8/11/2019

Measuring Software Engineering

Amy Pierce

CSB, 17330305

MEASURING SOFTWARE ENGINEERING

AMY PIERCE, CSB, 17330305

TABLE OF CONTENTS

MEASURING SOFTWARE ENGINEERING

AMY PIERCE, CSB, 17330305

1. INTRO

It is worthwhile to note that in recent times there has been a shift in measuring the work of engineers in groups to measuring engineers as individuals, for the purpose of this paper, this report will look at a mix of measures of both individual and team engineering processes and products. Throughout this report, we will not only look at the matrics that are used to measure software engineering, the computational platforms available to make this possible will also be discussed along side the different algorithmic approaches that make processing this data easier, machine learning is an up and coming algorithmic method that is rapidly becoming a popular way to deal with big data which will be discussed later in the paper. Finally, we will explore the ethical concerns that do arise when we look at the constant monitoring of software developers and their work.

1.1 WHY SHOULD WE MEASURE ENGINEERING?

The idea behind monitoring software engineering is essentially to get a better understanding of the quality of the process currently being used and looking into ways to improve it and attempt to estimate what kind of quality the finished project will be. By measuring engineering, managers can look at increasing return on investment, identify areas that need improvement, better manage workloads, reduce the amount of overtime required and ultimately reduce costs of the project.[1]

2. METRICS

2.1 DECIDING ON A METRIC

Many different metrics can be used when attempting to measure data. When deciding on what metrics to measure, it is important to consider if they have certain characteristics. Metrics should be:[1]

- Simple and computable
- Consistent and unambiguous (objective)
- Use consistent units of measurement
- Independent of programming languages
- Easy to calibrate and adaptable
- Easy and cost-effective to obtain
- Able to be validated for accuracy and reliability
- Relevant to the development of high-quality software products

2.2 METRICS IN USE

2.2.1 SOURCE LINES OF CODE (SLOC/LOC):

Source lines of code essentially measures the size of a program in relation to how many lines of code it is made up of and it was an early measure of productivity. It may be the simplest way to measure data but theta doesn’t mean it

[1] https://stackify.com/track-software-metrics/

is the best. There are two ways to measure LOC; count each physical line of code, but this may include dead codes or comments. The other way is to consider each logical statement a line of code.

Some may argue that it is a decent measure of how much effort was put into the program in thinking that more lines of code means require more time and so that the longer the code, the more time and therefore effort was put into it. However, it is important to note that longer does not necessarily mean better, sometimes it can simply mean over complexity of a program. A program with 100 lines of code could do the same thing program with 30 lines of code, just because the first program is longer does not mean it is the superior. More time may have been spent writing the longer program but at the end of the day both programs perform the same function and so using source lines of code may not be accurate as it cannot be said that a program is inferior simply because it is shorter, many may argue that the shorter version is the better version.

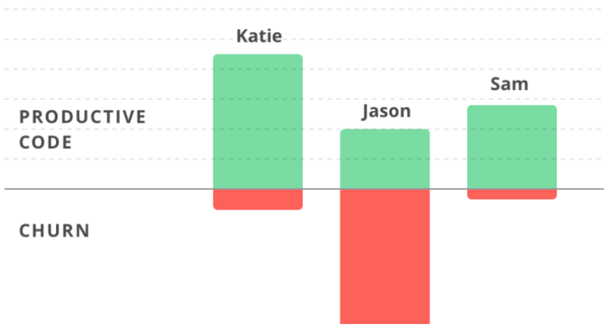
"My point today is that, if we wish to count lines of code, we should not regard them as "lines produced" but as "lines spent"- Dijkstra[2]

2.2.2 CODE COVERAGE:

This is a measure of how much of the code is being run when being tested. A program with a high code coverage means more of its lines of code were ran and tested and will have less of a chance of having undetected bugs or exception cases. This is a good metric to use in terms of monitoring how much testing is being done on the code in the development of the software.

