|  |
| --- |
| A picture containing text, computer, electronics, computer  Description automatically generated |
| Measuring Software Engineering  CSU33012 |
| |  |  |  | | --- | --- | --- | | Ailbhe Merriman | 12/29/21 | 19335244 | |

**Introduction**

In this report I aim to answer why we need to be able to measure software engineering, how and what ethical considerations. Employers want and need to know how efficient their engineers are and in what ways they can improve. Managers need comprehensive measurement tools in order to be able to give clients reasonable deadlines that their team of engineers are able to meet. Without these measurements that I will outline throughout this report, it would be very hard for companies to see where a team or individuals may be slacking off and not pulling their weight, which could lead to problems within teams or the company. It is certainly in the company’s best interests to have a system in place to measure overall productivity and to be able to see places where improvement is necessary.

However, measuring this productivity is not an easy feat because there is no one metric used in software engineering. People do not work in one way, people in every field have different measures of productivity and success and that is true for those involved in the software engineering field. It is a creative field as much as it is logical and as such there are many solutions to take an engineer to the destination of their project. One solution could have 100 lines, the other 500 and the end product the same. In this report I will highlight the ways in which we currently measure software engineering such as the measurable data like line count, hours spent working and code quality. I will identify the platforms used to measure this data and lastly, I will explore the ethics concerning measuring software engineering productivity.

**Approaches to Measuring Software Engineering**

* **Line Count**

The most obvious way one would think to of measuring an engineer’s productivity would be to look at their code line count over a period of time and perhaps compare it to their peers. In theory this sounds like a plausible way for a company to determine the value of an employee, but this method really is flawed. Firstly, this method could potentially reward inefficient code that is long and produces many lines of code but could really be executed with fewer lines of more efficient code. Similarly, different languages will produce more lines of code than others, even if they produce the same effect. For example, Java and Python, developers working with Java will most likely write more lines of code while producing the same effect as that of a developer working with a less verbose language such as Python. Another flaw of using code line count to determine productivity is in the case of a developer whose role is to maintain a codebase rather than create and design new code. Their code line count would be significantly lower than someone who is writing and implementing new code daily, yet they likely would be working as hard.

* **Commit Count**

How many commits a developer makes throughout a period may seem like a reliable method to assess their productivity but this method, like code line count has a major flaw. This way to measure software engineers is easy to exploit. A developer could exploit this system of measurement by committing after every tiny change to make it look like they are being productive.

* **Hours spent working**

I think this is an interesting way one could measure the productivity of a software engineer. Based off previous projects completed, it would be possible to set a minimum or maximum time that an engineer should spend on a particular project. If they go above or below these specified times it would be interesting to see how the quality of their code is affected. While I don’t believe this is a totally comprehensive way to measure an engineer’s productivity, I do think that it has some merits. If a developer takes significantly more time to complete code than their peers then the quality of their code should be better, in theory. If they take significantly less time, what quality is being given up in order to allow for quick turnover? What if engineers were asked how long *they* believed it would take them to complete tasks? Having engineers do this and when they complete the task, asking them to reflect on if they succeeded or not and why could be a really interesting approach to measuring these developers. It is also a be a benefit for the company and the individual to have these self-reflections, as they could lead to more accurate time frames to give to clients.

* **Code Quality**

Following on from hours spent working on code and its affect on the code’s quality, another way one could measure an engineer is by taking a look at the quality of their code. Code quality is hard to standardise and can mean different things for different companies and individuals. According to an article online by Perforce there are five main traits of good code, reliability, testability, maintainability, reusability, and portability[[1]](#footnote-1). Reliability refers to the probability that code will run and be able to be used without it failing, this is obviously important as code that fails often is not very useful and can cause problems for clients. Testability measures the testing capabilities of code, measured by how many tests you must run in order to find issues with the software. Maintainability is important as code should be easily understood by other engineers so they can continue to work on and be able to fix problems in the software. Portability refers to how easy the software is to use in different environments, this can be ensured by regular tests on different platforms. Finally, reusability is important to measure code quality as it is beneficial if code can be repurposed and reused as needed. While measuring code quality takes more time than the aforementioned measures, used in conjunction with hours spent working, I do believe that it may be the fairest way to measure a software engineer’s productivity level in comparison with peers.

