Measuring Software Engineering Report

Software Engineering

Aaron Byrne
19334098

Introduction

This report outlines the various ways on how software engineering can be measured to show the productivity or efficiency of engineers. It investigates different platforms that can be used to measure software engineering and goes into detail on the algorithmic and computational aspect on how we can measure software engineering to give us information on the overall performance of an engineer. Finally, the report discusses the ethics and legal or moral issues surrounding the processing of this kind of personal data.

Methods on Measuring Software Engineering

There are many approaches companies/individuals can take to measure software engineering. Many approaches can be inaccurate and misinterpreted and lead to an unfair way of judging the productivity of an engineer. 3 types of software metrics are:

- Formal Code metrics
- Agile process metrics
- Test Metrics

Formal Code Metrics

Formal code metrics measure Lines of code produced by an engineer. This type of metric is considered less useful in modern software development work environments. As a measure of productivity this could lead to rewarding software engineers who write unnecessarily complex and long code that could be done within a few lines. Code that is too long and complex can be hard to understand and can slow the development process down as other engineers could take a long amount of time to go through the code and understand what it's doing. This can lead to disputes between engineers and may eventually lead back to rewriting the code to make it more efficient and less complex than it needs to be.

```
const firstNumb = 100;
let secondNumb;
const secondNumb = firstNumb > 50 ? "Number is greater than 50" : "Number is less than 50";
```

The example above shows 3 lines of code that assigns secondNumb a string depending if the number is greater than 50 or not.

```
const firstNumb = 100;
let secondNumb;
if (firstNumb > 50) {
   secondNumb = "Number is greater than 50";
} else {
   secondNumb = "Number is less than 50";
}
```

This code above does the exact same thing using an if else statement. You can see it's unnecessarily longer. Measuring lines of code would reward the second option even though it does the same thing as the shorter first version.

Measuring lines of code can also depend on the language a developer uses. A commo example is comparing Quicksort in Haskell and C. An engineer can use Haskell's 5-line version of Quicksort compared to C's 30-line version. Measuring an engineer by using lines of code they produce as a metric can lead to further complications and slow down the development process of the software.

Agile Process Metrics

This type of metric measures the lead time of a development team and measures how fast they produce working and shippable software. As a type of measurement lead time can depend on the task at hand and can extremely vary. It would be more suitable to measure the team as a whole rather than individual engineers and will motivate them to work together and more in sync to create software quicker to release to customers. Its common for teams to release smaller updates to keep

clients interested and involved with feedback of the product rather than full all at once product launches.

Test Metrics

Using Unit tests and automated user tests are ways to measure code coverage and how efficient an engineer's code is by making it run through as many edge cases as possible. It can catch defects in the code which is a good way to see if an engineer is performing well or not. Using this type of measurement tests how well the system runs and by passing more and more test cases it shows the quality of the product/software. When engineers commit code, they can be put through tests automatically and will show their coverage and how many tests it passed. However, this type of approach can take time as test cases will need to be developed and it takes time for engineers to develop and discover all edge cases. If they don't realize all edge cases, then the code will be shown as much more efficient than it really is.

Platforms to Measure Software Engineering

GitHub and other software's such as CircleCI are used as a host for software engineering and version control to help engineers collaborate with one another and maintain their projects and ideas that branch from their projects. All engineers would use a platform such as GitHub and companies would want to analyses how each engineer individually performs.

There are platforms that companies can use to gather this information on their engineers.
3 platforms that measure software engineering are:

- Azure DevOps
- Jira
- Pluralsight Flow

Azure DevOps

Azure DevOps is an all-in-one platform where teams can have access to a version control platform as well many other services that help them plan, collaborate and ship code at a faster rate as everything is in an all-in-one application.

DevOps can let project leaders create their story boards to plan, track and discuss their work across their team and other teams if the project is that large.
DevOps connects to GitHub and other Git providers to help engineers build, test and deploy their software.
DevOps is a place where bugs can be documented and future sprints, current and future goals can be accessed, and all engineers has access to their goals and goals of fellow engineers.
All push/pull requests are showed in an easy-to-follow way and all insertions and deletions can be viewed from every single commit.

