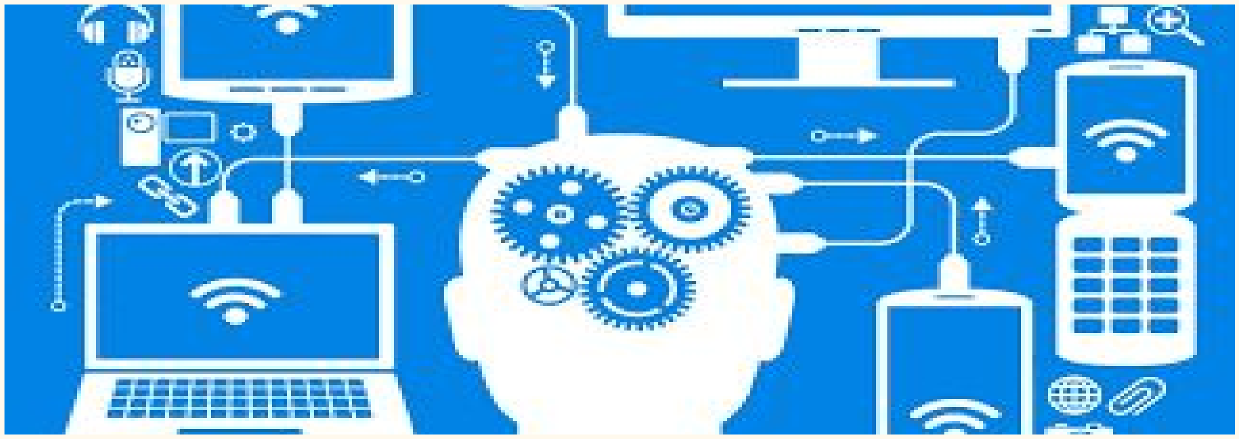
Report

Measuring Engineering

By Dylan Lewis Student No. 1531759



INTRODUCTION

For as long as software has been a commercial product companies have created methods of measuring whether a software engineer is actually worth their value. The goal of this report is to examine the various ways in which software is measured and an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding these analytics.

Lines of Code Measurement

One form of measurement is lines of code aka LOC. LOC measures the amount of lines of code of a project. This measurement was first used when the most common programming languages were FORTRAN and assembly languages both of which are line oriented languages. This form of measurement can be used to effectively measure the effort put into a particular project however in is a poor indicator of functionality as highly skilled developers may be able to produce the same feature with fewer lines of code.[1]Therefore a program with fewer lines of code than another doesn't indicate whether one contains more or less functionality than another.

Unfortunately this measurement method is not good at determining the productivity of any individual as a developer may be more productivity writing a few lines than another developer writing many more lines of code. This also encourages code duplication within a program because if repeated lines of code are placed within a function then the LOC is reduced. This is considered a negative however the practice of eliminating duplicate lines within code is considered good practice among developers as it increases readability and reduces complexity of the code. LOC's code duplication encouragement can also causes issues as code duplication is known to be bug prone and costly to maintain.

LOC is also nearly useless at comparing programs in different languages unless adjustments to the measurement system are made. For example assembly code will take many more lines to do any complex operation than a higher level language like python or java. Another major problem with this measurement system is the difference between machine generated code and hand written code. Most development environments used by software engineers have ways of generate large amounts of code meant to perform a specific task. These lines of code cannot be compared to other programs as they weren't written by the engineer and didn't require any effort on behalf of the developer.

Another major issue with the LOC method is how easy it is to manipulate the system, in order for a developer to manipulate this system they simply have to artificially extend code for any program, making a program that could be written in a line to being written to several lines, for example hello world program in c, could be written like so to inflate the number of lines of code in the program:

```
int main()
{
printf(
“Hello World”
);
return  0;
}
```

Overall LOC has numerous disadvantages as described above and doesn't accurately measure a software engineer's success on its own which is why it is not used by companies for measuring software engineers. However during the early years of software engineering this method was commonly used which resulted in unnecessarily long source code for programs and duplicate code within

Technical Debt Measurement

Technical debt is considered code that can be easy to implement to create and run in the short term however is not the best solution to a problem overall. Technical debt can incur costs due to the amount of maintenance necessary to maintain the working code It is also costly when trying to develop new features for the program or when trying to develop a new project using the existing code. When technical debt builds up over multiple projects it can cause serious delays to a project unfortunately with technical debt it is hard to calculate exactly how much work it takes in order to pay off the "debt".[2]

