer would spend as little time as possible on these overheads, and more time contributing to the code base. Active days is a metric that counts the days that an engineer contributes code. This metric allows a team to see the cost of these overheads and ensure that the overheads do not overburden them.

Another important metric is impact. Impact is a measure of the effect that an engineer's contribution has on a project. The impact depends on many factors, for example, the amount of code in those changes, the severity of those changes, and the number of files those changes affect. Impact can give an indication of which engineers are more familiar with the several aspects of the project. A contributor who has impacted several files in the project is going to have a greater understanding of the entire project than a contributor who has only impacted one file.

Platforms for Gathering and Analysing Data

As I have stated, every action that a software engineer makes gets creates data. That means every line of code, every comment, every commit, every pull, and every contribution to a discussion board generates data that can be gathered, stored, and processed. As you can imagine, each engineer generates a huge quantity of data. Knowing this, and knowing that in a company, there could be thousands of engineers, raises an important question, how is it possible to perform computational operations on such a large set of data? I will now discuss how the dawn of cloud computing has made it and financially possible to gather and analyse extremely large sets of data. Then I will explore some of the specialised frameworks that have emerged to perform data analytics.

First, I would like to discuss the concept of big data. The amount of data we produce has grown exponentially in the past ten years. In 2010, 2 zettabytes (2^{21} bytes) of data were created, captured, copied, and consumed worldwide. In 2020, however, that figure rose to 64.2 zettabytes. That means, that every person generates 6.185 megabytes of data each second.

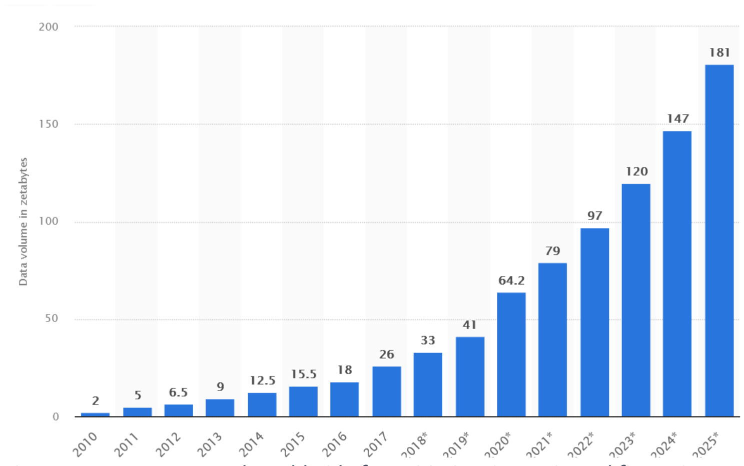


Figure 1: Data generated worldwide from 2010 to its projected figure in 2025. https://www.statista.com/statistics/871513/worldwide-data-created/

There are several different definitions of big data. Wikipedia defines big data as data sets too large to be dealt with by traditional data-processing application software, and its challenges include capturing data, data storage, data analysis, search, sharing, and visualisation. Sam Madden defines big data as data that is too big, too fast, and too hard to be for existing tools to process. In 2001, industry analyst Doug Laney introduced the 3V model to describe the features required for data to be considered big data. The three Vs are:

1. **Volume:** This describes the size of the data. Volume is often considered the defining characteristic of big data. Terabytes or petabytes are considered the benchmarks for big data.

2. **Velocity:** This refers to the rate at which the data is generated or the rate at which it must be processed. When we consider velocity, there are two main types of data processing: batch and stream. Batch processing allows data to be stored over a period and processes the data in blocks. This model is suitable in a situation when no real-time processing is required. Stream processing is the opposite. It allows data to be as they arrive, and it is key for real-time processing and data analytics.

3. **Variety:** This refers to the several types of data being produced. Big data is classified into unstructured (audio, video, images, etc.), semi-structured (XML, JSON, etc.), and structured (spreadsheets, relational databases, etc.).

Storing, processing, and analysing big data requires an extraordinary amount of computational power. Using standard, commercially available desktops, or laptops to analyse big data would be either impractical or impossible due to the amount of time and energy it would take. Purchasing the hardware required to work with big data is impossible for most businesses due to the prohibitive cost.

