Kathleen Garrity

Professor Barrett

CS3012 Software Engineering

8 November 2019

Measuring Software Engineering Report

**Introduction**

In the field of software engineering, the topic of software engineer productivity tracking and measuring has been hotly debated. Specifically, there are varying opinions as to whether the productivity of a software engineer can actually be measure or not. Some claim that software engineer productivity cannot be measured because productivity is typically measured by an individual’s ability to produce objects, but software engineering is not a simple production line.[[1]](#footnote-1) Others argue that since the development process is complex that involves making thousands of little decisions, there is no one single objective metric that can fully represent the productivity of a software engineer, and thus, there is no possible way to measure this productivity.[[2]](#footnote-2) While many objective metrics do exist to measure engineers, some researchers argue that once engineers know what is being measured, it is easy for them to adapt their code to game these objective measurements.[[3]](#footnote-3) Still others suggest that it is more effective to measure job satisfaction or happiness because these both correlate to increased productivity.[[4]](#footnote-4) While it is true that software engineering is a complex process, it is easy for me to look around at my classmates or my coworkers and see that some people are more productive engineers than others. Just because measuring software productivity is complex does not mean it is impossible and there are a variety of ways of going about it. In this paper, I will first review what aspects of software engineering are measurable. I will then describe a number of computational tools available for analyzing large datasets that exist for measuring software engineering. I will next discuss algorithmic approaches used to analyze software engineering and the implications that using such algorithms has on the future of work. Finally, using the information I have provided as a reference, I will make a judgement as to the ethics of gathering and analyzing objective data about software engineers and workers in general.

**Measurements**

Even those who argue that software engineering productivity cannot be measured agree that some aspects of software engineering are measurable. We will take a look at different quantitative, qualitative, and process metrics that can provide some information about the productivity of a software engineer.

There are a number of quantitative metrics for measuring software engineering. One of the most obvious quantitative metrics is lines of code (LOC). There are a variety of different versions of this measurement that can be considered. For example, it is possible to measure all lines of code, all non-blank lines of code, or even all non-blank, non-commentary lines of code.[[5]](#footnote-5) These distinctions draw our attention to an important idea to consider; that is, it would be very easy for programmers to mislead their employers with this metric by filling their code with blank space. To get a more accurate depiction of the code, we can instead measure a coding solution in terms of effective lines of code. This only includes lines of code that contain programming statements. However, measuring productivity in terms of effective lines of code becomes complicated when we considered code modifications made to an extensive pre-existing codebase.[[6]](#footnote-6) In this case, the work a programmer has to put into understanding the original code and reworking it is often not proportional with the amount of code they write. It is also important to note that measuring software engineer productivity by lines of code may be misleading because the most effective and efficient solution often requires fewer lines of code.

We can also measure the number of hours worked. However, this measurement is often problematic as a stand-alone representation of how much an engineer is working as not all time in work is necessarily spent actively coding. Also, studies have shown that past a certain threshold, an increase in working hours actually leads to a decrease in productivity. We can also quantitatively measure the number of bugs an engineer introduces into code or the number of bugs an engineer fixes and how long a fix takes. Other quantitative metrics include number of commits, pull requests, or external libraries used.

We also can measure a number of qualitative aspects of software engineering. One qualitative aspect is performance. This is a measure of how long key specific use cases take to execute on the front end for users.[[7]](#footnote-7) A few other qualitative metrics include customer problems and satisfaction, number of tests passed, and code coverage, or the percentage of the lines of code that are actually executed during the testing phase.[[8]](#footnote-8)

For measuring qualitatively, there are also a number of metrics associated with defects and defect correction. For instance, it is possible, and important, for teams to measure escaped bugs, or bugs that are found by the user. Another defect measurement is defect removal efficiency, which is the measurement of how efficient engineers are at removing bugs before a product is released and is often tracked as a ratio of bugs fixed before release to bugs found after release. To track over a smaller increment of time, a team can measure the number of new bugs compared to the number of closed bugs. This gives a better sense of the health of the software a one particular moment in time. You may also want to measure the percentage of high priority defects to see if the bugs in a codebase are mostly minor or major faults.[[9]](#footnote-9)