2.2.3 CODE CHURN:

This is a measure of the adding, changing and deleting of code throughout a short time frame. It is measure in lines of code as a way of counting how many modifications are being made. It is useful especially when examining the quality and productivity of the programming in the development stages that is, if a developer is constantly modifying existing code rather than adding new code, this would be considered code churn. It is almost like when you are playing with the food on your plat rather than eating it, moving the food around your plate is unproductive. High levels of code churn can slow down the productivity and development of a project and can cause delays in production. However, it can be useful when looking to identify if and what a programmer is stuck on and then makes it easier to address the fact that a programmer is stuck and needs help on that area of the program that they are continuously going back to make changes to. This issue can then be addressed by providing assistance to the programmer and allocation more time to solving the problem which can then get the project back on track[3]



2.2.4 TEAM VELOCITY:

Team 'velocity' can also be referred to as productivity This is a measure of how long it takes a team to complete a task assigned to them. Before the development begins an estimation is made for how long it might take and then this issued as a comparison point for how long it

[2] https://www.cs.utexas.edu/users/EWD/transcriptions/EWD10xx/EWD1036.html

[3] https://blog.gitprime.com/why-code-churn-matters/

actually took the team. It can be a useful measure of productivity and can also be a good motivator for the team as it gives them a goal to strive for. However, the quality of the work also needs to be accounted for tin such scenarios to avoid the team speeding through the development to simply meet the estimation but so long as the estimation is realistic there should be no reason for the team to feel the need to rush and sacrifice the quality of the program.

2.2.5 QUALITY

There are many different metrics that can be used to measure software quality, for the purpose of this report, Cyclometric and Halstead seem most relevant for measuring software quality in relation to software engineering.

2.2.5.1 CYCLOMATIC COMPLEXITY

Cyclomatic Complexity is metric for measuring software quality. It is used for indicating a programs complexity by measuring the number of linearly-independent paths through a program's source code. When comparing two same size programs, it can be deduced that the more decision-making statements in a program, the more complex it is considered. As one would logically assume the lower the complexity of a program, the better the quality the code due to complex code being more difficult to test and is more likely to result in errors. [4]

2.2.5.2 HALSTEAD COMPLEXITY MEASUREMENT

Halstead complexity measurement is a measure of the complexity of a program. It is calculated directly from the programs source code, with emphasis on computational complexity. Halstead developed this method as a means of calculating a quantitative measure of complexity directly from the operators and operands in the program.

Halstead has a number of different metrics incorporated into it including: [5]

- Program length
- Vocabulary
- Program volume
- Potential minimum volume
- Program level
- Program difficulty
- Programming effort
- language level
- Intelligence content
- Programming time

While Halstead is a better measure for the complexity of the code that lines of code, it is not a perfect algorithmic approach to measuring software engineering due to the occasional issue of it having trouble distinguishing between operators and operands which can lead to errors

2.2.6 PSP- PERSONNEL SOFTWARE PROCESS

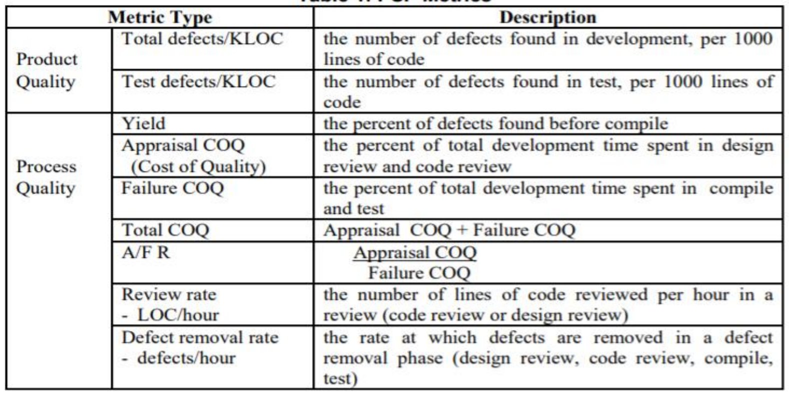
[4] https://www.perforce.com/blog/qac/what-cyclomatic-complexity