Cloud computing, however, means it is now possible to work with big data. Cloud computing is the on-demand access, via the internet, to computing resources. These resources could be applications, servers, data storage, development tools, or networking tools. There are three main models of cloud services:

1. **Software-as-a-Service (SaaS):** SaaS is application software that is hosted in the cloud. You can access the application through a web browser, or a desktop client. It is the primary delivery model for most commercial software today.

2. **Platform-as-a-Service (PaaS):** PaaS gives developers on-demand hardware, infrastructure, and development tools for running, developing, and managing applications.

3. **Infrastructure-as-a-Service (IaaS):** IaaS provides access to computing resources, such as servers, networking, and storage over the internet.

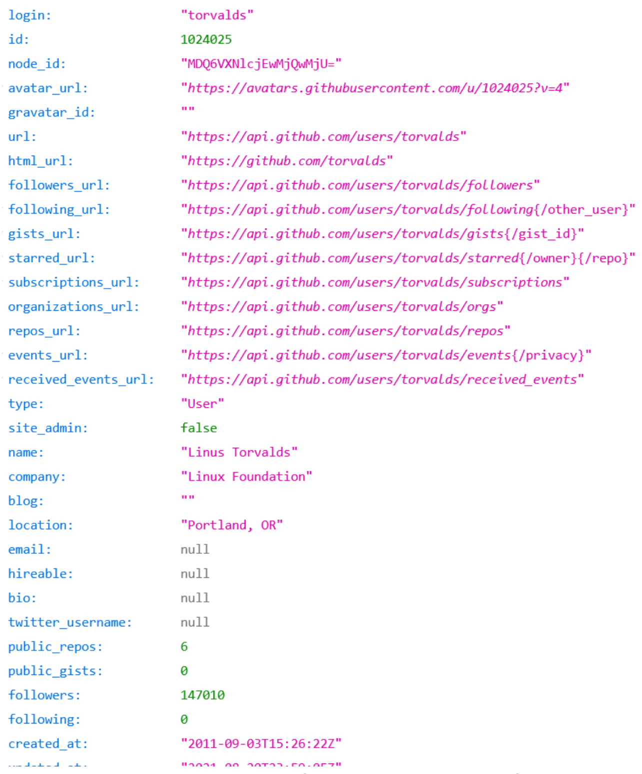
The most important aspect of cloud computing, in a wider context, is that it often follows a pay-as-you-go or subscription pricing model. This means that businesses do not have the high capital expenditure of building their own, bespoke infrastructure to process big data. They only need to use the infrastructure built by cloud service providers and pay for the time and power that they need.

Now that we have explored the possibilities of exploring enormous quantities of data, I will now discuss some specialised platforms for gathering and processing data in software engineering.

GitHub

GitHub is an internet hosting service for version control using git, founded in 2008. GitHub provides all the functionality of Git, including commit, push, pull, branch, and merge, as well as feature requests and wikis.

As of 2021, GitHub reports having 73 million developers and more than 200 million repositories. Crucially, GitHub stores a massive amount of data on these developers and repositories and makes this data available to access through its Rest API. Using the API, a company can access the raw data on what each of their engineers is doing. With this data, using languages like Python, they can create bespoke processing and analysing tools to evaluate the performance of their engineers.



```
login:                "torvalds"
id:                   1024025
node_id:              "MDQ6VXNlcjEwMjQwMjU="
avatar_url:           "https://avatars.githubusercontent.com/u/1024025?v=4"
gravatar_id:          ""
url:                  "https://api.github.com/users/torvalds"
html_url:             "https://github.com/torvalds"
followers_url:        "https://api.github.com/users/torvalds/followers"
following_url:        "https://api.github.com/users/torvalds/following{/other_user}"
gists_url:            "https://api.github.com/users/torvalds/gists{/gist_id}"
starred_url:          "https://api.github.com/users/torvalds/starred{/owner}{/repo}"
subscriptions_url:    "https://api.github.com/users/torvalds/subscriptions"
organizations_url:    "https://api.github.com/users/torvalds/orgs"
repos_url:            "https://api.github.com/users/torvalds/repos"
events_url:           "https://api.github.com/users/torvalds/events{/privacy}"
received_events_url:  "https://api.github.com/users/torvalds/received_events"
type:                 "User"
site_admin:           false
name:                 "Linus Torvalds"
company:              "Linux Foundation"
blog:                 ""
location:             "Portland, OR"
email:                null
hireable:             null
bio:                  null
twitter_username:     null
public_repos:         6
public_gists:         0
followers:            147010
following:            0
created_at:           "2011-09-03T15:26:22Z"
```