Azure includes a testing service too. When software has been pushed and committed by an engineer it's automatically run through unit tests where project managers can view the test coverage and how many test cases the code successfully passed. As the development process continues more and more test cases can be

added as the software gets more complex. This will make it harder to track efficiency of code so having a testing service within your version control provider minimizes the hassle in knowing if code is efficient and customer ready or not as time goes on.

All engineers apart of the project are linked to the DevOps service and work items can be assigned to specific engineers. Their workflow is tracked, and project managers can view where abouts the work item is within the pipeline.
DevOps provides an easy way to view where an engineer is within their sprint and how their code is performing against tests.
DevOps includes extensions to connect to other tools such as Slack for example to send notifications of defects in code or successful commits etc.

Jira

The Jira platform is suited to more agile teams. It's main focus is to help software engineers to ship early and often by constructing efficient plans and giving the team to track these plans until the final goal of releasing small incremental updates.

Jira has a reporting feature that gives teams real-time insights displayed through agile reports, dashboards and more. Jira, like Azure gives insights to where the team is in terms of progress during the sprint. It breaks down what has been completed and tells the team where they should be in order to complete the sprint on time and meet their next release.

The engineer's tasks are tracked from when they are started to when they are finished covering all the processes in between like the testing stage. If it fails then back to development.

Pluralsight Flow

Pluralsight Flow is another platform similar to Jira in the way it shows git analytics and displays them in easy-to-read charts, graphs and reports. This platform collects data from version control services such as git. The data it collects ranges from commit history to tests it has passed to how many lines code an engineer wrote to pull requests. There's countless data for teams to utilize to create a more efficient process of development.

Pluralsight displays this data in a helpful and fair way to teams, as it can lead to confusion and which engineers are performing to their standards. Pluralsight flows main aim is the stop project managers from counting code and instead see how much time is spent refactoring legacy code vs. new work. Recognize project bottlenecks and remove them. Get concrete data around commit risk, code churn and impact to better guide discussions.

With this it collects great data on commit efficiency by language and can give recommendations to your team on how to improve their skills and workflow.

The market for these platforms is very competitive and saturated. All platforms can measure very similar metrics. Everyone has access to a git API so project managers can always receive the data they want to see specifically and will cut the need for any of these external platforms. In my opinion, I feel like Azure would bring the most value to companies. Everything is all in the one platform and the data managers can see is enough to evaluate a worker's performance.

Algorithms used to Measure Software Engineering

Algorithms are needed to use the data collected and represent this data that can measure the performance of an engineer. Two algorithms I have researched are:

- Halstead complexity measures
- Functional Point (FP) Analysis

Halstead Complexity Measures

Developed by Maurice Halstead, his goal was to identify measurable properties of software, and the relations between them. This is similar to the identification of measurable properties of matter (like the volume, mass, and pressure of a gas) and the relationships between them (analogous to the gas equation). Thus, his metrics are actually not just complexity metrics.

This measure is to be done before product release and the algorithm allows you to measure testing time of source code.

Parameter	Meaning
n_1	Number of distinct operators
n_2	Number of distinct operands
N_1	Number of operator instances
N_2	Number of operand instances

Above are parameters and their meanings when it comes to measuring the complexity of software. Below shows how we can use these parameters to get different meaning and show the complexity of what an engineer has produced.

Metric	Meaning	Formula
n	Vocabulary	$n_1 + n_2$
N	Size	$N_2 + N_2$
V	Volume	$N * \log 2\ n$
D	Difficulty	$n_1/2 * N_2/n_2$

E	Effort	$V * D$
B	Errors	$V / 3000$
T	Testing time	E / k

Vocabulary is the total number of unique operator and unique operand occurrences. The size is the total number of operator occurrences and the total number of operand occurrences. Volume is proportional to program size, represents the size, in bits, of space necessary for storing the program. The difficulty metric just shows how hard it is to manage and run the program/software. Effort is proportional to difficulty and volume and it just measures how hard it is to translate into a program language. Testing time Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.