[5] https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

PSP is not a simple metric in itself, it is comprised of several metrics. By collecting and analysing data on the size, effort, and quality of the software they produce, it can help provide estimates about future work and allow developers look at ways they could improve their software engineering. The biggest drawback of PSP is that it requires all the data to be inputted manually by each individual developer into a spreadsheet like system. However, due to it being in a spreadsheet, it does allow a huge range of metrics to be collected and each individual company could customize and chose what metrics to be included and recorded in the spreadsheet, for example; size estimations, number of defects, hours spent on work, etc. Each developer would be required to make regular entries in the PSP regarding the metrics being recorded so as to make the tracking of the metrics as accurate as

The table below includes some of the basic metrics PSP uses in its measurement.[6]

Table 1: PSP Metrics



Metric Type		Description
Product Quality	Total defects/KLOC	the number of defects found in development, per 1000 lines of code
	Test defects/KLOC	the number of defects found in test, per 1000 lines of code
Process Quality	Yield	the percent of defects found before compile
	Appraisal COQ (Cost of Quality)	the percent of total development time spent in design review and code review
	Failure COQ	the percent of total development time spent in compile and test
	Total COQ	Appraisal COQ + Failure COQ
	A/F R	$\frac{\text{Appraisal COQ}}{\text{Failure COQ}}$
	Review rate - LOC/hour	the number of lines of code reviewed per hour in a review (code review or design review)
	Defect removal rate - defects/hour	the rate at which defects are removed in a defect removal phase (design review, code review, compile, test)

3. COMPUTATIONAL PLATFORMS

After deciding on possible metrics that can be used to measure the data collected, we need to look at where we will measure it, that is a computational platform that we can use to collect and analyse data. Given that there are many different metrics that use many different 'units', there is a multitude of platforms available with different functionalities to suit different metrics. The following examples of computational platforms represent some examples of how no one platform can measure or collect all types of data that managers may want to analyse. There are plenty more examples that could be used these are just some of the open source and most notable examples that spring to mind.

3.1 UNIT TESTS

For testing and code coverage, many IDE's and programming languages have their own built platforms used for testing the code. Unit testing is a widely used way to look at how well a program is running. Junit tests are something I am very familiar with from using Java for the LCA and DAGs assignments earlier in the module. Junit is an open source testing framework for Java programming and is a prime example of languages having their own platform to test source code and easily identify errors and bugs in the code which helps the development of programs.

[6] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.3505&rep=rep1&type=pdf

3.2 PSP- BEFORE THE PLATFORMS

Although PSP is not a technically a ‘platform’, rather a method of data collection, it is worth mentioning under the heading of computational platforms simply to give an idea of how data was once collected and the issues that accompanied this method. Such issues include PSP being very vulnerable to human error due data having to be entered manually it was easy to make a mistake or alter data already submitted. This resulted in a large amount of overhead to be required to maintain the quality of the data and reduce the amount of human error which inevitably lead on to the creation of actual computational platforms for software measurement, some of which will be discussed below.

3.3 GIT

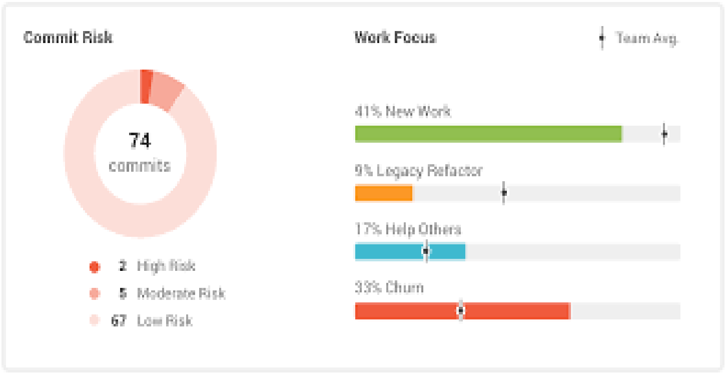
Git is also a useful platform to measure engineering.

