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

CSU33012 Software Engineering - Measuring Software Engineering Report

Vitali Borsak - 19335086

Introduction

Measuring software engineering, how can we achieve this, can this even be done? The answer is yes, although not as simple as measuring business endeavors through the income that they produce or how successful a music album is based on how many copies have been sold. Measuring software engineering has become a recent interest with many companies, teams, and even the engineers themselves due to the rapid take-over of software engineering over most other disciplines.

In this report, I’m going to go through various methods of calculating and measuring software engineering activity, some existing software that allows for this, the techniques and algorithms that enable such software and measurements to exist, and finally, the ethical considerations for such measurements to be performed and the outcomes that it may have.

Measuring Software Engineering

You can analyze and compute software engineering activity in various ways which I will explore more in-depth in this section.

1. Commit Frequency

One very simple and common way to measure software engineering productivity used by various companies is to analyze how many commits are performed by a person and how frequently these commits are made.
This is a very basic and straightforward approach to measuring software engineering, and it might seem quite effective at first, but on second thought, this isn’t enough to measure the efficiency of a software engineer as this alone can be easily bypassed by committing minuscule changes to the code frequently to bring the frequency of commits up or moving blocks of code around.

2. Bug Fixes

Another method to measure the productivity and quality of a software engineer is through the number of software bugs an engineer fixes. This metric paired with commit frequency can show that when commits are made by a software engineer, that they do not just bypass or minor changes to the code, but rather beneficial fixes and improvements to the program.

3. Test Cases Passed

Another of measuring software engineering is the number of test cases passed by a program. This calculates the number of test cases passed after each new commit which shows how impactful the code committed is. This is quite similar to checking the number of bug fixes, which means that if paired with code commit frequency, will show how frequent the code committed by a software engineer is and how impactful the code is.

4. Gamification

Gamification is an interesting approach that has been iterated into many software engineering teams to improve the productivity and overall activity of the engineers. The basic rundown of the concept is that a scoring system is in place in a team, awarding various points to the engineers based on the number of commits and the quality of the code. This is all tallied up and displayed on a scoreboard for the entire team to see. The basic idea behind the concept is that the engineers will see their position on the leaderboard, which will trigger their competitiveness and entice them into being more productive to climb the leaderboard.

5. Code Complexity

The final method of measuring software engineering is how complex the code that has been committed is. This shows the team/ company that the more complex that the code committed is, the more valuable the software engineer is to the project. However, this method, while very effective, has drawbacks as actual code analysis is quite difficult compared to the simpler, code metric approaches discussed above, this is due to the fact that the other methods can be easily measured by other software as they are only metrics, while the quality of code would have to be tested by another engineer. Unfortunately, no software exists to calculate the quality of code, this is due to the limitation that a program can't distinguish between "good" and "bad" code, and even when another person is evaluating the quality of code, their definition of "good" code might differ from the definition of another person. So while this would be the best approach to use in assessing the activity of a software engineer along with their commit frequency, it is quite difficult to achieve, especially software-wise.

Existing Measuring Tools

Now that we have discussed some approaches to measuring a software engineer's activity, let us explore the software that exists at this moment that allows for these measurements to be possible.

1. GitHub

Platforms such as GitHub perform calculations of code commit frequency and capacity and by default expose how often and with what volume you commit code for each day of each month, all you have to do is just navigate to a user's profile and it's one of the first things you see.
This is, however, only a visual representation, and is difficult to apply this to a leaderboard or any other use that a team or a company might have for an engineers commit frequency, this is where the GitHub API comes in. Using an API such as PyGithub, which is the API that I personally used for one of my assignments, allows anyone with the username of the engineer that they wish to gather data on to gather any and all publicly available information of that engineer. This information ranges from, how many commits the person has made to each one of their repositories, what languages the person has used in each repository, the amount of branches, merges, and pull requests made by the engineer, etc. This data can then be saved easily as files in JSON format or in a database such as MariaDB or MySQL, to then be used for statistics through graphical application or gamification.

2. Mobile Phones & Computers

   Since almost every engineer would have a phone and since all of them are working on their computers, this would allow them to access social media or any other entertainment applications or websites such as YouTube or Twitch. Companies can implement a system that calculates the amount of time spent on such applications and try to work around this issue accordingly to increase productivity amongst the employees.