Because software engineering is such an intricate process, it can sometimes be easier to track it with process metrics. This is specifically applicable for engineering teams that are using the Agile approach and Scrum framework in their method of software development, as this method involves breaking the software development process down into smaller, more focused processes. One process metric specifically related to the Scrum methodology is burndown rate. Burndown rate is a measurement of how many development tasks are complete over time. Burndowns are usually measured in 2-week periods called sprints.[[10]](#footnote-10)

We can also measure the process in software engineering in terms of story points and function points, both of which are important features in agile development. Story points describe the measurement of how much work a team thinks a certain function or feature will take to build

while a function points are a standard measurement that quantifies features with respect to the whole project.[[11]](#footnote-11) Both these measurements can give teams a better sense of how well they are meeting their project deadlines.

Another process metric is velocity. In this case, velocity is defined as the average amount of work a scrum team completes during a sprint.[[12]](#footnote-12) This can give the team a better idea of if they are on track to hit their deadlines. It also makes it easier for managers to see who is completing more, less, or the average amount of work over a finite period of time.

Code churn is another process metric. Code churn is measurement of how code evolves over time. Most projects should see high churn at the beginning of a project as an engineer starts to explore how to implement new features and then should see churn level off towards the end of the project as the engineer has built the main features and only has to focus on making minor tweaks and bug fixes.[[13]](#footnote-13)

A final process metric is cycle time. This is the time it takes to full carry a task from beginning to end.[[14]](#footnote-14) Measurements of cycle time can not only show you how long the entire development process takes, but also can measure the time an engineer spends in each cycle of production, such as coding, defect correction, code reviewing, etc. This can help teams see where they are spending most of their time and also allows them to see if a certain process, a certain bug fix for example, is taking longer than usual.

There are also a handful of metrics that can be tracked, but don’t fall neatly into one of the three categories discussed. These include overall cost, engineers’ interactions with others, communication on message boards or via email, content of comments, and more. It is also possible to track physical aspects of engineers such as heart rate, daily movements, energy levels, and content of office interactions with wearable technology.

**Computational platforms**

As we have seen, there are numerous aspects of software engineering that can be measured and large amounts of data that can be collected with regard to the software engineering process. However, if we are to actually derive value from this data, we need a way to analyze it and understand its trends. There are a number of both paid and open source tools available to help both engineers and organizations make sense of software engineering data.

One such tool for measuring engineer performance is GitClear. The goal of GitClear is to “analyze Git repositories to calculate team output and actionable opportunities.”[[15]](#footnote-15) It boasts performance metrics that can show managers who is doing what. GitClear also has a feature that showcases developers’ strengths and helps “identify subject matter experts and assign projects to the most qualified developer.”[[16]](#footnote-16) For analyzing code, GitClear uses its own specific Line Impact metric. By ignoring activity occurring in libraries and auto-generated files, adjusting for large but simple insertions or deletions and discounting churn and simple, ineffectual code shifting,[[17]](#footnote-17) GitClear is able to provide a clearer picture of how much work a developer is actually doing. GitClear offers two paid versions of its product with slightly differing levels of features. For the basic pro package, GitClear starts at $295 per month.

GitPrime is another popular code analysis tool. GitPrime gathers data from Git and analyzes it to show teams and individuals where they are working efficiently and where there is room for improvement. With GitPrime, managers and engineers have a comprehensive view of a project’s code commits and ticket activity in one dashboard, making it easier for them to visualize their progress and how well they are hitting their deadlines. The Work Log feature offered by GitPrime provides a view of the major features that engineers produce in a given time period.[[18]](#footnote-18) This tool also boasts a Dev Snapshot feature which offers managers a view of the work patterns of each developer on their team. There is also a Snapshot Quadrant which files engineers into different quadrants, enabling them to see which engineers are weaker or struggling on a certain project. GitPrime also records developer coding days, commits per day, impact, and efficiency (as the ratio of churned code to contributed code) and helps track collaboration. It is available for free for a 15-day trial and then it costs $749 per month for the most basic levels, with two more expensive levels that provide increased functionality.[[19]](#footnote-19)