* **Measuring productivity within teams**

For individuals, the above methods are adequate but as one can imagine, most software engineers working for bigger companies work with teams. In an article by Better Programming[[2]](#footnote-2) they outline some ways in which it is possible to measure the efficiency of a team, they include time to delivery, quality, uptime, and speed of delivery. Time to delivery refers to the lead time and cycle time of a project. Lead time is from the client’s point of view and is concerned with the time between the request and the delivery of the product while cycle time is from the developer’s side and is the time elapsed between getting the request and having delivered it to the client. If a team can meet their client’s expectations in terms of lead time, this would definitely be a sign of a productive team. Quality refers to the defect rate and percentage of test coverage. The less defects in the software and the higher percentage of code that is testable is a sign of good quality code. Uptime is the same as the reliability of the software, in particular the latency of the code. Latency refers to the time taken for a packet to be transferred from one point to another. Speed of delivery can refer to the team velocity, average age of closed bugs, number of releases, and number of story points completed.

**Platforms Concerned with Measuring Engineers**

Development analytics tools are a great way for companies to keep track of their employees who are developers. These are applications that provide real-time information on the software development process using historic data. Engineers use these tools to gather information about barriers, daily and weekly advances, work habits and trends. An article by WayDev[[3]](#footnote-3) outlines the competitors in this field and what exactly they do. I will discuss some of the apps that I find interesting below. The main characteristics of these development analytical tools are integration with one or more repositories, showcase trends and statistics, provide aggregate data of a team’s data in one place.

* **LinearB**

LinearB is a development tool that allows managers to see insights into how well one of their software engineering teams is faring. The key metrics used to measure the productivity if developers are cycle time, deployment frequency, lead time, and time to release. Cycle time, like mentioned before, is the time between when the work was started and when it was completed. Deployment frequency refers to the number of releases over a specific period. Lead time, again like before, is the time elapsed from when the work was requested and when it was delivered. Time to release is the amount of time between when a pull request was merged to the release. 371 users of LinearB reported seeing a 38% reduction in code churn and a 28% increase on feature efforts[[4]](#footnote-4). LinearB also has capabilities to identify outliers in teams and flag high-risk engineering activities. Although, they don’t support API integration and don’t have many executive-level reports available.

* **Pluralsight Flow**

Formerly known as GitPrime, Pluralsight Flow is a popular Git analytics tool, which provides insights on how developers work and what areas need improving. Their key metrics that they provide include churn, impact, influence, and reaction time. Churn referring to code that is changed or rewritten within 21 days of its initial commit. Impact is a metric that involves the complexity of the code that has been changed and how it effects the rest of the software. Influence is in reference to the amount of follow-up commits that can be seen after a review to a pull request. Reaction time is the amount of time taken for a reviewer to reply to a comment addressed to them. This app is an industry favourite with testimonies on their website claiming that they noticed a 20%[[5]](#footnote-5) increase in impact to codebase after beginning to use Pluralsight Flow. Companies such as Google Cloud Platform and Amazon Web Services use this app.

* **Jellyfish**

Jellyfish is a similar app to the two mentioned previously, with some different key metrics that they follow. They key reports that Jellyfish use include a productivity report, an alignment report, and an investment distribution report. According to their website, the productivity report helps users understand whether the work of their engineers is miscategorized or if they spend time on low priority work. The alignment report allows managers to see how management decisions influence resource allocation and the net gain for the company. The investment distribution report showcases how time is spent on different aspects of the business. While Jellyfish provide interesting insights into effort made by individuals and a high-level perspective of the engineering activity it is focuses quite a lot on other aspects of the business unlike the other measurement platforms I discussed earlier. Companies like Toast use Jellyfish to measure the productivity of their software engineers[[6]](#footnote-6).

While these options are very comprehensive and can give some companies what they are looking for, with increased desire for data privacy some companies and employees may find it invasive to give data to 3rd parties like this. Some of the large tech companies like Facebook and Microsoft have their own measurement tools for measuring their employee’s productivity. I believe from what I’ve learned throughout the course for researching for this report, it’s interesting to see the available tools and how companies get value form them. The statistics to be found on these app’s websites are so significant that I don’t think we can discredit the merit of tracking productivity like this.