Figure 2: GitHub API response for the user account of Linus Torvalds

Pluralsight Flow

Formally GitPrime, Pluralsight Flow is an engineering analytics platform. It can gather information on commits, pulls, and commits across multiple platforms. Pluralsight believe that their metrics can help engineering teams identify bottlenecks in their workflow, thus increasing the team's efficiency. Flow's metrics are broken into five categories:

1. Code metrics. These include coding days (a day when a developer contributed code to the project), churn (code which is deleted or rewritten

shortly after being written) and impact (the severity of edits to the codebase, as compared to repository history).

2. Submit metrics. These include responsiveness (the time it takes a submitter to respond to a comment on their pull request), comments addressed (percentage of comments to which a submitter responds) and receptiveness (percentage of comments the submitter accepts).

3. Review metrics. These include reaction time (the time it takes for the reviewer to review a pull request), involvement (percentage of pull requests that the reviewer participated in) and influence (ration of follow-on commits made after the reviewer commented).

4. Team collaboration metrics. These include time to resolve (the time it takes to close a pull request), time to first comment (time between when a pull request is opened and the time of the first comment) and follow-on commits (number of code revisions added to a pull request).

5. Knowledge sharing metrics. These include sharing index (how information is being shared amongst a team), number of pull requests reviewed (total number of pull requests that were reviewed) and number of users reviewed (total number of submitter users that were reviewed).

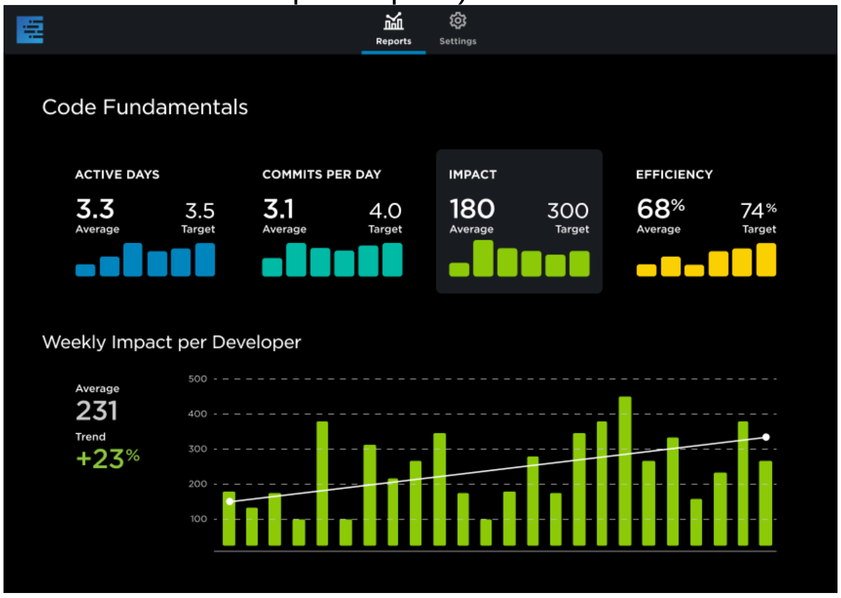
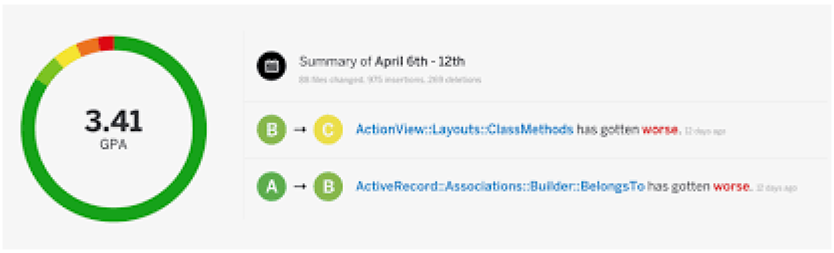


Figure 3: Example visualisation from Pluralsight Flow

Code Climate

Code Climate is a web-hosted software that provides analytics and reviews of code. Like Flow, Code Climate works in conjunction with Git. At every pull request, Code Climate analysis the code, considering the complexity, churn, duplication, and coverage, to quantify the quality of the code. It then provides automated comments on the pull requests. These comments are designed to focus review discussions about the code, thus making the team more efficient.



