Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Measuring Software Engineering Report

Tom Roberts - 19335276

Table of Contents

Introduction

The IEEE Computer Society defines software engineering as the use of a “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”[1]. Despite being a relatively new industry, software pervades many aspects of modern life. And as the industry of technology continues to grow, so does the demand for software engineering.

This report outlines the different ways in which software engineering can be measured. First, it outlines the different types of data that can be collected to perform these measurements. In the second section, some of the computational platforms available to facilitate these measurements will be described. The third section will outline some of the computation techniques that can be used on these datasets. Finally, I will discuss some of the ethical issues that come with the collection, and use of this data.

Why Measure Software Engineering?

Measuring software engineering activity is an incredibly important task. Certain measurements can help determine the quality of a software application. Not to mention, measuring a software engineer’s work can help to determine their overall progress as a developer. This type of information is very important for both the individual engineer, and the bigger corporations, as it can help improve efficiency and productivity.

Measuring Data

Measuring activity and productivity in software engineering is not a straightforward operation. Because of the nature of the industry, determining the inherent value of what has been produced is challenging.

Data analysis is commonplace across most industries, and is how engineering activity is measured. The goal of data analysis is to extract meaning from data in order to uncover valuable information, draw conclusions, and assist in decision-making. However, to perform data analysis, one must first determine what data to collect.

What data can be collected?

Due to the broad scope of what software engineering can entail, there are many different types of data we can collect, each with their own unique characteristics.

Number of Commits

Looking at the frequency or total number of commits is one way to measure software engineering. While the total number/frequency of commits reveals close to nothing about codes quality or efficiency, it has the ability to provide some useful insights. It's generally good practice for developers to be committing their work to version control systems regularly. However, programmers tend to each have their own preferences when it comes to when commits should be made. This means that the number of commits can vary greatly between developers, making this metric quite flawed.

Lines of Code

The lines of code, also referred to as Source Lines Of Code (SLOC), is a metric used to determine the size of a program. It is calculated by simply counting the number of lines in the source code. There are two main methods of calculating the lines of code, Physical Lines of Code (LOC) and Logical Lines of Code (LLOC). Physical Lines of Code is the easiest metric to measure, since it simply involves counting each line of code present in the source code.

Logical Lines of Code is slightly more complicated. Instead of counting each line of code, it tries to calculate the number of executable statements.

Both of these calculations can be easily manipulated, making this metric counter productive in practice. This can encourage developers to write unnecessarily long and complex code, which could be achieved much more easily. This ruins readability, making the code more difficult to follow.

Time spent Working

Another method to measure software engineering activity is to measure the time spent working on a project. It makes a lot of sense in theory. In a perfect world, the more time a developer spends on a project, the more work gets achieved. Unfortunately, life just isn't that simple. The more time spent on a project doesn't necessarily reflect the amount of work or skill that went into it. To show this, let's compare two developers, Developer A - who took 2 hours to implement a feature, and Developer B- who took 20 mins to implement a feature. Determining the productivity of each developer by simply counting the time they spent on a feature is flawed in this situation, and would encourage Developer B to work unnecessarily slower. Next, let's flip the situation, and measure the productivity of each developer by how time efficient they were. This is a good idea in theory, and would reward Developer B for their efficiency. However, this would rush Developer A, which would make their job much more stressful. It would also encourage them to rush their development cycle, which would most likely increase the number of bugs/errors present in their work.

Test Coverage

In general, code with a high level of code coverage has a lower probability of containing an undiscovered software flaw.

Test coverage measures the coverage of test cases on the source code. It is the percentage of lines of code that have been executed by the test cases. In most cases, code that has a high amount of code coverage is less likely to contain an undiscovered software bug or error. A piece of software that has a very low code coverage has a higher probability of encountering an undiscovered bug.

Code Churn

Code churn is a way to measure the evolution of a piece of code. Code churn is when a developer changes or deletes their own code shortly after it has been written. It is a normal part of the development process, and can show the progress of a developer. However, high levels of code churn can indicate that the developer is spending a lot of their time fixing their own mistakes.

Due to the complexity of what software engineering entails, there are many forms of data we can collect. Each of these have their own advantages and disadvantages, and there is no “perfect” way to measure software engineering. Measuring a developer's commit frequency can tell us how much a developer is making changes, but doesn’t tell us whether these changes are productive or not. Measuring test coverage can help us determine whether a piece of software is likely to encounter undiscovered bugs. Test cases are usually defined by the developers themselves, so they can be prone to human error, and can be manipulated easily. For the measurement of software engineering, it is important to collect a wide range of data to allow for better analysis, since individual measurements aren’t the best representation of productivity.

Platforms Available

Personal Software Process (PSP)

The Personal Software Process (PSP) is a systematic software development process that aims to assist software engineers in better comprehending and optimising their performance. It was first introduced by Watts Humphrey, who was a huge advocate for

structured development procedures. PSP gives developers a basic way to measure and monitor their work, and it encourages them to change their process when problems arise.
The Personal Software Process collects the majority of their information from developers through forms. This is where problems can start to rise, since these forms require manual input from developers. This not only is more work for developers, but also increases the likelihood of human error in the data gathered. Data should also be completely unbiased, and since data is entered manually, there is a chance that it could be intentionally manipulated.

