Measuring Software Engineering

Introduction

For managers in any field, measuring the performance of workers is vital to executing the goals of a project. By measuring workers performance, managers can help identify and solve bottlenecks in team efficiency and better understand the value generated by each worker in the team. Without measuring worker performance in some way, whether it be qualitative or quantitative, it is impossible for management to make informed decisions on which staff to promote, retain or let go and in how to structure teams.

Software engineering faces some unique challenges as a field to measure performance in. Unlike more routine tasks like manufacturing, each task a software engineer faces are highly contextual and specific to the project they are working on. Further, the quality of the code being written is much more important than the quantity of code and measuring the quality of code can be very challenging. However, as software engineers work in digital spaces, data collection of software engineers working habits can be much easier to record.

In this report I will be examining the ways in which the software engineering process can be measured and assessed in terms of measurable data. I will provide an overview of the platforms that can be used to collect this data and perform this analysis. I will look at various algorithms which can be used to measure software engineering performance. Finally, I will examine the ethical concerns of performing this kind of analysis.

Measuring Software Engineering: Metrics that can be used to measure productivity of Software Engineers

Source Lines of Code

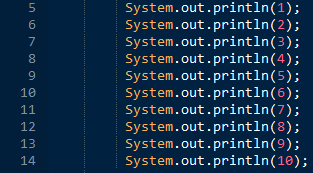
One of the oldest and most used metrics for measuring software engineering productivity is source lines of code (SLOC). This measures the number of lines of code written in the source code as an estimate of the complexity of the program. “The lines of code (LOC) count is usually for executable statements” but can also be used on all lines of code except for comments. [1]

This metric has several advantages. It is easy to automate as lines of code is a physical and measurable entity. It is also a very intuitive metric as it can be visualized and it is an easy concept for management who are not software engineers themselves to understand.

This metric comes with some serious limitations and weaknesses however [2]

1. Functionality is not necessarily correlated with SLOC as skilled developers may be able to develop the same functionality with far less code, so one program with less SLOC may exhibit more functionality than another similar program.
2. Another weakness is that SLOC cannot account for the different amounts of code needed in different languages to create the same functionality. This would incorrectly mark the software engineers who used the language requiring more code as more productive
3. Using SLOC as a measure of productivity may cause software engineers to intentionally write unnecessarily long code in order to appear more productive. This is particularly true if a measure like SLOC over time is used to measure individual software engineer’s performance

Take these two pieces of code as an example of these weaknesses.  

Both pieces of code achieve the same functionality of printing the numbers 1 through 10. The first one however has two logical SLOC while the second has ten. This shows that SLOC is not necessarily linked to functionality. In a firm where SLOC was used as a measure of productivity, a software engineer may be incentivized to write the second piece of code even though it is bloated and makes the code more cumbersome to scroll through. This demonstrates the danger of using such a crude metric for performance evaluation.

Defect Density

Defect density measures the amount of errors in a piece of code versus the size of the code. The standard units used is errors per 100 logical lines of code or errors per KLOC. The greater the density of errors the poorer the quality of the code according to this metric. There is no benchmark for good quality code versus bad quality code but software engineer Steve McConnel stated the industry average defect rate is between 1-25 bugs per KLOC. Defect density is useful for a few reasons. Firstly, it gives an easily measurable reference for the quality of a piece of code. Secondly, it can be easily compared with the defect density of other pieces of code. This gives teams the ability to find modules in their code which need more work. For example, if one module has a defect rate of 1 KLOC and another has 10 KLOC than a team can know to work on the second module. However, this method has a few limitations as well. Measuring KLOC does not tell you the severity of the errors in the code. 2 minor bugs are less detrimental to the quality of a piece of code than one critical bug but would give a higher KLOC score. It also has the limitations associated with using SLOC as a metric. A piece of code with the same functionality and errors but more redundant lines of code will have a lower KLOC even though the same errors exist.

Churn

Churn rate refers to the percentage of a software engineers code output which is a rewrite of previously written code. A high churn rate may indicate poor quality code as it is constantly being rewritten. High churn may also indicate that a team is not working cohesively as developers are constantly having to rewrite their code to fulfil new goals from management or work with other developers’ code. A high or low churn rate is not necessarily good or bad but must be contextually analyzed on a project by project basis. For example, a team using agile development will most likely have a higher churn rate than a team using a cleanroom method for example.

What platforms can be used to collect this data

Flow

Flow is a cloud service which collects and analyses information from source control systems in order to visualize the data and give feedback. On their website, they state “Flow aggregates historical git data into easy-to-understand insights and reports to help make your engineer teams more successful.” Flow presents data to management on both an individual and team level. Flow can analyze individual software engineers code churn, how often they commit to repositories, their experience in different languages and measure various other metrics. It can also identify how members of teams communicate with each other to measure the efficacy of the team.