3.3.1 GITHUB

GitHub is an open source version control system, although it’s primary function is not to asses or measure software development data it does have some useful functionality that allows you to track developers commits on a project and keep track of how often someone commits, how many commits a developer adds how many lines they commit and it can display graphs and statistics relating to how team members compare in terms of amount of commits and amount of lines commits. Git is a useful tool especially when trying to measure source lines of code. However, git may not be a very fair indicator of a developers contribution for example many companies use pair programming as a method of programming in teams, but this means that only one person is getting gits ‘recognition’ for work they produced together which is an unjust indication to the higher ups of how much effort has been put into the work by each developer

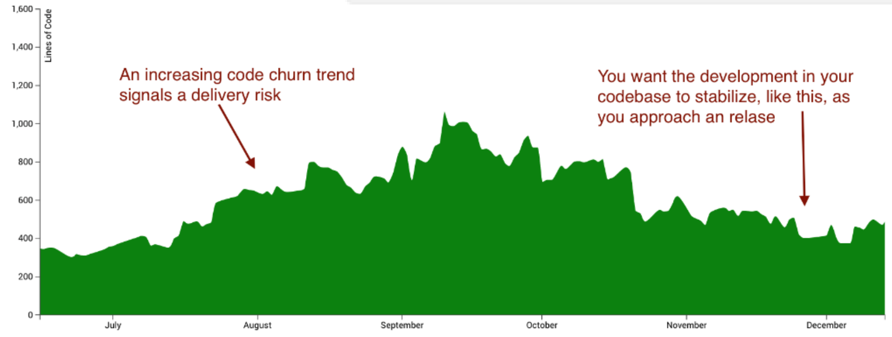
3.3.2 GITPRIME

GitPrime is a computational platform that can be used to provide information and data on metrics used to measure software engineering. It is linked with every major code repository such as GitHub, Bitbucket, GitLab, etc. It collects data from git repos, ticketing systems and pull requests and presents this data as reports.[7]



3.4 CODESCENE

CodeScene offers a useful platform for measuring code churn that provides not one but two metrics being, lines of code added and lines of code deleted. Code churn is one of the metrics with less computational platforms available for analysing it and CodeScene is one of the scarce platforms that does it. Using these two metrics it builds a graphical representation of the average code churn that can be useful when looking to identify patterns or trends in the software engineering process. Studying these trends can be useful for preventing bigger problems developing by inspecting the trends and finding ways to reduce the code churn and reduce the risk of not being able to deliver on time[8]. The core of CodeScene is based on machine learning which will be discussed later as an algorithmic approach for software engineering measurement.



3.5 LEAP TOOLKIT

LEAP, the Lightweight, Empirical, Anti-measurement dysfunction and Portable software process, is a platform that is a step on from PSP in that it deals with the problem of the quality of the data by making the data analysis automated. LEAP performs many forms of analysis on the dataset automatically and can then be used to calculate the size of the product, defects found, time estimations along with checklists and more.

Another feature LEAP has is that it provides personal repositories it provides for each developer to make it easier for them to move their personal data from project to project. LEAP still has the drawback of human error being a possibility as data still has to be entered manually which is also very time consuming and means less time is being spent ion actual work.