**Data Computation**

* **Function Point Analysis**

An interesting way of data computation that I came across was the Function Point Analysis (FPA) that is used during agile development[[7]](#footnote-7). It uses simple equations to answer basic questions like: How productive is our team? How good is the quality of our software? How much effort is required? How much should this software cost? By analysing the data to determine the impact of quality, productivity, schedule, and cost, software development organizations may be better able to choose the most appropriate development methodology for their projects.

|  |  |  |
| --- | --- | --- |
| Measurement Question | Agile Measures (by Sprints) | Final Delivered Project |
| How Productive are we? | Sprint FPs ÷ Sprint Hours | Delivered FPs  ÷  Total Project Hours |
| How good is our Software? | Sprint Production Defects ÷ Sprint FPs | Total Production Defects ÷  Project FPs |
| How much should it cost? | Sprint ROM FPs \* Historical $/Sprint FP | Project ROM FPs \* Historical $/Project FP |

* **Reliability**

Reliability is “the probability that a system performs correctly during a specific time duration”[[8]](#footnote-8). This means that during the specified time no repair is needed or performed, and the software follows the performance specifications. Reliability follows an exponential default law, which means that it is reduced as the time considered for reliability calculations passes. Since the failure rate may not stay static over time the average time-based quantities like Mean Time To Failure (MTTF) and Mean Time Between Failures (MTBF) can be used to calculate reliability of software.

* **Availability**

Availability is another way one can measure software engineering[[9]](#footnote-9). It is defined as “the probability that a system performs correctly at a specific time instance”. The software must be operational and adequately satisfy the defined specifications at the time of usage. Availability is measured at a steady rate and accounts for potential downtime that would make the software unusable. The calculation of availability determines the performance of a component at a specific time based on the time elapsed between its failure and recovery. It uses the formula for Mean Time to Recovery (MTTR).

**Ethical Considerations**

I think the line must be drawn between micromanagement and optimisation of work in the tech field. Workplaces where people are over managed and leaned over don’t work. I believe that when leaders begin to track an employee’s screen in real-time or track all the websites that are used during the workday, the line has surely been crossed. I think that the tools mentioned above really can be useful but at the end of the day I do believe that a human should have the final say in whether an employee is an asset to the team or company they work for. Because humans are such complex beings it’s impossible to dilute them to a few statistics at the end of the month or year and the value they bring could be so much more than the lines of code they contribute. So, determining true value to the team, I’m not sure can really be done solely based on these methods for measuring software engineering. Are they good and fair methods to use to see the places in which the company and team lack in terms of actual productivity, yes, I think so?

However, what does this data really mean when a person is aware they are being tracked all the time and what can this do to a person? On the one hand, people have a job to do and really if someone is not contributing to the company or team over a significant period of time, perhaps they are in the wrong job? On the other hand, I do believe if a person knows they are being monitored it could lead to burnout and dissatisfaction with their job, which in turn could lead to poor results and then the cycle would continue. This sense of managers and companies watching engineers and determining their value based off these metrics can lead to a negative work atmosphere. It could potentially lead to unwanted competition between colleagues and ultimately an unproductive workplace, which is the exact opposite of what they are trying to achieve.

From my reading, it does seem like these technologies are already widely used meaning that software engineers do not really have a choice if they want to be monitored like this. I don’t think it would be feasible to give up on a job and simply move to another company who don’t use these metrics. I think that this is where the problem lies.

**Conclusion**

I think at the end of the day, we are still a way off having a comprehensive and complete measure for software engineering that encompasses the true value of an individual to a team or company. While some of the metrics do have obvious merit and combined with a more human approach to determining value, they would be very useful. They do have significant and inexcusable flaws that have been outlined above. Software engineering is a field that truly just has too many variables to be fairly judged in this manner, it is a creative field but also very logic driven which I think is quite unique to software engineering. To conclude, while I really believe that these metrics are useful, I reiterate that they are not fully capable of determining a software engineer’s value or worth.

1. <https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality> [↑](#footnote-ref-1)
2. <https://betterprogramming.pub/how-to-measure-performance-of-engineering-teams-8da1d0975a7c> [↑](#footnote-ref-2)
3. <https://waydev.co/5-gitprime-competitors-what-development-analytics-tools-are-leading-the-way/#pluralsight-jumplink> [↑](#footnote-ref-3)
4. <https://linearb.io/> [↑](#footnote-ref-4)
5. <https://www.pluralsight.com/product/flow> [↑](#footnote-ref-5)
6. <https://jellyfish.co/solutions/performance-productivity/> [↑](#footnote-ref-6)
7. <https://www.agileconnection.com/article/agile-development-and-software-metrics> [↑](#footnote-ref-7)
8. <https://www.bmc.com/blogs/system-reliability-availability-calculations/#> [↑](#footnote-ref-8)
9. <https://www.bmc.com/blogs/system-reliability-availability-calculations/#> [↑](#footnote-ref-9)