Functional Point Analysis

This is a type of analysis for agile development/process. It's used to measure productivity, quality, estimation and change of scope. It can answer how productive a team is and how good a team's software is, and how much effort should it take to create the software and measures pricing (total cost of the development process).

Productivity is measured by functional points (FP's) divided by project hours.
How good the software is measured by Total Production Defects ÷ Project FPs.
How much effort should it take (how many hours to deliver) is measured by Project ROM FPs ÷ historical project productivity rate.
Finally, how much it should cost is: Project ROM FPs * Historical $/Project FP.

Ethics

Measuring the productivity of a worker in any industry can be a sensitive topic and could always lead to legal issues with the employer and employee. Adding value to a company/project can happen in many ways and sometimes it is unmeasurable. Engineers in a team could be there to motivate or be there for junior developers to learn from. They may not necessarily add many lines of code or commit very often play a pivotal role in the success of the project.
So, measuring these developers by looking at data may lead conflicts that can add tension within a team, resulting in the project being delayed.

Gathering a lot of data on someone concerns many people and many are sensitive about this ongoing discussion. If a team knows they're being watched and monitored, they may always be on edge and trying too hard and not enjoying the work they are taking part in. However, this may come as an advantage to some teams, and they will get more out of their engineers but it's certainly not in an ethical way.
This can also be reversed, as someone may add a lot to the actual software and perform amazingly according to the data but may negatively affect everyone else and be the cause that under workers are under performing. Numbers don't always paint the correct picture.

Using this data may result in a worker being unfairly dismissed and can bring legal actions to the company involved. It may work to a company's advantage to not measure the software engineering and trust the developers at hand to create some software within the required timeframe and reward them through meeting time goals and not rewarding them on what the numbers say.

Conclusion

This report has discussed that there are countless ways to measure software engineering. Many version controls such as git give us access to user data such as commits, insertion and deletions and countless other bits of data. We can visualize this sort of data the see how productive an engineer is on a certain project. There are many advantages to this but in my opinion and my conclusion is to not use this data. It adds pressure to developers to over perform or to even just to work to make it look like they are performing well. Engineers may not be performing because they like what they're doing and they're just working to please the system, and this would lead to poor releases of pieces of software.

The use of platforms like Azure DevOps can measure the work engineers do, but it does so in an ethical way and not displaying data on systems that a developer can easily work around.
Other workers in industries cannot be measured in this type of way so it feels unethical to compare engineers on the same team's performance. Leave it to the developers to see efficient results.

References

AgileConnection. 2022. Agile Development and Software Metrics. [online] Available at: <https://www.agileconnection.com/article/agile-development-and-software-metrics> [Accessed 3 January 2022].

GeeksforGeeks. 2022. Software Engineering | Functional Point (FP) Analysis - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/> [Accessed 3 January 2022].

GeeksforGeeks. 2022. Software Engineering | Halstead's Software Metrics - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> [Accessed 3 January 2022].

Pluralsight.com. 2022. Pluralsight Flow | Overview. [online] Available at: <https://www.pluralsight.com/product/flow> [Accessed 3 January 2022].

2022. [online] Available at: <https://azure.microsoft.com/en-us/services/devops/#overview> [Accessed 3 January 2022].

Atlassian. 2022. Jira Reports | Atlassian. [online] Available at: <https://www.atlassian.com/software/jira/features/reports> [Accessed 3 January 2022].

Sealights. 2022. Top 5 Software Metrics to Manage Development Projects Effectively. [online] Available at: <https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/> [Accessed 3 January 2022].

Getclockwise.com. 2022. How to Measure Productivity in Software Engineering | Clockwise. [online] Available at: <https://www.getclockwise.com/blog/measure-productivity-development> [Accessed 3 January 2022].