Another option for analyzing software engineering data is Code Climate’s solution, Velocity. Velocity takes information aggregated from commits, pull requests, and code, and helps teams and engineers better visualize their strengths and weaknesses. It offers managers the capability to automatically send team members alerts and notifications to help them stay on top of work when it looks like they are falling behind. Velocity is a paid service, with its cheapest plan starting at $1759 per month.[[20]](#footnote-20)

CAST is another software application that can be used to garner information about the productivity of engineers. CAST considers itself “like an MRI for software”[[21]](#footnote-21) in that it captures an image of the internal components of software. It also claims to provide “an ‘as-is’ architecture blueprint where one can navigate the system, simulate change impact, and spot monstrous flaws or faulty construction.”[[22]](#footnote-22) IBM has used CAST to analyze a developer’s work and make quantitative judgements about the developer’s skills. The application has also shown managers at IBM which developers may need to improve certain skills or overall performance. As one IBM manager reported, “Essentially, it permitted our people to walk around with a scorecard.”[[23]](#footnote-23)

Another tool for analyzing software engineers is Unified CodeCount (UCC). UCC is a free tool that can analyze code in multiple different programming languages. It’s main feature is that it provides users with a tool to compare and collect the differences in the effective lines of code between two versions of a codebase. This tool notes when code is added and modified and provides information about LOC counts, comment counts, and keyword counts.[[24]](#footnote-24) While this tool doesn’t provide users with as much information as GitClear, GitPrime, or Velocity, it does offer a basic analysis of code entirely for free.

C and C++ Code Counter (CCCC) is another simple free solution. Originally created to gather data from C++ files., its most recent version has expanded its processing abilities to include Java source code as well. CCCC provides information about a number of code characteristics such as lines of code, McCabe’s Cyclomatic Complexity (used to detect code which is likely to have errors, number of modules, comment lines, weighted modules per class, and more.[[25]](#footnote-25) For each of the measures mentioned, CCCC implements specific algorithms to calculate them.

SourceMonitor is a slightly different code analysis tool in that it is meant to be used by individual developers to help them improve their own code. SourceMonitor can process a handful of languages including C, C++, C##, and Java and saves metrics in checkpoints to allow for historical comparisons during development projects. SourceMonitor helps engineers visualize trends by displaying the data in charts and tables.[[26]](#footnote-26)

For tracking potential problems in a codebase, NDepend is also a viable solution. NDepend tracks code smells, or potential problems with code that could end up creating a lot of work for engineers in the future. It can provide accurate technical debt predictions to help save developers time that would be lost repairing and debugging code. It also has features for tracking LOC, cyclomatic complexity, nesting depth and rank. NDepend offers a 14-day free trial before you have to switch to the paid plan for $399 per developer for a 1-year subscription.[[27]](#footnote-27)

Clearly there are a wide number of computational tools available for analyzing data about software engineers. Different tools provide different levels of analysis on different metrics, so when choosing a tool it is important to make sure that that tool provides analysis on the metrics that you and your company are actually interested in.

**Algorithmic approaches**

While there are a number of tools available to perform the analysis of the software engineer data easily, there are also a variety of algorithmic approaches to analyzing this data.

One complex algorithmic approach involving machine learning is clustering. Clustering involves partitioning data into groups so that the members of a group share similar qualities while the members of different groups are dissimilar. This can help provide a better understanding of how data is distributed and can also be helpful for identifying outliers. In the case of measuring software engineer productivity, using clustering to visualize the data could show which engineers are significantly strong or significantly weaker.

Another algorithmic approach is classification, which involves assigning data points to a predefined set of classes. This approach relies on a function that maps a data point into one of the several predefined classes. Decision trees can be used in classification to help sort the data points. Through pattern and association rules, classification can help reveal underlying relationships between different attributes in a data set. For example, in the context of software engineer productivity, classification could reveal a correlation between the number of bugs introduced by a developer and the number of lines of code written by that developer that would have gone unnoticed in a basic analysis of the data.