This could be an effective measurement as it could determine whether an engineer is actually contributing more to the company than they are costing. It would also determine the skill of the engineer as a poor engineer would produce a lot of technical debt while a good engineer would ideally produce little to no technical debt. However this may not be true as it could be that the technical debt of a project was due to the limited time given for the project and not due to any engineer's lack of ability. Also as stated above it difficult to accurately calculate the amount of technical debt accrued from any one project which means that is hard to calculate whether any engineer is costing the company more money than they are worth. Overall this method is not that effective as technical debt is a vague concept that can be hard to quantify in exact terms which defeats the purpose of a measurement tool to determine whether a software engineer is worth it.

Ways of Measuring

Like with any metrics the question of how we collect them must be determined. Overall the easier a metric is to collect the more useless it is, however the original version of the Personal Software Process (PSP) can yield rich, high-impact analytics. However, it incurs significant overhead cost for developers. The Collaborative Software Development Laboratory at the university of Hawaii at Manoa began to use and evaluate the PSP as way of measuring software engineers ability and worth. The PSP was described in the book A Discipline for Software Engineering by Watts Humphrey's[3]. This version of PSP involves the use of manual data collection, and manual analysis however unfortunately the use of PSP as described above carries substantial cost as it takes a lot of effort to obtain all the necessary data. One of implementation of PSP required an engineer to fill out twelve forms which would require the developer to submit a project plan summary, a time-recording log etc.

These forms will produce up to 500 distinct values that would have to manually calculated by developers this is would take effort and is part of the cost of this analytic. PSP was viewed as bootstrapping method by Humphry's [3] who believed that the values had to be manually calculated rather than calculated using an automated method. However this manual nature is both a strength and a weakness for PSP as it makes it analytics fragile and open to the possibility of errors but PSP analytics are flexible which is a benefit.

Unfortunately PSP's reliance on the developer to manually calculate the values and determine a conclusion sometimes led to an incorrect conclusion to be made in spite of an overall low error rate[4]. However to combat this error rate the Leap (lightweight, empirical, antimeasurement dysfunction, and portable software process measurement) toolkit was developed by Collaborative Software Development Laboratory (CSDL) at the University of Hawaii at Manoa[5]. The leap toolkit was an attempt to address the error issues by automating the PSP process however the data is still manually entered by the appropriate developer[6]. This attempts to avoid errors by allowing developers to control their data files.

This anonymises data by not keeping track of the individual developers personal information simply the values given to it. It also allows for the data on individual developers to brought over from one project to another this allows for a developer's metrics to be tracked consistently over time. Unfortunately the automation which makes leap easier to use also makes collecting some metrics more difficult and other metrics easier as when a developer would need to measure whether something was affecting productivity than this developer would need to create a whole new leap instance and fill in the fields however this is more work than the original PSP method as that was simply creating a new spreadsheet[5].

Computational Intelligence

When people talk about computational intelligence they are typically talking about machines being able to learn to complete a task based on data or from experimentation. Computational Intelligence's goal is to convert real life scenarios into various metrics which a computer can analyse and allows the computer to perform an action. Computational intelligence is considered a form of soft computing which is when machines are able to use partially inaccurate solutions in order to solve problems for the most part these are used on NP-complete problems.[7] Computational intelligence also makes use of fuzzy logic to understand natural language i.e English, German, French, Chinese etc.

It should be noted that Computational intelligence is different to the idea of Artificial intelligence as while Artificial intelligence seeks to create a machine which can think and act like a living creature and be able to understand and adapt to its environment. Most companies will have moved away from the old model of trying to create a cognitive machine and more towards the statistical models employed by machine learning and computational intelligence.

Computational intelligence's main uses are in the fields of data analysis and bio-medicine.[9] Since one of the main ways in which a software engineer is measured is using vast amounts of data about their projects and code it should be possible to apply computational intelligence to measuring a software engineer's worth. This would require training the machine to recognise what the company considers to be good software engineers and then passing all relevant data into the program. The program should then be able to create some kind of statistical model on what is a good software engineer according to the company allowing us to completely automate the process and simply provide it with relevant data.