Summary of April 6th - 12th
88 files changed, 975 insertions, 269 deletions

B → C ActionView::Layouts::ClassMethods has gotten worse. 12 days ago

A → B ActiveRecord::Associations::Builder::BelongsTo has gotten worse. 12 days ago

Figure 4: Example output from Code Climate

Algorithmic Approaches

Form early stages in the development of software engineering, there have been attempts to develop algorithms to measure the performance of developers and the progress of projects. It was clear that it would be beneficial if concepts such as code complexity and cost could be calculated automatically. The issues are that these concepts can be difficult to define in such a way in which they can be measured.

One early attempt at developing algorithms to measure software complexity was the Halstead complexity measures. These were introduced by Maurice Howard Halstead in 1977. Halstead proposed that computer programs were implementations of algorithms. These algorithms were collections of tokens that could be either operators or operands. One could then count these tokens and determine which are operators and which are operands. These are the base measures:

- n1 = Number of distinct operators
- n2 = Number of distinct operands
- N1 = Total number of occurrences of operators
- N2 = Total number of occurrences of operands

The above variables are used to calculate several measures that allows one to quantify the complexity of a program.

Program Length: This is the total number of operator and operand occurrences. $N = N1 + N2$

Program Vocabulary: This is the number of unique operators and operands. $n = n1 + n2$.

Program Volume: This represents the space needed to store the program in bits. $V = N \times log_2(n)$

Program Difficulty: This measures the difficulty of writing or understanding the code. $D = \left(\frac{n1}{2}\right) \times \left(\frac{N2}{n2}\right)$.

Programming Effort: this measures the amount of mental activity needed to translate the algorithm into code. $E = D \times V$

Software engineering projects are notorious for going over time and budget. Due to the complex nature of software projects, it can be exceedingly difficult to estimate the financial cost and effort required to complete a project.

There have been many approaches suggested to estimate the cost of a project. A popular example of an algorithmic approach is the Constructive Cost Model (COCOMO). This method was proposed by Barry Boehm in 1981. It is a regression model based SLOC. The key parameters of COCOMO are effort, the amount of labour that will be required to complete a task and schedule, the amount of time required for the completion of the project.

In the basic COCOMO model, effort is calculated by:

$$E = a(KLOC)^b$$

KLOC is the source lines of code, measured in the thousands. The effort is measured in person-months. The schedule is calculated by:

$$S = c(E)^d$$

Using the schedule and the effort, we can then calculate the number of staff required to complete the project:

$$\text{Staff required} = \frac{E}{S}$$

The values a, b, c, and d are constants whose values are determined by the type of project.

These methods for calculating the complexity and cost of a software project are, at their core, based on counting lines of code. As I have previously stated, counting lines of code is not an effective way of measuring the performance of a software engineer. Engineers can easily manipulate measures based on lines of code and they do not capture some of the key behaviours of software engineers. In recent years, machine learning algorithms have been used more effectively to measure software engineering.

Machine learning is a branch of artificial intelligence that focuses on the use of data and algorithms to imitate the way humans learn. It uses statistical methods to identify clusters and classify objects in large data sets with multiple variables. Machine learning is a powerful tool as multivariate data sets are notoriously difficult for humans to comprehend and visualise. I would like to discuss two categories of machine learning, supervised learning, and unsupervised learning.

Supervised learning uses labelled data sets to train algorithms that attempt to classify data accurately. What this means, is that given a data set in which the data has been classified into groups of similar objects, supervised learning algorithms will compare input data with these groups and try to place the new data into the group with which it is most similar.

