Measuring Software Engineering

Cian Duffy

18322506

Software Engineering

Professor Barret

**Introduction**

This paper aims at delivering a report considering the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethical concerns surrounding this kind of analytics. A series of published research papers will be referenced to explore these subjects and to provide real world examples of the various metrics used to achieve this. In order to explore the process of a Software Engineer we can use the measurable data of the extensively debated field of productivity measurement which will be the main focus of this paper.

**Measurement**

In order to understand what is meant by the measurement of productivity we must concern ourselves with the definition of productivity. Is it a quantitative measurement of the work done by a person during a certain period or rather a qualitative measurement such as effectiveness or efficiency of the same work? In Software Engineering, the term productivity has been frequently defined as the effectiveness of productive effort, measured in terms of the rate of output per unit of input.[[1]](#footnote-1) This comes from an economic point of view which is appropriate considering the aim of most Software Engineers is to further the development of some form of product that they, or the company they work for, wish to sell. This specific term suggests an association or direct correlation between the rate of output versus the unit of input, and it is this relationship that we will concern ourselves with.

So how do we measure this unit of input within the scope of Software Engineering and how will it affect the rate of output? One of the first approaches at this was the famous Lines of Code (LOC) method, which simply uses the number of lines of code produced by a Software Engineer as a means of measuring his or her productivity. [[2]](#footnote-2) Due to some inherent shortcomings, it became clear that this method did not yield satisfying results due to problems such as redundancy, where lines of code are dispensable as they are duplicates or simply not used during execution. This shows how problematic a quantitative approach of productivity measurement can be. The next logical step is to improve this method by introducing a qualitative approach using some extra key requirements.

A particularly interesting approach to measuring productivity using this LOC method was made where several performance indicators were used to improve its reliability and accuracy. These include identifying and ignoring redundant parts of a system in the source lines of code, identifying defects such as bugs or failures to implement required functionality and finally the total days taken to develop from setup to delivery. By using the methods outlined above many of the shortcomings of the LOC method can be minimalised and the results can even help to reduce code verbosity and improve code functionality/ accountability. [[3]](#footnote-3)

So far this idea of productivity measurement has been applicable to both groups of software engineers and the individual programmer by the notion of using a qualitative and quantitative approach. We have concerned ourselves mainly with the physical work of the programmer(s). What if we were to direct our attention to some external factors affecting the working quality of the programmer such as the environment in which they work or the lifestyle that they have? After all we are attempting to quantify productivity through some metrics of input vs output and if these are to be affected by some other factors, then by exploring these we can further refine our approach.

One such factor that I found intriguing is the theory of the relationship between workplace productivity and happiness, and how this can be measured to reflect overall productivity. By using data spanning over 9 years which recorded peoples activity using wearable technology, the company “Hitachi” was able to identify characteristics of human behaviour which has a strong correlation with happiness. Coupling this method of measuring “happiness” (which they called the 1/T fluctuation) with a workplace study, a recent research project focused itself on proving this theory. The study showed that workers who led a more active and “happy” lifestyle displayed a productivity rate of almost 34% higher than those who did not. Other such results implied that there are measurable factors that can influence this level of happiness such as setting “challenging yet achievable” tasks and creating an open and collaborative working environment. “Happiness corresponds to the roof covering the valleys of boredom and anxiety and is a state of high productivity. The measurement of 1/T fluctuation is a valuable tool for staying on top of this roof.” Here the 1/T fluctuation refers to their definition of the proposed measurable happiness. This experiment shows the direct correlation between the external working factors, such as environment and lifestyle, and the productivity it influences.[[4]](#footnote-4) The possible ethical implications of this will be discussed later. We have seen that these factors are measurable using this technology in the common workplace therefore could be integrated into the measurement of productivity of a Software Engineer.

By exploring the quantification of productivity through various forms, we have shown how the process of a Software Engineer can be measured and assessed. This brings us to our next topic of discussion which is a deeper look into the platforms that are available to gather and process this data.

**Platforms**

To further develop our understanding of the software engineering process through the scope of productivity, we can look to the frameworks and platforms that are readily available and being used today to not only collect but analyse software engineering metrics. We will look at some established companies like Pluralsight Flow and Code Climate that provide such services.

