



CS33012 Software Engineering

Measuring Performance of Software Engineering

Matthew Grouse, 19335171

November 2021

Contents

1. Introduction.....	1
2. Approaches To Measuring Software Engineering.....	2
2.1 Lines of code.....	2
2.2 Commits.....	3
2.3 Pull Requests.....	3
2.4 Velocity.....	4
2.5 Throughput.....	4
3. Analyzing Software Engineering.....	5
3.1 Waydev.....	5
3.2 GitPrime.....	5
4. Classifying the performance of Software Engineering.....	6
5. Ethical Concerns.....	7

1.Introduction

Software engineering is a broad term used to describe people who develop software, libraries and documents required for a specific purpose and combine them with scientific methods and principles. There are many stages to software engineering which make it a difficult process, this is largely due to the multifaceted timeline of a software product. In the early stages of one of these products consist of developers contacting their clients to obtain the requirements for the product. After a prototype is made then next stages follow a process which involves constant change and communication between the development team

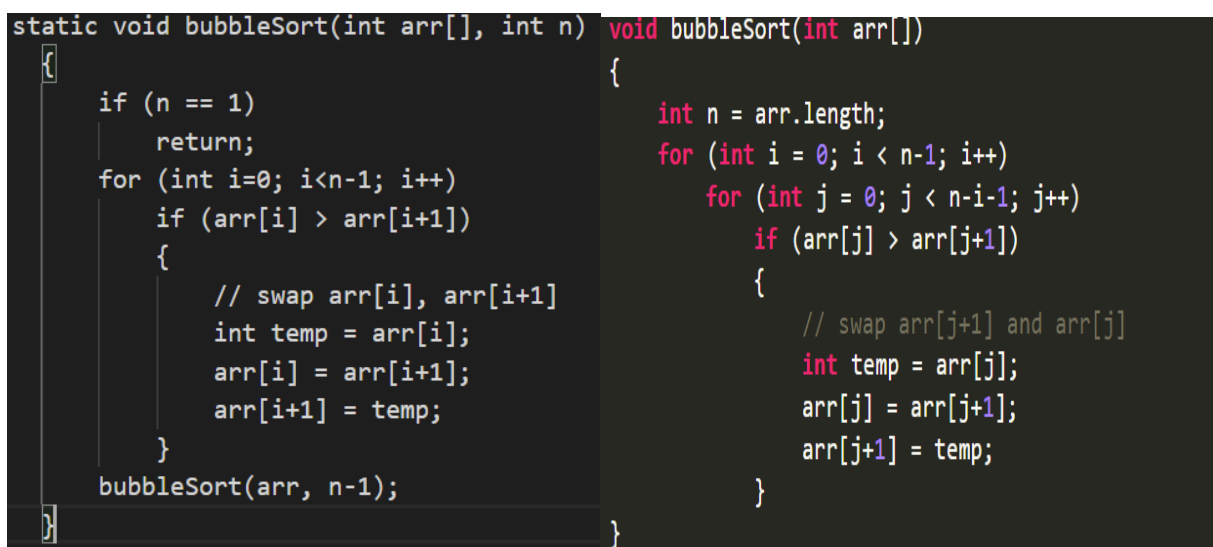
and customer. These stages are where we can analyze the performance of a software engineer and get an insight on how to be an effective software developer. In this report, I will discuss the topic of measuring software engineering. I will be assessing the ways software engineering is measured, the platforms which collect this data, the algorithms which analyze this data and the ethical concerns of collecting it.

2. Approaches to measuring Software Engineering

There are a number of ways in which software engineering can be measured. This involves lines of code, commits, pull requests, velocity and throughput. Companies monitor these metrics to assess the work environment productivity and to hopefully see an increase in company profits as a result.

2.1 Lines of Code

One of the methods to measure the performance of a software engineer is to record how many lines of code(LOC) they write over a specific time period. In this simple method, we calculate the amount of code an engineer has committed in a certain amount of time and compare it to previous periods and other software engineers. If a worker has a lot of LOC, we would consider them to be more productive and perform better than one who writes less. The main issue with this method is that we would consider someone who writes more complex longer code to be better than someone who writes simple code in less lines. In this case, we value more code over the quality of code, which is seen where the software engineer writes the same code as another but in less lines making more effective use of space and also having a quicker runtime.