[7] https://www.getapp.com/business-intelligence-analytics-software/a/gitprime/

[8] https://codescene.io/docs/guides/technical/code-churn.html

The whole idea behind developing LEAP was to produce tools and techniques to support software process improvement based on them being:

- Light-weight: easy to learn, easy to integrate with existing methods and tools and not impose additional overheads on developers
- Empirical: it should be quantitative and qualitative, that is improvements should be able to be shown through various measurements like effort, defects, size, etc.
- Automated: empirically-based developer improvement requires some level of automation but automation does not guarantee light-weight process or empirically-based improvement
- Portable: it should accommodate for regular job or company changes, i.e. improvement mechanisms cannot be linked to organizations in such a way that a developer will lose their data and tools if they leave a job.[9]

3.6 HACKYSTAT

Hackystat is a platform that attempts to deal with the issues that arise with manual data entry by changing its approach to data collection. Unlike LEAP and PSP, that chose metrics to measure before collecting the data from developers relating to that metric, Hackystat's approach is to gather the data and then perform some analysis on this data to determine what metrics would be useful to measure. To do this, Hackystat does not require any data to be entered manually, instead it automatically collects large amounts of data from the background and caching it. After analysing the data, the platform produces an easy to understand graph for individuals or teams to access.

This computational platform is not without its flaws though as developers themselves were not completely on board with the idea of their data being automatically collected. The fact that managers could so easily have access to the vast amount of data collected on each developer raised some concerns with the ethics behind the data collection methods. The process may be automated, but at the end of the day the information is still easily accessible to developers' peers and this can cause unease for some engineers about having their sensitive information so open to judgement and analysis

4. ALGORITHMS

As well as computational platforms, there is also a number of algorithmic approaches available which have been to use to measure data in the software engineering process. Among the metrics discussed earlier, complexity is one which relies on different algorithms to measure this data, namely cyclomatic complexity and Halstead's complexity measure.

The algorithmic method is designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and historical data and use inputs such as Source Lines of Code

[9]OPQ Version 2: An Architecture for Distributed, Real-Time, High Performance Power Data Acquisition, Analysis, and Visualization.

(SLOC), number of functions to perform, and other cost drivers such as language, design methodology, skill-levels, risk assessments, etc.

General advantages of algorithmic methods: [10]

1. Able to generate repeatable estimations.

2. Easy to modify input data, refine and customize formulas.

3. Efficient and able to support a family of estimations or a sensitivity analysis.

4. Objectively calibrated to previous experience.

General disadvantages of algorithmic methods:[4]

1. Unable to deal with exceptional conditions, such as exceptional personnel in any software cost estimating exercises, exceptional teamwork, and an exceptional match between skill-levels and tasks.

2. Poor sizing inputs and inaccurate cost driver rating will result in inaccurate estimation.

3. Some experience and factors cannot be easily quantified. Expert Judgment Method

4.1 ALGORITHMS IN USE

Although it may be a challenge to decide on a metric to measure engineering on, collecting the data is actually far easier than interpreting it. It can be difficult to determine what the data actually means and concluding results is not an easy task. You can collect all the data in the world, it is all useless unless you know how to pull something useful out of it. This section will be looking at algorithms that can be used to characterise performance and turn the data collected by the computational platforms mentioned above into a useful piece of information.

4.1.1. CYCLOMATIC COMPLEXITY

As mentioned previously cyclomatic complexity is a metric used to measure software engineering. When we are looking at algorithmic approaches for engineering measurement it is worth looking at the algorithm behind cyclomatic complexity.

To calculate the cyclomatic complexity, a control flow graph of the program is used. Each control flow graph displays many edges and many nodes. It is also possible to calculate the cyclomatic complexity with respect to functions, modules, methods or classes within a program. A mathematical formula is used to calculate the cyclomatic complexity where nodes represents processing tasks and edges represent the control flow between them. There are tools available to carry out this work such as OCLint or devMetrics. Working out the cyclometric complexity can be useful as it can aid software engineers reduce complexity in the development and testing of the software. The following is the formula used to calculate cyclomatic complexity.[11]

```
Cyclomatic complexity = E - N + 2*P
where,
  E = number of edges in the flow graph.
```

[10] https://pdfs.semanticscholar.org/4935/410e22756a262e41614747e0d94291cf860b.pdf

[11] https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

N = number of nodes in the flow graph.