To analyze the natural language textual elements of code, a summarization algorithm can be used. A summarization algorithm can shorten long streams of text to outline just the main ideas.[[28]](#footnote-28) Summarization algorithms would also be helpful for analyzing communication data between engineers via emails or message boards as well.

Another complex algorithmic approach is to use neural networks such as a Neural Hidden Markov Model. Neural networks are trained to recognize patterns and a Neural Hidden Markov Model has even been used to predict the probability that a developer was coding for each minute of their recent history and then used to estimate the expected actual coding time between successive commits.[[29]](#footnote-29)

There are also a number of more basic algorithmic approaches to analyzing data about software engineers. One of the simplest algorithmic approaches is a linear regression. This shows the relationship between one single input variable and one single output variable. In terms of software engineering data, a linear regression algorithm could help you understand how the number of commits and engineer made varied over time. It could also how the number of lines of code changed over the lifespan of a project. This, however, may not be a great predictor of engineering productivity as it only has the capacity to examine data sets with a single independent and dependent variable, and as previously discussed, software engineering productivity is a measure that depends on a wide array of variables.

Increasing the complexity of the linear regression, a multiple linear regression is another option for analyzing data. A multiple linear regression allows multiple independent variables. These variables are each assigned a weight that corresponds to their coefficient. This gives us a much broader picture of the software engineering process than a simple linear regression. With a multiple linear regression, we can look at how multiple variables, such as lines of code, code churn, and number of bugs closed change over time. Each variable that we want to look at can be assigned a weight based on how much in contributes to productivity to give us a better picture of the overall productivity of an engineer.

Another simple algorithmic approach to analyzing software engineering data is by using a simple ranking algorithm. This can be done by first choosing the factors that should affect the ranking. Then the factors can be categorized based on whether they increase or decrease the overall ranking. The factors can also be assigned weights based on their importance. This type of ranking algorithm could be used to rank the skills of engineers and compare them to each other to see who the stronger and weaker performers are.

While algorithms are useful tools for helping us to analyze data, there is a danger in relying too much on algorithms to inform our decisions. Algorithms are written to maximize efficiency and “humans are considered to be an ‘input’ to the process rather than as real, thinking, feeling, changing beings. Also, because algorithms are written by people, they have the potential to be bias towards people who are similar to the writers of the algorithm.[[30]](#footnote-30)

While some people believe that machines will never be able to replace jobs that require creative thinking or certain complex problem solving, Daniel and Richard Susskind, researchers on the future of professions, point out that machines do have the capability to perform non-routine tasks, even though they do it in a different way.[[31]](#footnote-31) This could eventually lead to even more complex jobs being taken over by algorithms and machines. It could also lead to more and more decisions being made by machines based on the data they have aggregated. While many people would trust machines to make decisions some decisions for them, such as when to stop a car before oncoming traffic, machines could start to be used to make other decisions that would be more problematic and potentially harmful, such as when to take someone off life support.[[32]](#footnote-32)

Thus, when we are using algorithmic approaches to inform decisions, we must also remember to take the human aspect of the decision-making process into account.

**Ethics**

In an increasingly digital world, a focus on improving efficiency across all business processes has become the paramount goal of many organizations. By relying on data to inform decision-making many businesses have found ways to save time and money. However, there are numerous ethical considerations that we all must take into account when thinking about employee and workplace monitoring.

When collecting data about a group of employees, one of the first ethical issues we should think about is how that data will actually be used. While data can help inform decisions, it does not tell the whole story and thus, should not be the only thing that is considered in terms of understanding an employee. For example, it could be potentially problematic if the data alone was used to decide which employees should be terminated from a company. Thus, we need to tread carefully on how much of the business decision-making process is staked on data.

We also have to be careful in considering how employee data is used because misuse of data could potentially harm employees. If employees learn that their company is tracking and rewarding certain behaviors, they may try to adapt their behaviors to fit these metrics, even if doing so has negative consequences for the developer. For example, if a company is tracking the number of hours their software engineers spend in front of a computer screen coding, some engineers may feel the need to sit at their desk more or even do work when they should be out of the office relaxing.