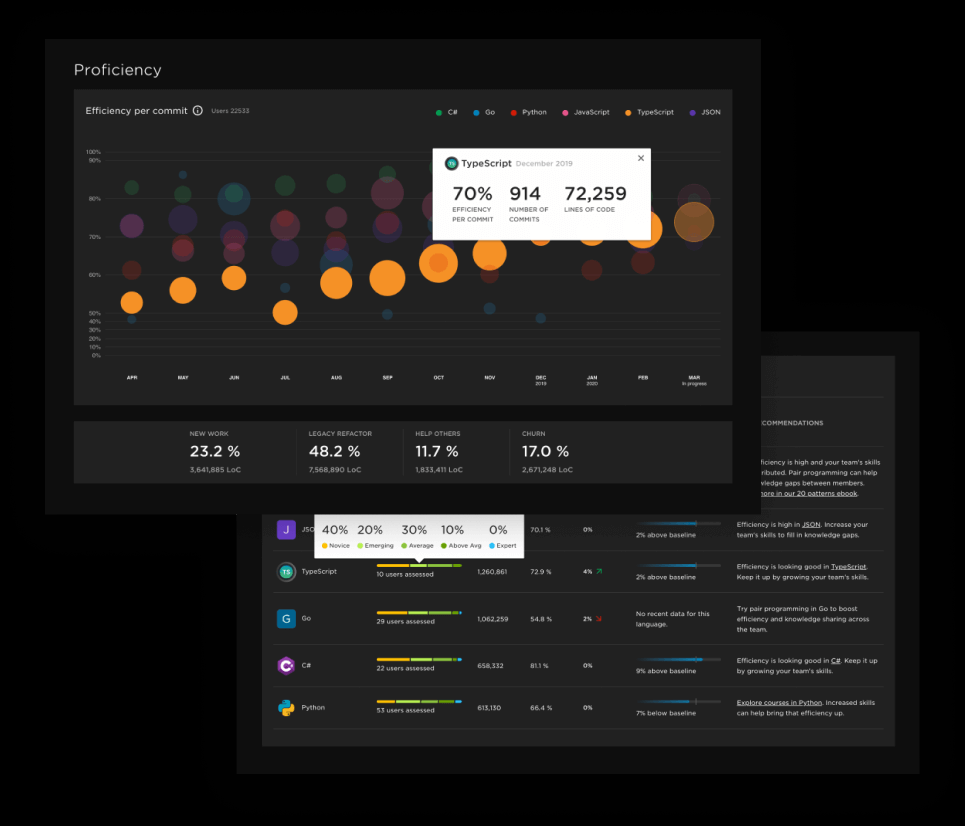


Figure 1 Flow provides an efficiency score for each commit

Code Climate’s Velocity and Quality

Code climates quality cloud service analyses code on GitHub repositories and can offer feedback on the quality of it. It can automatically produce code review comments on pull requests to give easy to understand feedback. It can test diffs before merging code to make sure the code is as robust as possible. It can also analyze which areas of code have the highest churn rate so management can spot potential issues.