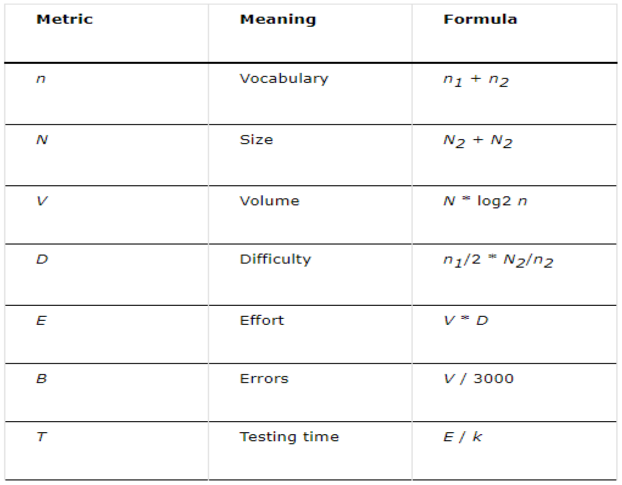
P = number of nodes that have exit points

4.1.2. HALSTEAD COMPLEXITY

Maurice Howard Halstead observed that software measurement metrics should reflect the implementation of algorithms in varying languages but they should at the same time be independent of their execution on different platforms. Such metrics depends on the actual implementation of are computed directly from the operators and operands, or ‘tokens’, in the program source code, in a static manner. Base measures collected include:[12]

- Number of distinct operators(n1)
- Number of distinct operands(n2)
- Total number of operators(N1)
- Total number of operands(N2)
- Number of potential operators(n1*)
- Number of potential operands(n2*)

Each metric mentioned earlier has its own algorithmic formula. Below is a table taken from an IBM source describing each of the formulas.[13]



Metric	Meaning	Formula
n	Vocabulary	$n_1 + n_2$
N	Size	$N_2 + N_2$
V	Volume	N * log2 n
D	Difficulty	$n_1/2 * N_2/n_2$
E	Effort	$V * D$
B	Errors	V / 3000
T	Testing time	E / k

[14]

4.2 LOOKING INTO THE FUTURE- MACHINE LEARNING

[12] https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm

[13] https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.2/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

[14] https://code.google.com/archive/p/hackystat/wikis/Tutorial_GuidedTour.wiki

In recent times, machine learning is becoming increasingly more relevant. Machine learning has become one of the main approaches to analysing data and this section of the report will deal with the algorithms relevant to how machine learning can analyse the data collected using the likes of the computational platforms mentioned in the previous section.

Machine learning is growing increasingly more powerful and due to the rise in popularity of Big Data, it is not surprising that people are relying more on machine learning to make predictions and calculated suggestions based on these large amounts of data. The likes of Spotify and Netflix are prime examples of how machine learning algorithms are used to analyse data and make suggestions based on the results of running algorithmic calculations on that data. One major and obvious advantage that machine learning has over human analysis is the fact that it is completely unbiased.

When we talk about machine learning we can split it into three categories: Supervised Learning, unsupervised learning and reinforcement learning. These algorithms are easily used to measure software engineering which will be discussed in the following points.

4.3.1 SUPERVISED LEARNING[15]

Tis algorithmic process involves ‘training’ the algorithm based on some input to predict some well-defined output. The algorithm is presented with a test dataset and requested to map out some desired output, for example given a set of photos and asked to separate them based on different objects in the photos. The algorithm then creates a set of rules that will map the given input to the desired output. It us then up to humans to select the most accurate model the algorithm produced, once the ‘training’ is complete the system using the algorithms should have the ability to use produce an accurate output for any unseen input dataset presented to it.