One of the industry’s leading companies that provides a platform for ensuring efficiency and productivity within software engineering teams is Pluralsight Flow which is powered by Git Prime. The main aim of this framework is to monitor a team’s version control system to provide real time feedback and insight into the productivity and practice using various metrics. These include viewable data parameters like commits per coding day, churn, help other and impact.[[5]](#footnote-5)

One of the most interesting of these is churn as it tracks individual commits made to a project and monitors them to determine if they’ve been modified by the same author over the following couple of weeks. If a deadline approaches and the code churn increases, it can often be a sign that the code is getting more volatile which can be easily be caught by checking this metric on the system.

The help others metric is also fascinating as it helps teams to ensure that the senior developers are the ones leading the software development. The system measures the frequency at which code written by one author is then modified by another, so you would of course wish to see a high rating in this metric for the senior developers. It is worth noting that this system differs from others due its aim being centred at improving collaborative working dynamics through these metrics, rather than improving the quality of the code being written by the team. However, by processing this data through the metrics mentioned, it can highlight negative trends in how the team works which will in turn improve the overall quality of code. Essentially, Pluralsight Flow is a computational platform which is responsible for compiling and processing data about the software engineering process.

We have already mentioned that the quality of code an author can write is an essential part of the software engineering process. The strength of the aforementioned platform is its ability to collect data about how teams work together and aims at improving that collaboration. But what if we wanted a more reliable source of feedback for code quality? This brings us to the next platform, Code Climate. Code Climate is an automated static quality analysis tool that aims at providing real time feedback on the quality of an author’s code. To accurately examine and improve the standard of code being produced by individuals and teams, it offers a number of features such as test coverage, integration and tracking. [[6]](#footnote-6)

The test coverage metric measures and displays the coverage and range of code which shows the abilities of the code and the processes that it entails. This can report on individual files for important micromanagement or on the overall system to independently check the validity of a software system.

The assessment platform integrates directly with GitHub to retrieve all this data in order to process it and return it to the user. Using the browser extension, line by line feedback can even be given to ensure code is not only of appropriate quality but can reduce technical debt. This is a notorious obstacle in the software development process which entails the financial cost of code reworks caused by using an often faster yet unreliable approach.

Finally, the tracking metric offers a retrospective view on the code commits to show the most important changes and merges by visualizing the trends in the codebase. This measurement allows authors to learn from previous efforts and possible mistakes to improve their skills. It is through these three methods that Code Climate provides a platform that can gather and process data in the form of files of code to provide an overall assessment on this essential part of the software engineering process.

These two computational platforms offer interesting feedback on two of the main facets of software engineering by gathering data on teamwork dynamics and code quality respectively. These are arguably the two most important parts of the software engineering process and through these platforms we can start to see the ways in which data regarding these subjects can be assessed in terms of measurable data which is ultimately focused on improving productivity and efficiency.

**Algorithms**

Thus far we have explored some of the key concepts involved in quantifying and measuring the software engineering process and we have shown some examples of how that can be done using platforms available today. Now we can go even further to discuss the technical approach of this by examining some of the assessment models that use various techniques and algorithms in order to accurately determine the measurable data of performance levels.

One of the more interesting approaches includes the use of machine learning, of which I have personal experience in due to my work in the software engineering project of last year and will be the main subject of exploration. The project entailed using machine learning techniques for time series forecasting whereas here, machine learning techniques are used to assess the productivity of the industrial process of a software engineer. I have already mentioned that you cannot simply assess productivity either through quantity of code *or* quality of the code, but an accurate and structured measurement of the two combined would suffice to bring forth an accurate measurement. One of the major strengths of machine learning is its ability to compute outputs from large amounts of inputs or datasets which brings us to where such datasets can be found.

As this approach requires mass amounts of this type of data, we can look to LGTM which as a service, provides millions of source control commits of code and treats the code as data. This covers the quantity of data necessary for the algorithm however we still do not have appropriate code quality data. Conveniently LGTM analyses each commit identifying things such code mistakes which supplies measurements on code quality through a classification of error alerts.

Now that the conditions are satisfied for the input of a machine learning algorithm, we can talk about the first algorithm proposed by a team of software engineers in 2018 to draw some productivity metrics from this empirical data.[[7]](#footnote-7) Firstly, the team attempted to train a neural hidden Markov model which is used to capture long term dependencies. The main theory behind using this involves attempting to identify coding time information from intervals between code commits. The algorithm succeeds in the aim to “predict the probability an individual developer is currently coding for each 1 minute of their recent history”. (Measuring software development productivity: A machine learning approach, 2018, p.4) This outputs the first metric which is an accurate measurement of active coding time which will be used in the next algorithms to produce some more standards of productivity measurement.