3. CCTV

   Using CCTV footage which is most likely installed in most offices to prevent theft, could improve the productivity of the employees, by understanding the behavior of certain employees through the use of artificial intelligence or another employee. Gathering information on a certain engineer could be beneficial as some employees might be in conflict with others, while other employees might be good friends with each other, which will improve mood and in turn improve productivity. By gathering this data and making changes such as moving certain employees' workspaces to be further apart or closer to other employees may improve the overall productivity of the office.

Possible Computations and Methods

I have discussed a multitude of existing tools for gathering activity data, now we need to explore and talk about the available methods and algorithms that allow the processing of this data in order to get some useful insight on the activity of software engineers in a team or company.

1. Basic Counting

   From all the advanced algorithms that have been derived in the past few decades, basic counting is still one of the most useful and versatile algorithms out there for processing information. Basic counting can be applied to the previously mentioned GitHub API data gathering by counting the number of commits made to a repository, the number of the repositories themselves, the number of pull requests and merges, etc. This data can then be easily displayed in the form of graphs or leaderboards. While basic counting is quite versatile and useful, it’s still quite primitive and is best paired with other methods and algorithms.

2. Machine Learning

   Using artificial intelligence and giving it the ability to learn could result in some fruitful results. We could allow an AI to learn the trends in an engineer’s code commits, bug fixes, test cases passed, etc. which over time, could predict when an engineer will have a coding slump, when they’ll be at the top of their game, or when it’s time to fire them or give them a raise/ promotion.

Ethical Considerations & Other Issues

Arguably some of the methods and technologies for gathering activity data mentioned above are unethical and in this section, I will explore how this is the case.

1. Gamification Issues

   While gamification is an interesting and effective method to improve the productivity of software engineers, it could also potentially be a determent to the employees’ mental health and wellbeing. If you’re constantly trying to compete and push yourself, it might exhaust you which might affect your productivity, furthermore, if this happens, your position in the leaderboard might drop which might cause anxiety, panic, and depression. So although it might boost employees’ productivity and bring them joy when they move up in the scoreboard, occasionally it might backfire and destroy productivity and be damaging to the person’s mental health.

2. GitHub Activity Issues

   When applying to a software engineering job, the majority of the time, you are asked to provide a link to your GitHub account, and since one of the first things you see on a person’s page is how many commits they have made over the last few days of the last few months, this might discourage the employer from moving forward with your application if you have a poor contribution to your account. This might further discourage you from pursuing a career in software.

3. Mobile Phone & Computer Monitoring Issues

While taking action against employees ignoring work and procrastinating on social media or other entertainment platforms is an important task for an employer, monitoring all your employees' actions on their computers and phones is quite unethical as this might reveal sensitive information about the employee that might have been quite private and they did not wish for their employer to know. This might leak any websites and applications that an employee might use that might be for the purpose of dealing with personal issues like dealing with a mental illness, dealing with the loss of a family member, or any sensitive personal interests of the employee.

4. CCTV Monitoring Issues

Although monitoring and analyzing surveillance footage and taking action in order to improve workplace productivity and overall happiness of the employees, constantly monitoring and inspecting people is quite unethical. Eventually, the algorithm will be able to work out the employees' bathroom schedule, typical food preferences, workplace habits, etc.

5. Machine Learning Issues

Even though an employer can receive information that an engineer might have a coding slump and make according to preparations which might help both the employee to not be so impacted by their work problems and aid the employer with still getting the required work completed, it might cause some issues. If the AI falsely predicts that an engineer will have common coding slumps and cause bugs, it might cause them to be fired over something that hasn't even happened and might not even happen, which is quite wrong to fire someone over the prediction of a machine.

Conclusion

There are many ways to gather information about the activity of a software engineer using various methods and tools, then processing said information using algorithms and computations to try and improve the productivity or predict trends of negative productivity in the workplace in the future. The take-away from this is that it is possible to measure software engineering which leads to the possibility of quantifying the quality of a software engineer.

References

1. https://www.getclockwise.com/blog/measure-productivity-development