LEAP Toolkit

The LEAP toolkit was developed to address the problems that came with the Personal Software Process system. It creates a store of developer-specific process data that developers keep with them as they move between and between projects. The main advancement between the PSP and the LEAP toolkit was the automation of data analysis. Unfortunately, like the Personal Software Process (PSP), the LEAP Toolkit required manual data input, which created its own set of problems, such as human error and bias, making the data inconsistent.

Hackystat

Hackystat is a free and open-source platform for collecting, analysing, visualising, and interpreting information about the software development process. It can be easily integrated with the most popular code repositories, making it quite popular among developers. Users of Hackystat integrate sensor software into their development tools, which captures and transmits raw development data to their web service, the Hackystat SensorBase database. This data is collected unobtrusively, which means that the user is not aware of when data is being collected. Other online services can query their web service, the Hackystat SensorBase, to construct higher-level abstractions of this raw data. Making it easy to generate visualisations, or integrate the data with other services.

GitHub

GitHub is a web hosting service for software development and version control using Git. It offers all of the functionality of Git, with many of its own features. It provides access control and a variety of collaboration tools for each project, which include bug tracking, feature requests, task management, continuous integration, and wikis.
GitHub is without a doubt, one of, if not the most popular version control systems used globally. Not only is it used for version control, it can also be used to measure software engineering activity. A large amount of user/repository data is already available on the site. The GitHub REST API can also be used to retrieve data from the site, which is very useful for visualisations.

Pluralsight Flow

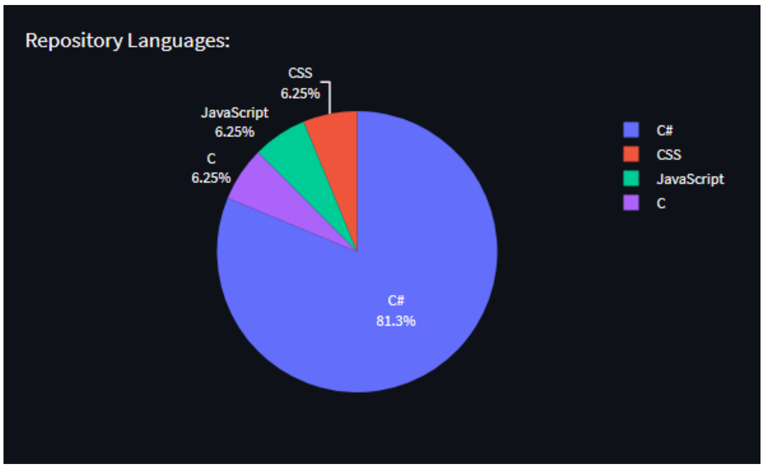
While GitHub is a great platform which gathers and performs calculations on data, the only downside is that it only provides data on repositories made through GitHub. Pluralsight Flow (previously known as GitPrime) fixes this problem.
Pluralsight Flow is a tool that examines Git metadata, which supports a wide range of modern-day version control systems. It then converts the examined date into easy-to-understand insights and reports. It is capable of visualising work patterns, which enables the leaders of software engineering projects to debug their development processes using data, resulting in a more efficient workflow for their teams.

Data Computation Techniques

There are a number of different computations that can be done over a set of software engineering data. These computations can help us determine the productivity and performance of software engineers.

Simple Counting

Compared to other more complex algorithms, simply counting can sound rather trivial. However counting can be quite useful to us in profiling the performance of software engineers, and often goes underappreciated. One useful implementation of simple counting is counting the occurrences of each coding language present in a set of repositories. This can be used to determine what languages a certain developer is more comfortable with, compared to others.

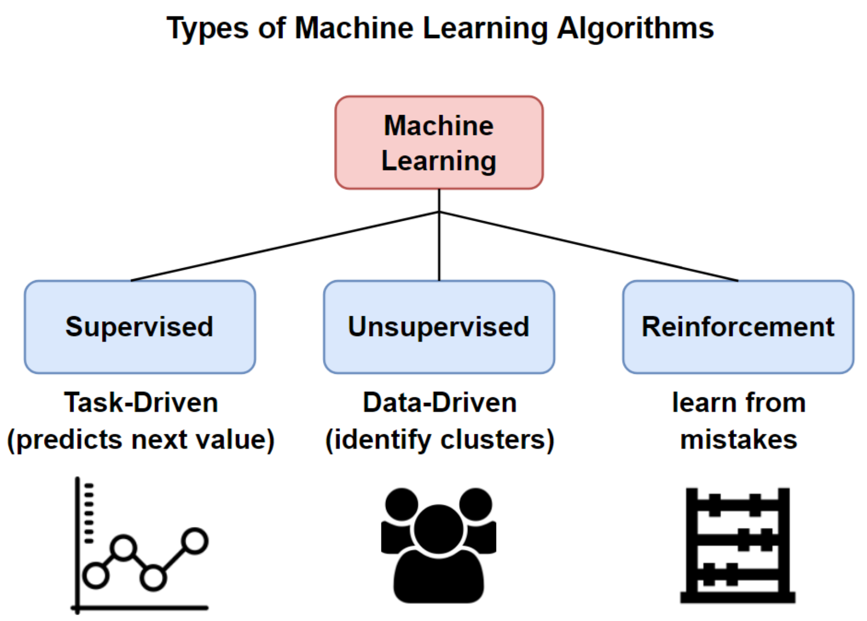


The figure above shows a visualisation on a user's most common repository languages, which was computed through simple counting.