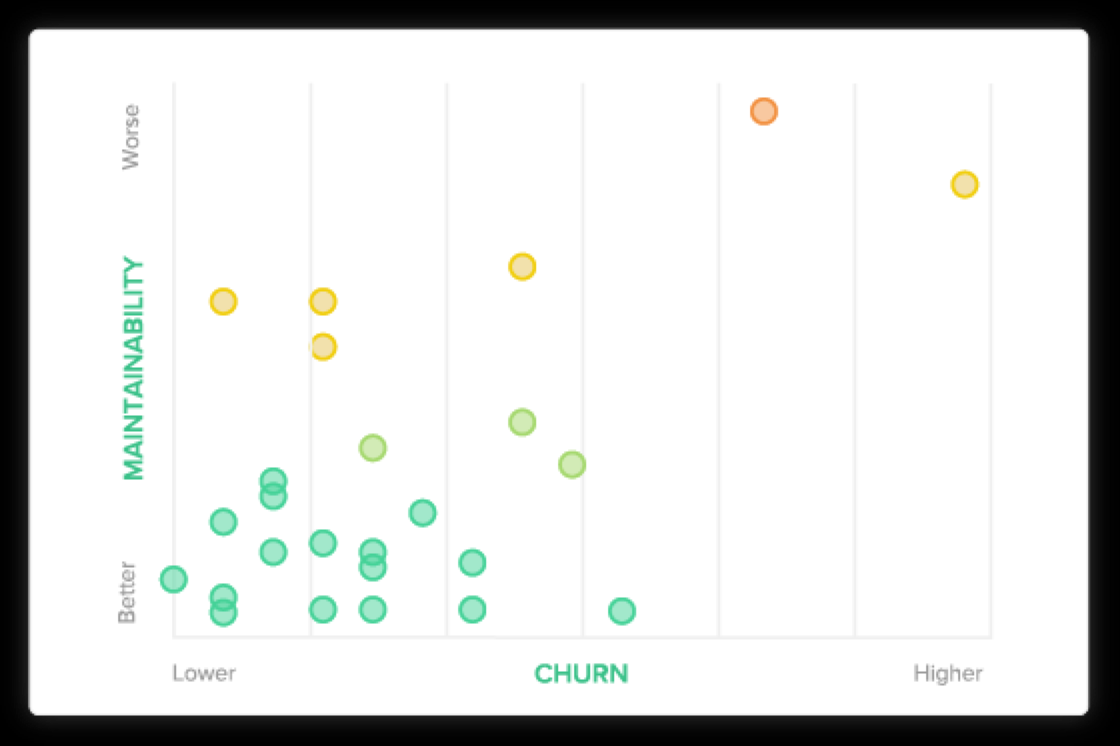


Figure 2 Code Climate graphic measuring churn rate of code against its maintainability

Code climates velocity service can examine workflow within GitHub and give insights into how team workflows operate. It can visualize how often pull requests are merged or how often software engineers are pushing to the GitHub repository.

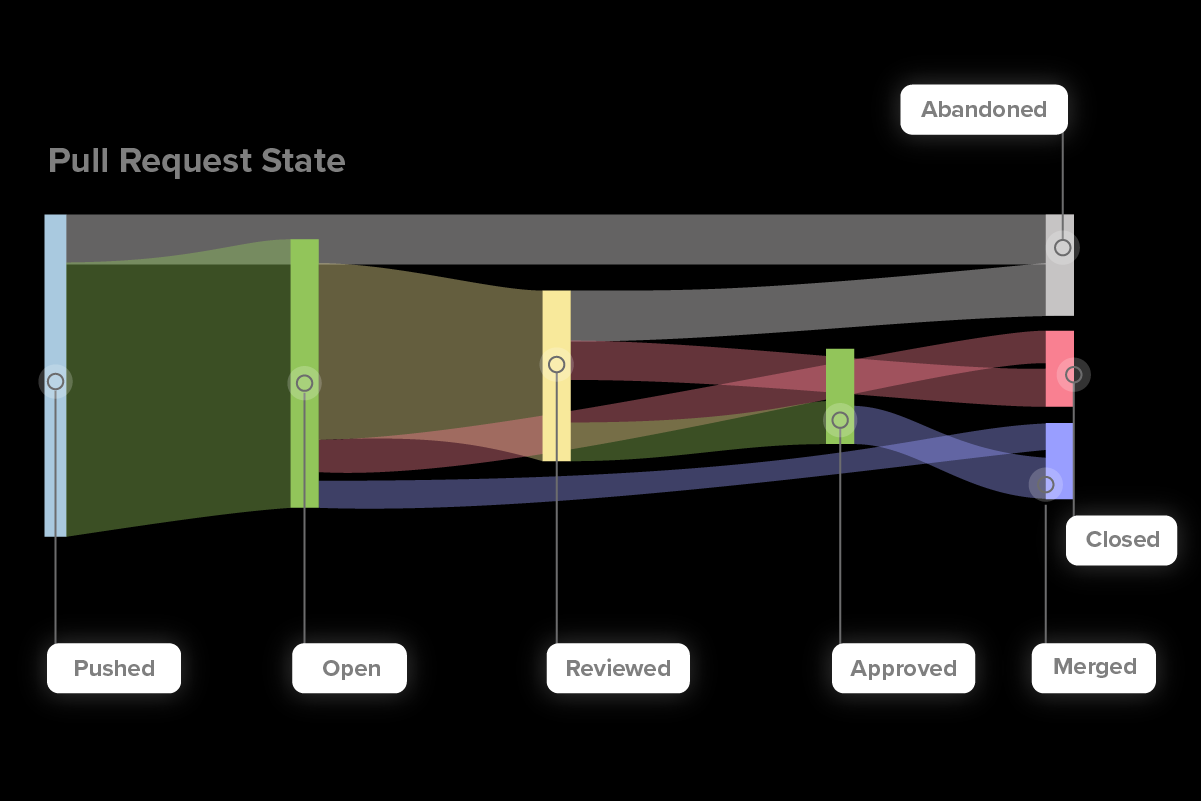


Figure 3: Graphic examining what happens to pull requests

Humanyze Elements

Humanyze has a series of tools to measure not only software engineer’s efficiency but workplace efficiency. These include online software tools as well as pieces of hardware that gathers data on in person activity. One tool called the employee badge can track employee location in the office as well as the tone of voice of the employee badge user. This data can be used to figure track which workers communicate with each other and infer details about employee relationships. Using this data in conjunction with analysis with employees online behavior through emails, messages or other online platforms can give a very comprehensive picture of the social dynamics of a workplace and the team leaders inside that workplace.