4.3.2 UNSUPERVISED LEARNING[12]

Unsupervised learning is similar to supervised learning in the sense that the goal is to create the most accurate model possible to complete an assigned task however, unlike supervised learning, unsupervised learning does not work towards producing a desired output.

It can be split further into three separate algorithms:

- **Clustering algorithms:** These are algorithms tat attempt to identify and separate observations into different groups. Examples include algorithms such as K-means, hierarchal clustering or mixture models

- **Dimensional reduction algorithms:** These are algorithms that look to find the best representation of the data being analysed with as few dimensions as possible. Examples include PCA, ICA and autoencoder.

- **Anomaly detection algorithms**: These algorithms try to find outliers in the data, that is observations that do not follow the data patterns.

[15] https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

12

Unsupervised learning s most useful in cases where the difficulty is discovering implicit relationships in a dataset where items are not pre-assigned, that is to uncover hidden structures. It is not used nearly as much as supervised learning due to it being quite difficult to implement. The majority of unsupervised learning algorithms are used for pre-processing the data during the analysis or used to pre-train supervised learning algorithms as the internal or hidden structure of the data may provide extra information on how to best reproduce the desired outputs supervised learning aims to create.

4.3.3 REINFORCEMENT LEARNING[12]

These algorithms aim to find the best ways to earn the most rewards such as; winning a game, earning money or beating opponents. Reinforcement learning looks for these best ways through trial and error and works to finish the task at hand step by step and it will backtrack if the decision produces a negative response and try a different step to produce a more positive response.

Reinforcement learning algorithms follow different circular steps to produce a result:

- The machine learning agent will choose the action that will maximise result given an environment and state or explore a new possibility
- These chosen actions will alter the environments and agent states and be interpreted into a reward for the agent

This loop is performed repeatedly and allows the agents behaviour to improve.

4.3 DECISION MAKING

All of the above algorithms have their own advantages and disadvantages, however if we consider them, I term of decision-making abilities, Machine learning seems to lead that group. The unbiased nature of machine learning is an obvious advantage for the algorithmic approach. The alternative algorithms discussed may be useful, but the decisions will inevitable be human based which could be seen as a shortcoming of them. In other words, the algorithms are doing the work but they stop just short of making the decisions that machine learning makes for us. Some may argue that a human input or overview is a good thing however, the stance this report will take is that the more unbiased the algorithm, the better as it removes the possibility of clouded judgements and lays a playing field for all those being measured in the process at hand.[16]

4.4 SCALABILITY

In terms of scalability, Machine learning once again takes the biscuit. Due to the whole concept behind machine learning being based off the increase in popularity of 'Big Data'[17] it is quite obvious that the algorithms utilized are highly scalable. This is not to say the other algorithms couldn't handle larger data set, the simple fact is that the larger the data set the more likely a mathematical algorithm is to have errors whereas machines learning constant growth approach deals better with larger data sets and is less likely to be effected by errors that may be in the data set or that would occur when using algorithms such as cyclomatic complexity. It can be argued that all algorithms

[12]

[16] https://www.raeng.org.uk/publications/responses/algorithms-in-decision-making

[17] https://dzone.com/articles/what-scalable-machine-learning

should in theory be scalable, however this may not translate into reality. Machine learning algorithms most definitely have the most potential to be highly scalable.

5.ETHICS

When we talk about measuring software engineering and collecting data relating to developers of software, it is impossible to ignore the fact that it raises a few ethical alarm bells. The main concern is in relation to the invasion of privacy of the software engineers being examined. It is important to look at how these ethical concerns effect workers, the idea of being constantly monitored can have a negative effect on workers as it may have a negative impact on their happiness and wellbeing.