The next algorithmic approach involves using what’s called a deep mixture density network (DMDN). An inherent feature of machine learning techniques is their complicated mechanics and associated terms. It is therefore easier to provide an overview as to how the algorithm works using its inputs and outputs. It uses the aforementioned coding time models to create multiple datasets that compares coding times to code changes in order to form a metric that represents an average software developer. As you can imagine some of the difficulties of doing this mainly include the extremely varied levels of code quality within the datasets of code. This introduces a factor of random noise which in machine learning is referred to as irrelevant information or randomness in datasets.[[8]](#footnote-8) This is handled by the deep MDN which essentially refers to a multi layered network which can accept a nonlinear function of input parameters. The result is an algorithm that when given a dataset of code changes, can predict the average coding time required to produce it. This gives us a working metric which can be used to describe the average process of a software engineer and can also be used as a tool to gauge how productive each individual can be by assessing their code, measuring their productivity using this algorithm and referring it to this average.

The final machine learning model involves the use of a nonlinear quantile regression model which is a form of regression analysis in which data is fit to a model and then expressed as a mathematical function[[9]](#footnote-9) This is then used with the data concerning the ranked code alert information to produce a quality grade. The output is a weighted result determined by a function which outputs how well a project performs.

These machine learning techniques use gathered data from the software engineering process to output what is a standardised measurement of code output and quality. Not only does this help us to measure productivity, but it allows us to see different ways of measuring the software engineering process as a whole.

**Ethics**

The previous sections of this paper have purposely provided a purely scientific approach to the ways in which a software engineer *can* be measured as opposed to including whether a software engineer *should* be measured. As we have seen, through some of the ways in which measurements have been devised and implemented, almost every method has included the retrieval of people’s data and in some cases the recording of social interactions. This brings us to the ethical implications of large-scale analytics particularly in relation to the software engineering industry through the measurement and assessment of the software engineering process.

Earlier when exploring how one might go about measuring productivity, the company of Hitachi developed a wearable piece of technology which was aimed at quantifying happiness by using metrics such as characteristic patterns of physical activity that have a strong correlation with happiness. In additions to this, the study included that the company had “collected and studied more than a million days’ worth of data on people’s activities using wearable technology”. (Measuring Happiness Using Wearable Technology, 2015, p.98) Where the study had taken place it is even noted that it “was possible to enhance overall happiness and productivity by controlling communication between staff”. (Measuring Happiness Using Wearable Technology, 2015, p.103) Conversations during breaks of work were treated as variables that could be controlled and even enhanced to improve their key performance indicator.

The main questions surrounding this type of research is should this be taking place? Does it cross some ethical boundaries by invading personal privacy. grading social interaction during working hours and even physical activity during off work hours? Even the active research into monitoring the interactions between workers to grade and improve productivity almost eludes to themes explored in Aldous Huxley’s “Brave New World”. It centres around a utopian society where a drug called “soma” is used to alter a person’s mood by repressing certain emotions to boost productivity and focus. The point of this is merely to show that this type of data analysis aimed at measuring productivity is entering into unprecedented areas of research which is more concerned with the personal activities of a person rather than the professional work they exhibit.

One of the most interesting things I discovered when researching for this paper is in relation to the companies and platforms in which the research papers are published. Much of the reference material cited comes from one of the leading companies for computing and engineering IEEE. As of 2014, the ethical guidelines had not been updated in almost two decades.[[10]](#footnote-10) This is particularly worrying when considering the amount of technological development like those discussed earlier and the new ways in which large scale data retrieval is being used for research. Given that the data retrieval we saw in the 2003 paper for machine learning algorithms, it is very possible that the ethical guidelines they followed could have been outdated. However, this is even hard to conclude as ethical guidelines were not even discussed in the paper.

It is extremely important that the research involving big data must adhere to relevant and contemporary ethical guidelines to ensure that a person’s private information is not taken without their consent. This is of course more than relevant in the case of measuring the work of a software engineer through the methods that were outlined above.

**Conclusion**

To conclude, this paper delivered a report considering the ways in which the software engineering process can be measured and assessed in terms of measurable data through the discipline of productivity evaluation. Various methods aimed at achieving this were explored including the platforms used to gather and process this type of data, the algorithms and assessment models involved and finally the ethical problems and concerns surrounding these kinds of analytics. This field of study is widely debated in the world of computer science and with the recent surge of big data orientated research, it will certainly continue to grow and introduce even more interesting ways in measuring the software engineering process.