An example of a supervised learning method is k-Nearest Neighbours. Suppose we have a new observation, kNN looks at the k closest labelled points in the data set and places the new observation in the group to which most of those k points belong. kNN is a non-parametric classification method. This means that it does not make assumptions about the underlying distributions of the data and can be used on any data set. It also means, however, that the classifications are fixed, and it does not consider the probability that an observation belongs to a group. The results of kNN are also sensitive to how the optimal value for k was estimated.

Discriminant analysis is an alternative method to kNN. Unlike kNN, it is a parametric method. It assumes that the distinct groups follow the multivariate normal distribution. This means that this method can find the probability that a new observation belongs to each group. The new observation is then placed into the group in which it is most likely to belong. If the parametric assumption is appropriate, this method can be more accurate than kNN. If it is not appropriate, however, the results can be very inaccurate.

Unsupervised learning uses algorithms to analyse and cluster unlabelled data sets. These methods are used to identify groups of similar data sets that humans might not be able to identify. The groups that these methods identify can then be used in supervised learning.

Hierarchical clustering is an example of an unsupervised learning method. In it, all points are considered groups. Then, the two most similar groups are joined together, meaning there is now one less group. This is repeated until there is only one group remaining. Hierarchical clustering results in a tree-like structure known as a dendrogram. The results of hierarchical cluster are extremely sensitive to the choice of linkage method. A linkage method is how the algorithm decides which two groups are the most similar. Performing a hierarchal cluster analysis on the same data set with two different linkages can produce two entirely different groupings.

K-means clustering is another unsupervised learning method. It is an iterative algorithm. The first step of k-means clustering is to choose the number of clusters, k, and designate the initial cluster centres. The second step is each data point is assigned to the cluster whose centre it is closest to. The third step is that for each cluster, the new centroid is calculated. The fourth step is that the sum of the squared distances of each object to its cluster centroid is found. The fifth step is to re-assign each observation to the cluster whose centroid it is closest. Steps three to five are repeated until no observations change cluster. K-means clustering can be sensitive to the initial cluster centres, so great must be taken when choosing them.

Ethical Considerations

The measurement of software engineering relies on the gathering, storage, and processing of enormous quantities of data produced by engineers. This data could be viewed as personal, and as is often the case when personal data is being used, several ethical questions are surrounding the measurement of software engineering.

When considering the ethics of managing copious amounts of personal data, it is important to consider what type of data it is and how it is gathered. The examples I have given throughout this report have been based on the engineer's contributions to the source code. Metrics like code churn, active days, and impact can be measured by analysing the data produced by version control systems. This data is a by-product of the engineering process. Therefore, I do not believe that the gathering of this data can be interpreted as infringing on the individual engineers right to privacy. This data is directly related to the primary function of a software engineer, so it is reasonable to use this data for performance evaluation.

In this report, I have focused on quantitative measures of software engineering, based on contributions to the source code. There are, however, metrics and platforms that base their measurements on the health of the employee and how they interact with their colleagues. An example of these platforms is Steelcase. Steelcase produces office furniture that measures the posture, heart rate, breathing, and stress levels of employees. They can then use this data to coach the employee on good posture habits and recommend standing breaks. This data is also shared with the employer.

Another example of these platforms is Humanyze. Humanyze produces a badge that the employee can wear. This badge can track the location of the employee within the office. It can also record who that employee is conversing with and the tone of that conversation.

I will not discuss these measurements in detail since I did not cover them in my report. I will say, however, that the measurement of this data raises more ethical concerns than the metrics based on the source code. This data is more personal, as they are based on the measurement of a person's health and conversations. This data would give an employer a level of insight into an employee's personal life that I would consider inappropriate. This data is not directly related to their primary function within the company where they work.

If employers are using data to evaluate an employee’s performance, then the employer has the responsibility to analyse that data fairly. With the increase in popularity of using machine learning to analyse large data, however, this fairness is not a guarantee. Bias is a key concept in statistics. It is the difference between an estimator’s expected value and the true value of the parameter being estimated. In a more practical sense, machine learning bias can be seen as the occurrence of results that are prejudiced due to erroneous assumptions made by the machine learning algorithm. In 2017, it was found that speech-based machine learning algorithms associated the words “female” and “woman” with arts and humanities occupations, while “male” and “man” were closer to maths and engineering professions. Another example of machine learning bias was in 2015, Google’s image-recognition algorithm labelled images of black people as gorillas. Unable to immediately fix the problem, Google prevented any image from being labelled as a gorilla, even if the image was of a gorilla.