The image shows two code snippets side-by-side on a dark background. The left snippet is a recursive bubble sort function in Java, starting with 'static void bubbleSort(int arr[], int n)' and using a recursive call 'bubbleSort(arr, n-1);'. The right snippet is an iterative bubble sort function in Java, starting with 'void bubbleSort(int arr[])' and using nested 'for' loops to iterate through the array. Both snippets include a swap operation using a temporary variable 'temp'.

```
static void bubbleSort(int arr[], int n)
{
    if (n == 1)
        return;
    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
        {
            // swap arr[i], arr[i+1]
            int temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
        }
    bubbleSort(arr, n-1);
}

void bubbleSort(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
                // swap arr[j+1] and arr[j]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

Figure 2.1: The above figures display two different ways of solving a bubble sort, the code on the left is recursive and the code on the right is iterative.

For measuring work efficiency, we can measure the code quantity by looking solely at the lines of effective code. Each and every software engineer have their own coding habits so instead of looking at every line of code committed, we can look at the logical lines of code. This means we count only the number of statements in the code, which works for lines of code with multiple statements. However, there are still disadvantages to this method, such as when a developer helps other members of their team. This help will not be recorded by this system but they are still performing well by contributing to the team.

Overall LOC is a good method of measuring the performance of software engineers as it is very easy in estimating the effort of a developer and it has fewer disadvantages compared to other methods. For example, there is a correlation between LOC and the overall length and cost of developing a product with a lower LOC usually implying that the team are better at the development of the product.^[1]

2.2 Commits

Commit frequency is another factor which is often used to measure the performance of software engineers however, it is considered to be a flawed metric as commits do not translate to productivity. In this method, the number of commits of the software engineer is recorded to see how often they contribute to the development of the product. If its done right, it encourages continuous delivery and collaboration for the software product. On the contrary, you can also see teams with lower commits which implies they aren't doing their work or regularly contributing to it. This highlights the main problem of measuring commit frequency as it does not show the value of work committed and just like LOC you can manipulate it by just committing less work more regularly.

From research done by Yilin Qiu, and Weiqiang Zhang in a 2015 paper [2] showed that a higher commit frequency usually showed that the developer had a higher quality of code. Not only this but they found a correlation between commit frequency, code quality and code stability.

2.3 Pull requests

We can measure the performance of a software engineer by looking at their pull request count. A pull request allows a developer to display to their team the changes and additions the have pushed to a repository or branch and discuss these changes before merging it with the source product code.

Pull request counts only show how many were created, not the complexity of the requests or the size. This like above, encourages negative behaviours. For example, an engineer can make small changes to code for each pull request such that this method becomes a quantity over quality measurement. This metric fails when software engineers who are experienced and commit a high standard of code less often compared to more frequent members of the team which shows that they perform worse.

2.4 Velocity

Velocity measures the amount of work a single team completes during a software development iteration or sprint.[4] It is one of the most popular methods for measuring software engineering performance and works by adding values to each task based on difficulty and time required to complete it. If the development team completes tasks quicker than the allotted time the velocity has then it shows the team as over performing. These values are subjected and are decided and agreed upon by the team or organization.