However there is a catch to this kind of solution in that as mentioned above Computational Intelligence uses inexact solutions to make a choice which means that ultimately it's a game of statistics as to whether or not someone is fired. This means that over a large enough sample size a significant amount of good competent engineers could be let go despite doing a good job. This could lead to a company losing talent and creating a climate of fear due to the prospect of being fired despite doing being by all accounts a good software engineer.

Machine Learning

Machine learning that gives computers the ability to learn without being explicitly programmed[10]. Tasks for machine learning are divided into two general categories, supervised and unsupervised learning. Supervised learning is when a machine must infer a function from a labeled training data[11] this data consists of a set of training examples. Each of these training examples consists of an input and a desired output. For supervised learning to work the programmer must decide what the training examples are going to be, create some kind of training set that will be used, decide on how to represent the data inputted into the program. Once this and a few other steps have been completed the programmer must test whether the program has produced the correct learned function this should be done using a test set which is unique to the training set.

This kind of approach could be used to evaluate software engineer as it would simply requiring determining what individuals are considered good software engineers based on select metrics then design a program which learns what makes a good software engineer based of the metrics inputted to it. However I am very skeptical of this approach because ultimately when automating the evaluation of software engineers the goal should be to remove any kind of human bias and look at them purely based on their work but supervised learning doesn’t actually remove any bias as it requires a human to distinguish[12] what makes a software engineer good or bad.

This means that any machine learning program using supervised learning will have the same biases as the individual who created the training sets which means that it won’t actually evaluate someone’s performance. Overall this could waste money and resources trying to fix biases within the system and actually reinforcing them. Supervised learning similar to computational intelligence is also a statistical model which means that it goes with the most statistically likely result based on the input[11] which ends up with the possibility of firing software engineers who are doing their job satisfactorily.

The other side of machine learning is unsupervised learning, this is when a function is inferred without the labeled training sets. In this type of machine learning the examples given to the program have no label there is no evaluation of the accuracy of the final function. This kind of system does remove bias like above but is a statistical model system and is not currently at the stage where it can be used for such a complex task which means that perhaps in the future unsupervised machine learning can be used to measure a software engineer but it will have similar problems to computational intelligence where engineers can be still be lost for no reason and there is no evaluation of the accuracy of the model.[15] Unsupervised learning also has a larger risk of the model being inaccurate due to the lack of an accuracy check involved[16].

Ethics

One the biggest issues with all the measuring of software engineers is the amount of data that any corporation should have access to. This is a fairly big issue in society at the moment as well. Companies can be unethical in their measurement of engineers with the sheer invasiveness of the data they want and what they do with that data. Some companies try and measure what mood their workers are most productive at then try to keep them at that mood constantly[13] I believe this to be unethical as the company is manipulating its employees unknowingly, allowing the company to ultimately treat the employees purely as assets rather than employees. In this example there was an information disparity, the company had access to their mood at all times and their productivity all without the employee realising it. This allowed for the company to exploit the employee without them knowing the scale of the data and what it’s being used for.

This leads to the question of what data can a company take and what is an invasion of your privacy. Does an company have the right to know your mood, your location, your favourite foods, how often you go to the toilet etc. It would be easy to say that a company can only collect data that an employee has consented to giving, but this is easy to get around as people may not be fully aware of the scale of the data given away. This kind of idea also doesn't work as it would mean that security cameras are a violation of people's right.

A major problem with companies having access to any kind of confidential data is the level of security that data is under we have seen time and time again that companies don't properly invest in cyber security such as equifax where 143 million American customers had sensitive data breached and exposed. Putting all these people at risk of identity theft.[14] If companies want access to this kind of data then they need to prove that they are responsible and are not at risk of losing it as if this data is exposed to the public then people are but at risk of identity theft or worse.

The more and more data these companies collect give them more power of any individual which makes people vulnerable to exploitation this allows companies to target people with behaviours they consider negative for productivity but is acceptable in the workplace and manipulate them into correcting that behaviour through manipulation. However companies may need to monitor your activity on your working environment(desktop) to ensure that you are not going to inappropriate websites and to get an estimate of how much work you do during your work hours. This kind of monitoring can be acceptable as it may be needed to measure the productivity of an individual and can be used to justify a termination of an employee if the employee is not performing well enough.