Error is an unavoidable part of machine learning. Many of the most accurate and popular machine learning methods involve forcing a data set to fit a probability distribution. In this way, machine learning tends not to handle outliers well. It also would not take an individual employee’s circumstances into account. Machine learning offers advantages in automation and can give you a good general picture, but its results must be considered and investigated carefully when evaluating an individual.

When considering the ethical questions of measuring software engineering, it is natural to be drawn to what is expected of employers. Employee’s, however, also have to shoulder ethical responsibilities if engineering measurement is to be effective. I have previously discussed the use of the number of source lines of code as a metric. When this metric was more commonly used as a measure of the amount of work an engineer has done, a phenomenon emerged where engineers would intentionally extend the programs they were writing. This led to code being less concise than it could have been, which goes against better practices. Employees must not let the knowledge of metrics used to measure their performance dictate how they work. They must ensure they work in a way they believe to be effective and they believe will benefit the development of the project.

Conclusion

We need to ask if we should measure software engineering. Doing so means greater worker surveillance and raises concerns about the use of machine learning in its analysis. I believe, however, that it is very important to accurately measure the software engineering process. An obvious advantage of the use of these metrics is that it more accurately identifies engineers who performing well than simply relying on a manager's opinion. These engineers can then be assigned similar projects or rewarded with a promotion or pay rise. It can also identify areas in which engineers need to improve.

Another important benefit of measuring software engineering is that it can give an indication of funding and resources a project requires, as well as the progress of a project. This information is vital to know, as it can ensure that when a piece of software is deployed, that it is efficient, robust, and fit for purpose. The London Ambulance Service disaster in 1992 is an example where a rushed and under-funded software development project led to deaths. The deployment of the LAS's second attempt to automate its dispatch system led to days of severe problems within the ambulance service, before total failure of the system. Up to 46 people may have died over the two days due to the lack of emergency services.

While the measurement of software engineering may raise some concerns over the privacy of the engineers, I believe that the potential benefits that this practice provides makes it an important part of software engineering practice.

References

https://www.merriam-webster.com/dictionary/software%20engineering
https://en.wikipedia.org/wiki/Source_lines_of_code
https://en.wikipedia.org/wiki/Software_bug
https://www.pluralsight.com/blog/tutorials/code-churn
https://www.pluralsight.com/blog/teams/5-developer-metrics-every-software-manager-should-care-about
https://stackify.com/track-software-metrics/

https://www.researchgate.net/publication/348937287_Big_Data_Analytics_in_Cloud_Computing_An_overview
https://www.ibm.com/cloud/learn/cloud-computing
https://www.statista.com/statistics/871513/worldwide-data-created/
https://en.wikipedia.org/wiki/Big_data
Madden, S. (2012). *From Databases to Big Data. IEEE Internet Computing, 16(3), 4–6.*
https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/
https://www.computer.org/publications/tech-news/trends/big-data-and-cloud-computing
https://en.wikipedia.org/wiki/GitHub
https://help.pluralsight.com/help/metrics#anchor-4
https://www.blissfully.com/code-climate/
https://codeclimate.com/quality/
https://venturebeat.com/2021/09/01/software-engineering-intelligence-platform-code-climate-raises-50m/
https://www.computing.dcu.ie/~renaat/ca421/report.html
https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/
https://www.geeksforgeeks.org/software-engineering-cocomo-model/
https://www.ibm.com/cloud/learn/machine-learning
https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides5.pdf
https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides7.pdf
https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides8.pdf
https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides9.pdf
https://en.wikipedia.org/wiki/Bias_of_an_estimator
https://www.techtarget.com/searchenterpriseai/definition/machine-learning-bias-algorithm-bias-or-AI-bias
https://www.theguardian.com/technology/2017/apr/13/ai-programs-exhibit-racist-and-sexist-biases-research-reveals
https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people
https://www.researchgate.net/publication/3792694_Disaster_in_London_The_LAS_case_study