The instance of Amazon’s tracking of its warehouse workers exemplifies the negative repercussions that can result when businesses do not think about the ethics of how they are using the data they gather on employees. In 2018, Amazon obtained a patent for a wristband for warehouse workers to wear that would track their movements to measure their productivity. The technology would also emit sounds to guide workers to where they needed to be.[[33]](#footnote-33) While these wristbands have not been seen in use in Amazon warehouses yet, workers report that Amazon has a very strict productivity policy and even uses technology that can automatically generate warnings for workers who are not hitting productivity quotas.[[34]](#footnote-34) This can all be done without any input from a supervisor. It has also been reported that lack of productivity is a major reason for termination at Amazon with around 300 workers being fired for this reason at one warehouse in a single year.[[35]](#footnote-35) Monitoring practices such as these have raised questions about the ethics of employee monitoring as well as privacy issues. While productivity is necessary for meeting business goals, when companies put data and productivity at the forefront of their mission, they may be overlooking the health, safety, and needs of their employees.

Another area of ethical concern regarding measuring the productivity of software engineers, and workers in general, is how doing so will affect the future of work and the role of machines in decision making.  In their book *The Future of the Professions: How Technology will Transform the Work of Human Experts*, Daniel and Richard Susskind discuss the decomposition of professional jobs. This is the idea that in today’s data-driven productivity-focused world, professional jobs are being broken down into smaller parts that are then crafted for maximum efficiency.[[36]](#footnote-36) The book also argues that in the future, the parts of professional that can be replaced by algorithms will be as the algorithms become more efficient than their human counterparts. This will lead to a reduction in the number of low-skilled jobs available in the workplace.

Monitoring employees’ productivity and trying to streamline all business processes could lead to a dehumanization of the workplace. Algorithms are primarily designed to optimize productivity without regard to the social implications surrounding the data that is being analyzed. Eventually, we may live in a world where the algorithms that analyze data and inform our decisions are themselves written by algorithms.[[37]](#footnote-37) When this is the case, humanity is left entirely out of the loop.

The issue of privacy is another ethical concern that arises when we talk about measuring engineer productivity. Today, data-driven decisions drive increasing productivity across industries. However, as companies try use data to increase the productivity of more and more aspects of their businesses, they find that they must keep collecting more and more data. This is potentially threatening for workers’ privacy. For example, it could lead to businesses tracking all their employees’ activities, not only at work but also in their free time. This could create a system where workers are forced to share all of their private information with their employer just for the sake of increasing productivity. Clearly, we must have to draw a line somewhere about what information should be allowed to be kept private and off limits to your employer.

For me personally, I initially had a pretty neutral view of workplace tracking. I tend to stay focused on my assigned tasks at work pretty well, so the idea of being tracked didn’t really bother me. However, through my research, I have found that while collecting data on employees can be helpful for increasing productivity, it can often be misused in ways that hurt employees. Also, as a software engineer, I know that some simple metric like lines of code does not fully encompass the work involved and that sometimes the effort put into engineering is hard to see when just looking at the final result.

**Conclusion:**

Overall, we can see that although software engineering is a highly complex process, there are ways to get a measurement of software engineering productivity. Software engineering has many measurable aspects and a number of both computational tools and algorithmic approaches available for analyzing the data we measure. When looked at individually, no one metric provides us with an accurate view of engineering productivity. However, when we analyze combinations of multiple different measurements, we start to get a clearer picture of a software engineering team’s health and strength.

Data alone, however, should not be used to make decisions, as there is always a human component involved in the software engineering process that should be considered. There are a number of subjective measurements that characterize a good engineer, such as how dedicated an engineer is to the job or how willing he or she is to learn and grow. While productivity measurements do not appeal to many engineers, it does not seem as though they will be going away anytime soon. In order to alleviate the stress associated with measuring productivity, businesses should always be transparent about what they are tracking and about how the information they gather is being used. Ultimately, what both businesses and engineers have to understand is that no single tool or piece of data tells the whole story of an engineer’s productivity.