Personal Software Process

The personal Software process is a structured software development process designed to help individual developers track the predicted and actual development of their code. The PSP instructs software engineers to input certain requirements for the project and has the engineer measure the actual output of the project against these expectations. These areas of measurement include how long a task takes in minutes, how many errors exist in the product, when tasks were expected to be completed versus their actual completion day as well as the size of modules for the project. The personal software process is intended to be used in conjunction with the team software process in order to facilitate teams to make the best quality products as efficiently as possible.

Ethics of measuring software engineering

Privacy

The main ethical concern in regards to measuring software engineering is privacy. When measuring the performance of a software engineer, large amounts of data about a person must be collected. This can include pieces of public data such as commits on GitHub repositories as well as more private information like emails between employees.

Publicly released work software engineering

Personally I think the collection and analysis of software engineering work an individual has publicly released is ethical. By releasing a piece of work publicly, a software engineer is aware they are allowing any firm or person to look at that code and analyze it in any way they choose. Further, it is hard to decipher any personally information about a software engineer from the code they publicly release online. Finally, the code a software engineer releases is a relevant metric for the quality of work they are likely to produce in the future and is not as likely to be used for unfair discrimination as private information about the software engineer might be.

Social Media Information

I do however have ethical concerns about the collection and analysis of personal information a software engineer might release about themselves on social media or other public forms for several reasons.

Firstly, this sort of information is released publicly for a radically different purpose than for analysis of their productivity as a software engineer. If employers start to analysis this information, individuals will have to change their social media usage in ways which limit their personal freedoms. For example, a software engineer may not want to have any political content on their social media accounts in case this upsets the sensibilities of future employers. Attempts to measure software engineering should not affect the private lives of individuals this way

Secondly, employers may engage in unfair discrimination using information they have gathered online. Employers might look at a prospective employees social media and see that they have a newborn child and decide not to hire them as they believe they may not be as committed to their job for example. There are certain factors in peoples lives which should not be used to discriminate against employees for. The collection and analysis of social media information could aide firms in doing just that.

Health Information/ Interpersonal work information

I would have ethical concerns about the collection of health information or information about interpersonal work relationships even if it is consensual with the employees

The health of an individual is a factor they often can’t control as health issues are often hereditary or happen through misfortune. In many countries such as the US, employees receive their health insurance or health benefits through a company. As a result, employeers may be incentivized not to hire employees or not to renew their contracts if they have certain health conditions or at risk of having certain health conditions as they have to cover the cost of health insurance or the loss in productivity if they leave. This would result in individuals being negatively affected in the workplace due to factors beyond their control which may not even affect the quality of their software engineering. This is something which I believe is unethical

I also believe the collection of data relating to interpersonal relations within offices is unethical. This could include things like employeers reading emails or private messages between employees. As I discussed earlier in the report, Humanyze have begun using tools which can track your location within a firm and register your tone of voice to give insights into the relationship between employees. This is unethical I believe as it violates the privacy of the individuals in the office. Employees should be free to have personal relationships with one another without considering the effect it will have on their job prospects.

Conclusion

In this report, I wanted to explore four topics. How software engineering could be measured. The platforms that could be used to measure it. The algorithms that could be used to measure it and finally the ethics of measuring it. I discovered that the measurement of software engineering has been a task that companies have been struggling with for many years due to its difficulty to measure. However, I learned that there have been many rapid developments in this area recently with many more to come. When these new techniques are developed and begin to mature, I believe the key question will be this. How can we maximize the benefits of these new software engineering measurement tools while maintaining the privacy of the software engineers it measures?

Biography

1. Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
2. Nguyen, V., Deeds-Rubin, S., Tan, T. and Boehm, B., 2007, October. A SLOC counting standard. In *Cocomo ii forum* (Vol. 2007, pp. 1-16). Citeseer.
3. McConnell, S., 2004. *Code complete*. Pearson Education.
4. <https://www.softwaretestinghelp.com/defect-density/>
5. <https://www.pluralsight.com/product/flow>
6. <https://codeclimate.com/velocity/understand-diagnose/>
7. https://www.humanyze.com/elements-2/