This method however does have its disadvantages. Its main purpose is to act as an estimation tool but it is often used as a measure for performance and as a result, developers and team leaders end up rewarding team with points. This means that teams can over-estimate the value of specific tasks and upon completion their performance looks better compared to others.

This method also doesn't take quality into account. Since teams are incentivized to complete tasks quicker, it can cause code to become more rushed or include more errors when testing. In this case, the software engineer would have completed their task but would have to spend more time down the line improving the quality of the code. Velocity is a good method to measure this performance however it contains issues which can prohibit results.

2.5 Throughput

Throughput or impact is a software engineering performance measurement which measures the total output of a team. It measures the number of tasks, features or bugs fixed in a certain time period and rewards software engineers with tickets. The higher the ticket count of an individual the more productive they are. By measuring throughput, you're looking for alignment with your goals [5]. This means that whatever part of your development you are focused on you will most likely get tickets for it, if you focus on features, you will get tickets for features.

Throughput fails to measure external forces. For example, if an engineer decides to help team members with their part of the project, the engineers performance will be low and they will

look less productive. This however, is very good for development and adds long term value to their development.

3 Analyzing Software Engineering

3.1 Waydev

There are many ways in which we can collect data to measure the performance of a software engineer. This is usually done with open sourced platforms where developers can get access to the repositories of development teams. One of these platforms is called Waydev. This platform tracks the engineers technical work based on an agile data driven method of tracking their git activity.[6]

From collecting this data there is great potential for more data driven methods which could analyze a developers performance. For example, with access to their git activity, we could measure which languages the engineer is proficient in and also how many repositories they contribute to. This shows the productivity of the engineer as we could display how much work they do, however there would be no way to see the size and standard of their code.

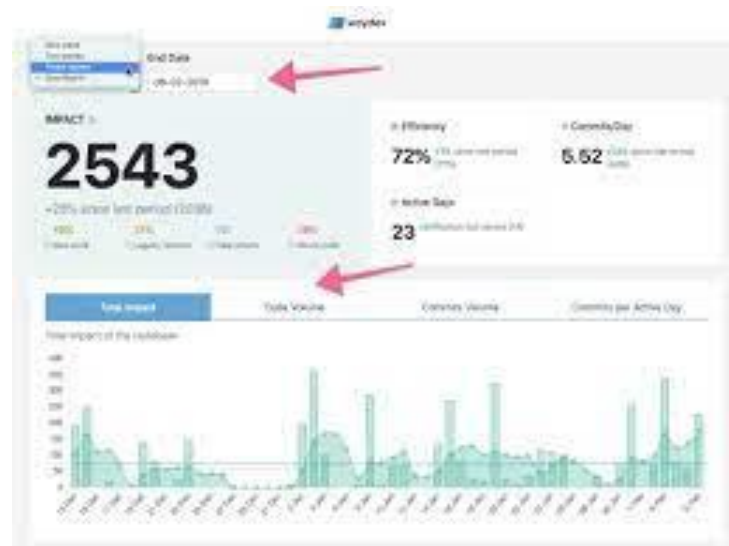


Figure 3.1: The above image displays the interface for Waydev's performance tracker

3.2 GitPrime

GitPrime is another git management system, which is more recently known as Flow, and tracks the productivity of a software engineer. This platform uses algorithms to analyze user data and show their commit count, the size of their commits, and their merge and pull requests.

This way software metrics can be easily implemented and the data visualized. GitPrime is more focused on individual performance metrics and allows the team manager to keep a closer track of their team members so they can see the strongest and weakest members.

The platform integrates with many tools such as github, bitbucket and gitlab to display the performance of engineers on these platforms. These data collecting algorithms used by GitPrime have great potential to assess a software engineers performance but also performance of teams. Using more agile metrics we could see the true breakdown of each member but also how members help each other and achieve deadlines.

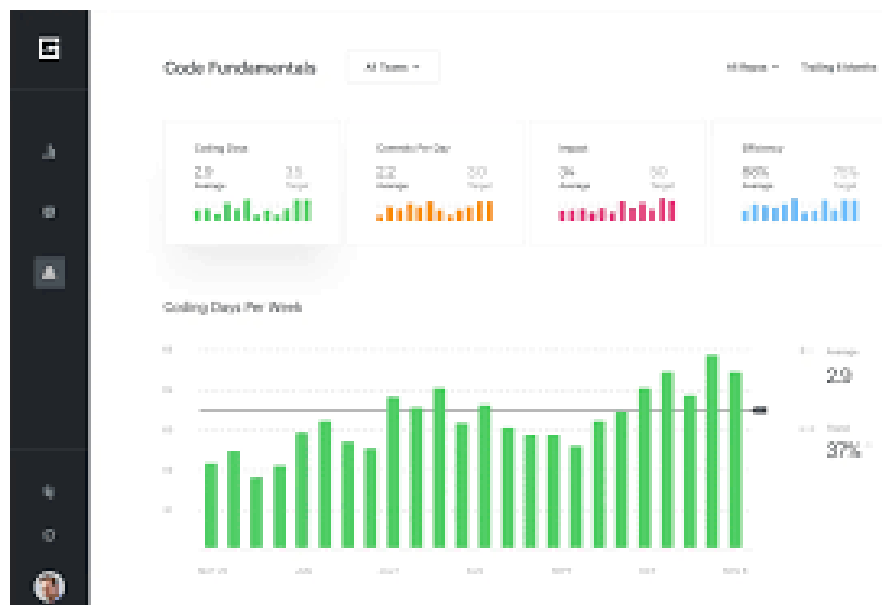


Figure 3.2: This figure shows the interface for gitPrime, one of the main platforms for showcasing software engineering performance

4. Classifying the performance of a Software Engineer

Machine learning

Machine learning is a great way to measure the performance of a software engineer using a algorithm or statistical method. It is a computer system that is able to adapt and learn without following specific instructions, using algorithms and statistics to analyse and make conclusions from patterns in datasets. Two common forms of machine learning algorithms which work with measuring software engineering performance are supervised and unsupervised learning algorithms.

Supervised Methods