1. <https://www.devteam.space/blog/how-to-measure-developer-productivity/> [↑](#footnote-ref-1)
2. <https://nortal.com/blog/the-myth-of-developer-productivity/> [↑](#footnote-ref-2)
3. <https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo> [↑](#footnote-ref-3)
4. <https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451> [↑](#footnote-ref-4)
5. <http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/slides16.pdf> [↑](#footnote-ref-5)
6. <http://scet.berkeley.edu/wp-content/uploads/Report-Measuring-SW-team-productivity.pdf> [↑](#footnote-ref-6)
7. <https://www.devteam.space/blog/how-to-measure-developer-productivity/> [↑](#footnote-ref-7)
8. <https://blog.usenotion.com/13-essential-software-development-metrics-to-ensure-quality-219cfc264ed1> [↑](#footnote-ref-8)
9. <https://blog.usenotion.com/13-essential-software-development-metrics-to-ensure-quality-219cfc264ed1> [↑](#footnote-ref-9)
10. <https://stackify.com/measuring-software-development-productivity/> [↑](#footnote-ref-10)
11. <https://www.estimancy.com/en/2018/09/25/story-points-vs-function-points/> [↑](#footnote-ref-11)
12. <https://stackify.com/measuring-software-development-productivity/> [↑](#footnote-ref-12)
13. <https://blog.gitprime.com/3-misconceptions-code-churn/> [↑](#footnote-ref-13)
14. <https://stackify.com/measuring-software-development-productivity/> [↑](#footnote-ref-14)
15. <https://www.gitclear.com/> [↑](#footnote-ref-15)
16. <https://www.gitclear.com/> [↑](#footnote-ref-16)
17. <https://www.gitclear.com/line_impact_glossary> [↑](#footnote-ref-17)
18. <https://www.gitprime.com/> [↑](#footnote-ref-18)
19. <https://www.gitprime.com/> [↑](#footnote-ref-19)
20. <https://codeclimate.com/> [↑](#footnote-ref-20)
21. <https://www.castsoftware.com/products/application-intelligence-platform> [↑](#footnote-ref-21)
22. <https://www.castsoftware.com/products/application-intelligence-platform> [↑](#footnote-ref-22)
23. <https://www.infoworld.com/article/2621328/how-ibm-started-grading-its-developers--productivity.html> [↑](#footnote-ref-23)
24. <http://sunset.usc.edu/research/CODECOUNT/> [↑](#footnote-ref-24)
25. <http://sarnold.github.io/cccc/CCCC_User_Guide.html> [↑](#footnote-ref-25)
26. <http://www.campwoodsw.com/sourcemonitor.html> [↑](#footnote-ref-26)
27. <https://www.ndepend.com/> [↑](#footnote-ref-27)
28. <https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f> [↑](#footnote-ref-28)
29. <https://semmle.com/assets/papers/measuring-software-development.pdf> [↑](#footnote-ref-29)
30. <https://www.pewresearch.org/internet/2017/02/08/code-dependent-pros-and-cons-of-the-algorithm-age/> [↑](#footnote-ref-30)
31. <https://www.youtube.com/watch?v=Dp5_1QPLps0> [↑](#footnote-ref-31)
32. <https://www.youtube.com/watch?v=Dp5_1QPLps0> [↑](#footnote-ref-32)
33. <https://www.mhlnews.com/warehousing/will-amazon-s-worker-tracking-wristbands-cross-privacy-line> [↑](#footnote-ref-33)
34. <https://www.technologyreview.com/f/613434/amazons-system-for-tracking-its-warehouse-workers-can-automatically-fire-them/> [↑](#footnote-ref-34)
35. <https://www.technologyreview.com/f/613434/amazons-system-for-tracking-its-warehouse-workers-can-automatically-fire-them/> [↑](#footnote-ref-35)
36. <https://www.youtube.com/watch?v=Dp5_1QPLps0> [↑](#footnote-ref-36)
37. <https://www.pewresearch.org/internet/2017/02/08/code-dependent-pros-and-cons-of-the-algorithm-age/> [↑](#footnote-ref-37)