Once again though this must monitoring must be done carefully so as not to be abused by any individual in the company. This kind of monitoring can end up recording private data depending on the sites the employee visits, such as banking sites.

Another question surrounding this is whether a company has a right to use the data it collects as it creates an imbalance of power between the employee and the company. For instance if a company offered home insurance it might only offer you home insurance if you live in an area where home insurance is unnecessary.

This collection and use of data can also lead to a company reinforcing biases for instance if the employees who had been hired from south Dublin were less productive than the ones hired from north Dublin this might lead to a company deciding not to hire anymore people from south Dublin. However this doesn’t mean that south Dublin people are less productive than north Dublin or that north Dubliners are more productive because correlation doesn't imply causation.This means that while some values may correlate unless there is some proven cause and effect relationship between them the correlation is assumed to be purely coincidental.

Overall measuring software engineers through the use of mass data collection about the engineers is very complicated ethically as the question of how much data a company is allowed to collect on individuals without infringing on their rights. As the company still needs to be able to evaluate whether their employees are doing a good job and more often than not the more personal the data the more useful it is[6]. This is also coupled with the question of what companies are going to do with this data, as they have been shown to exploit and manipulate their employees in the search for greater productivity[13].

Unfortunately there is no one easy solution to this problem that will satisfy both the companies and the employees . If you value privacy the rights of the employees must come first if you value productivity above all else then the company should have the right to monitor whatever data they want. These issues can be solved by ignoring them at some point there must legislation clarifying to what extent companies can collect and retain data from people.

REFERENCES

1. Vu Nguyen; Sophia Deeds-Rubin; Thomas Tan; Barry Boehm (2007), *A SLOC Counting Standard* , Center for Systems and Software Engineering , University of Southern California

2. Allman, Eric (May 2012). "Managing Technical Debt". *Communications of the ACM*. **55** (5): 50–55. doi:10.1145/2160718.2160733

3. W.S. Humphrey, A Discipline for Software Engineering, Addison-Wesley, 1995.

4. P.M. Johnson and A.M. Disney, "The Personal Software Process: A Cautionary Case Study," IEEE Software, vol. 15, no. 6, 1998, pp. 85–88.

5. http://www.citeulike.org/group/3370/article/12458067

6. P.M. Johnson, "Leap: A 'Personal Information Environment' for Software Engineers," Proc. 21st Int'l Conf. Software Eng. (ICSE 99), IEEE CS, 1999, pp. 654–657.

7. Siddique, Nazmul; Adeli, Hojjat (2013). *Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing*. John Wiley & Sons. ISBN 978-1-118-53481-6.

8. Cornell University Library. "Breiman: Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". Retrieved 8 August 2015. Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani (2013). *An Introduction to Statistical Learning*. Springer. p. Vii.

9. R. Pfeifer. 2013. Chapter 5: FUZZY Logic. Lecture notes on "Real-world computing". Zurich. University of Zurich.

10. Supposedly paraphrased from: Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". *IBM Journal of Research and Development*. **3** (3). doi:10.1147/rd.33.0210 Confer Koza, John R.; Bennett, Forrest H.; Andre, David; Keane, Martin A. (1996). *Automated Design of Both the Topology and Sizing*

of Analog Electrical Circuits Using Genetic Programming. Artificial Intelligence in Design ’96. Springer, Dordrecht. pp. 151–170. doi:10.1007/978-94-009-0279-4_9. Paraphrasing Arthur Samuel (1959), the question is: How can computers learn to solve problems without being explicitly programmed?

11. Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) *Foundations of Machine Learning*, The MIT Press ISBN 9780262018258.

12. S. Geman, E. Bienenstock, and R. Doursat (1992). Neural networks and the bias/variance dilemma. Neural Computation 4, 1–58.

13. http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

14. https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-do
15. Jordan, Michael I.; Bishop, Christopher M. (2004). "Neural Networks". In Allen B. Tucker. *Computer Science Handbook, Second Edition (Section VII: Intelligent Systems)*. Boca Raton, FL: Chapman & Hall/CRC Press LLC. ISBN 1-58488-360-X.

16. Hastie, Trevor, Robert Tibshirani, Friedman, Jerome (2009). *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. New York: Springer. pp. 485–586.