A supervised learning algorithm is one which receives raw information straight from the dataset and makes predictions on the output. In this method the machine uses the algorithm

to keep predicting outcomes until it achieves the correct one at a good enough level. If an output is incorrect then the developer can correct it with the imposed result.

There are many issues with this method of machine learning as with a human designed algorithm and supervisor correcting wrongly predicted outcomes, it suggests a level of bias. For example, the machine could achieve the right outcome for the developer who created the algorithm but this might not be a preferable outcome for other developers. Another issue with this method is seen with what data you can give the machine. We have seen how the data collected from software engineers can display their performance but there are many problems with all these metrics. For example, LOC isn't always a good method of measuring performance and as a result, the quality of data used would negatively effect the machine learning algorithm.

Unsupervised Methods

There are also unsupervised machine learning algorithms. These methods don't rely on human confirmation of the output and such eliminates bias. It is used when we don't know the expected outcome of a dataset analysis so it keeps running patterns until a clear result is achieved.

One main concern with this method is that if the output is a an accurate measurement of a software engineers performance. Without a known output, the given output from the algorithm may not be helpful in demonstrating performance and with no system to correct the machine, we see more problems.

5. Ethics

Ethics are moral principles that govern a person's behaviour or the conducting of an activity. Ethics are very important when dealing with personal information and even more crucial when collecting data on people to measure their performance. In this section, I will discuss the ethical concerns raised when collecting data on an employee to grade their performance.

Collecting personal data

For most of the metric methods discussed and algorithms to analyse data, there are not many ethical issues raised. When a software engineer submits code to a repository, or even a work database, willingly, the code is usually company property and so they have the right to analyze and collect all data about the repository. However, when it comes to measuring and collecting data about an employee's health or activity level, there is a clear breach of privacy.

The main issue here is how the data is collected and if there is consent from the employee to not only collect this data but also run an analysis on it. In theory, this is a great way to measure overall performance of an employee as there should be some work/life balance. It also means that team managers can see if there are underlying problems with a worker due to lack of exercise or being at a computer all day, which will naturally lower productivity. However, there is an issue here as the employer would impede on the workers home life and set expectations outside of the office.

Analyzing data

There are also major ethical concerns to analyzing data collected from employees. Once an employer has collected data on their workforce, the next step is the algorithms or computations they will run to analyze their performance. Algorithms produced by developers could contain problems leading to workers being judged harsher, but machine learning algorithms also raise the same concern with having errors in their pattern matching and as a result, grade workers poorly.

There is also the issue of not being able to see the inner workings of software engineers. By assessing LOC or commits, we only see a number and not the quality of code and with an AI analyzing the data, we could find more errors in the analysis due to no human supervision. Furthermore, this makes the data worthless and raises more concerns for measuring the performance and classifying it.

References

<https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/>

Y. Qiu, W. Zhang, W. Zou, J. Liu, and Q. Liu. An empirical study of developer quality. In 2015 IEEE International Conference on Software Quality, Reliability and Security - Companion, pages 202–209, 2015.

<https://linearb.io/blog/why-agile-velocity-is-the-most-dangerous-metric-for-software-development-teams/>

<https://www.linkedin.com/pulse/8-essential-speed-metrics-software-development-teams-kasey-jones>

<https://waydev.co/about-us/>