Machine Learning Approaches

According to IBM, machine learning is defined as a "branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy." It's safe to say that this has revolutionised the world of data analytics, as we no longer need people to meticulously analyse each data entry. There are three main types of machine learning algorithms, which are:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning


Types of Machine Learning Algorithms
Machine Learning
Supervised
Unsupervised
Reinforcement
Task-Driven (predicts next value)
Data-Driven (identify clusters)
learn from mistakes

Supervised Learning

Supervised learning algorithms train the machine by example. They are supplied with a known dataset containing the desired inputs and outputs, and the algorithm must devise a technique for obtaining those inputs and outputs. The algorithm recognizes patterns in data, makes conclusions based on observations, then generates predictions. This process is continued until a high level of accuracy has been achieved.

K-Nearest Neighbours

K-Nearest Neighbours is one of Machine Learning's most basic but crucial classification algorithms. It's commonly used in real-world circumstances since it's non-parametric, which means it doesn't make any assumptions about data distribution. First, the algorithm is fed training data, which consists of a known dataset. The algorithm then makes a prediction for, and classifies new data points based on the existing data. It achieves this by examining the K-Nearest Neighbours to this point. For example, if K = 3, the machine will select the variable that dominates the three closest observations to this point.

Unsupervised Learning

Unsupervised learning is a subset of machine learning in which models are trained on unlabeled data and then allowed to operate independently on that data. Their goal is to identify a pattern in the dataset.

K-Means Clustering

K-means clustering is an algorithm technique that divides an unlabeled dataset into a number of different clusters. K is the number of predefined clusters which are established as part of the process. It's a centroid-based algorithm, which means that each cluster has its own centroid. The main goal of this technique is to reduce the sum of distances between data points and their corresponding clusters.

Reinforcement Learning

Reinforcement learning is a machine learning algorithm that rewards desired behaviours and/or penalises undesirable ones. Reinforcement learning is rarely used for processing personal information, since the definition of a desired behaviour can vary drastically between different data sets.

Ethical Concerns

After discussing the process of measuring software engineering activity, one may question whether it is ethical to do so. To measure the activity of a software engineer, their personal data needs to be collected, stored, and processed. Which can potentially raise ethical concerns. The process can result in employees feeling as if they are under constant surveillance, negatively impacting their work environment.
The measurement of software engineering activity is critical for both individual and team evaluation, but an excess gathering of data can cross a line, invading employee privacy and autonomy.

The EU enacted the General Data Protection Regulation (GDPR) in May of 2018, which has had a major impact on how data can be handled. It has reinforced the laws regarding the protection and security of an individual's personal data. Any company that obtains a person's personal information must comply with the GDPR. Another significant change brought about by GDPR is that businesses must obtain the consent of any individual they wish to collect data from. They must also describe why this information is being collected. These sets of regulations have had a great impact on which data can be collected for measuring software engineering, and gives employees more privacy regarding their personal data. I have very little problem with the collection of data as long as GDPR standards are being followed.

I believe that the measurement of software engineering activity can provide some us with some useful insights, but can have many ethical flaws. As discussed previously, there is no “perfect” way to measure software engineering, so it’s morally wrong to make decisions on someone's career based solely on this imperfect data.

Conclusion

As software engineering develops, more metrics are being used to measure software engineering. Many platforms have been developed to assist with the measurement of software engineering, by simultaneously collecting data and providing analytical insights using various algorithms.
While the gathering of personal data can raise some ethical concerns, I believe that the measurement of software engineering activity is very beneficial, as long as GDPR standards are being followed. It is critical to consider data privacy and engineer autonomy, as well as to communicate honestly and clearly with individual contributors about the collection of any data.

References

- IEEE Computer Society. Guide to the Software Engineering Body of Knowledge (Swebok(r)): 2004 Version. Ed. Alain Abran and James W. Moore. IEEE Computer Society Press, 2005. Print.
- Fenton, Norman, and James Bieman. Software Metrics: A Rigorous and Practical Approach, Third Edition. 3rd ed. Boca Raton, FL: CRC Press, 2012. Web.
- The Personal Software Process (PSP) - Watts S. Humphrey https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283
- IEEE, & International Conference on Software Engineering. (2000). Proceedings of the 1999 international conference on software engineering (21st ed.). I.E.E.E. Press.
- https://code.google.com/archive/p/hackystat/
- https://www.pluralsight.com/product/flow
- https://www.ibm.com/cloud/learn/machine-learning
- https://www.sas.com/en_ie/insights/articles/analytics/machine-learning-algorithms.html
- https://gdpr.eu/