**Bibliography**

B. W. Boehm. Software Engineering Economics. *Advances in Computing Science & Technology*. Prentice-Hall, Englewood Cliffs, NJ, USA, Dec. 1981.

Crawford, K., Gray, M. L., & Miltner, K. (2014). Big Data| critiquing Big Data: Politics, ethics, epistemology| special section introduction. *International Journal of Communication*, *8*, 10.

Everything You Need to Ship Better Code, Faster | Quality. (2020). Retrieved from Codeclimate.com Website: <https://codeclimate.com/quality/>

Hélie, J., Wright, I., & Ziegler, A. (2018). Measuring software development productivity: A machine learning approach. In *Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC* (Vol. 18).

Kalapanidas, E., Avouris, N., Craciun, M., & Neagu, D. (2003, November). Machine learning algorithms: a study on noise sensitivity. In *Proc. 1st Balcan Conference in Informatics* (pp. 356-365).

Oliveira, E., Viana, D., Cristo, M., & Conte, T. (2017, April). How have Software Engineering Researchers been Measuring Software Productivity? - A Systematic Mapping Study. In *International Conference on Enterprise Information Systems* (Vol. 2, pp. 76-87). SCITEPRESS.

Parareda, B., & Pizka, M. (2007, December). Measuring productivity using the infamous lines of code metric. In *Proceedings of SPACE 2007 Workshop, Nagoya, Japan*.

Pluralsight. (2019). How to lead world-class engineering teams with Pluralsight Flow [Video]. Retrieved from<https://www.youtube.com/watch?v=OkPF27xW5kk>

What Is Nonlinear Regression? (2020). Retrieved November 26, 2020, from Investopedia website: <https://www.investopedia.com/terms/n/nonlinear-regression.asp>

Yano, K., Akitomi, T., Ara, K., Watanabe, J., Tsuji, S., Sato, N., ... & Moriwaki, N. (2015). Measuring happiness using wearable technology. *Hitachi Review*, *64*(8), 517.

1. Oliveira, E., Viana, D., Cristo, M., & Conte, T. (2017, April). How have Software Engineering Researchers been Measuring Software Productivity? - A Systematic Mapping Study. In *International Conference on Enterprise Information Systems* (Vol. 2, pp. 76-87). SCITEPRESS. [↑](#footnote-ref-1)
2. B. W. Boehm. Software Engineering Economics. *Advances in Computing Science & Technology*. Prentice-Hall, Englewood Cliffs, NJ, USA, Dec. 1981. [↑](#footnote-ref-2)
3. Parareda, B., & Pizka, M. (2007, December). Measuring productivity using the infamous lines of code metric. In *Proceedings of SPACE 2007 Workshop, Nagoya, Japan*. [↑](#footnote-ref-3)
4. Yano, K., Akitomi, T., Ara, K., Watanabe, J., Tsuji, S., Sato, N., ... & Moriwaki, N. (2015). Measuring happiness using wearable technology. *Hitachi Review*, *64*(8), 517. [↑](#footnote-ref-4)
5. Pluralsight. (2019). How to lead world-class engineering teams with Pluralsight Flow [YouTube Video]. Retrieved from https://www.youtube.com/watch?v=OkPF27xW5kk [↑](#footnote-ref-5)
6. Everything You Need to Ship Better Code, Faster | Quality. (2020). Retrieved from Codeclimate.com

   Website: https://codeclimate.com/quality/ [↑](#footnote-ref-6)
7. Hélie, J., Wright, I., & Ziegler, A. (2018). Measuring software development productivity: A machine learning approach. In *Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC* (Vol. 18). [↑](#footnote-ref-7)
8. Kalapanidas, E., Avouris, N., Craciun, M., & Neagu, D. (2003, November). Machine learning algorithms: a study on noise sensitivity. In *Proc. 1st Balcan Conference in Informatics* (pp. 356-365). [↑](#footnote-ref-8)
9. What Is Nonlinear Regression? (2020). Retrieved November 26, 2020, from Investopedia website:

   https://www.investopedia.com/terms/n/nonlinear-regression.asp [↑](#footnote-ref-9)
10. Crawford, K., Gray, M. L., & Miltner, K. (2014). Big Data| critiquing Big Data: Politics, ethics, epistemology| special section introduction. *International Journal of Communication*, *8*, 10. [↑](#footnote-ref-10)