5.1.AS A JUDGEMENT SCALE

Some developers may resent if their employees use the measurement of their software engineering as the sole indicator of how good of a worker they are. Many of the metrics used for software engineering analysis ignore the human interaction side to working on software development projects, that is to say that yes, their engineering skills are important but there are other aspects to working on projects. For example, when working in team it is important to have good communication skills and be able to co-ordinate yourself with others. There could be a case where in a group one programmer may not discuss anything with the rest of the team and continue to work on the project which the measurement algorithms will pick up on and decide that they are the 'best' in the team, when in fact they are asking life harder for their colleagues due to them possibly working on similar parts of the project or developing aspects that do not synchronize with the work the isolated developer is charging ahead with. Of course, software engineering should play some role in the judgement of a software engineers' performance, but other aspects should be taken into account too. This is basically saying that it is difficult to understand human behaviour using quantitative analytics.

5.1.1 VITALITY CURVE

This is a performance management system pioneered by GE's Jack Welch in which employees are ranked against each other in which a '20-70-10' system is in place that is the top 20% of the workforce are top performers, the next 70% are adequate and the bottom 10% should be fired[18]. This did prove to have a hugely negative effect on the moral of employees. Some well-known companies that have been known to use this vitality curve system for evaluating their staff includes[19]:

- Yahoo
- IBM
- Motorola

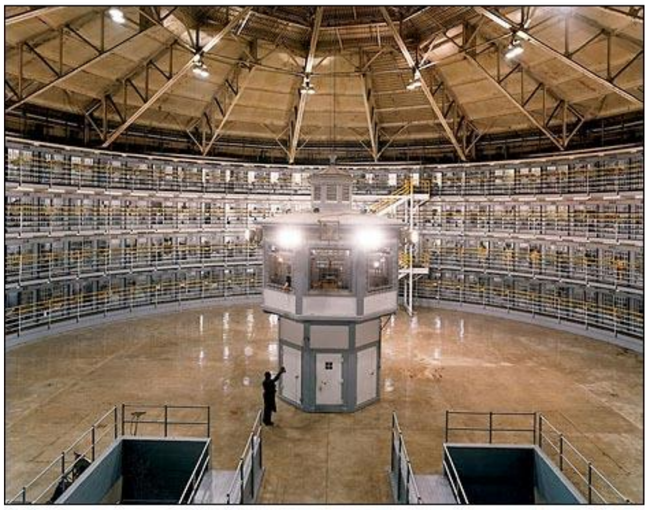
[18] https://creative.artisantalent.com/why-jack-welchs-10-rule-is-100-ridiculous

[19] https://en.wikipedia.org/wiki/Vitality_curve

This is the exact kind of ethical issues that can arise from measuring software engineering by constantly monitoring and collecting data on employees without taking real life aspects into account. By simply firing the lowest performing workers, you could end up unjustly letting someone with tremendous communication or leadership skills slip away and damage the productivity of the over all team.

5.2. PANOPTICON

The panopticon is a disciplinary concept that revolves around the idea that prisoners in jail cells never know if they are being watched that was introduced by English philosopher Jeremy Bentham. The idea is that an observation tower is strategically placed in the centre of a cell block surrounded by a circle of individual cells. The tower guard has the ability to see into every cell but the cells are protected by one sided glass, so the guard can see in nut the prisoners cannot see it. This is basically to ensure the good behaviour at all times under the pretences that they could at any particular minute being observed.


doug duBois & jim goldberg NYTImes 9-22-2002

This idea in recent times has made its way into the software development industry. Shoshanna Zuboff, a philosopher and psychologist, outlined the PC's role as an "informational panopticon" for monitoring the amount of work being complete by an individual, essential analysing their software engineering. Employers can easily get computational platforms to automatically and constantly track keystrokes of workers to measure how much work, effort and hours they are dedicating to work. This method of monitoring is very much a one-way source of information, much as the panopticon was a one-way surveillance idea that is, the employee can be monitored but cannot see when or what is being monitored. This would no doubt create an ethical unease for workers at the idea of being under constant and intense monitoring. The main ethical questions arising from this panopticon idea is the lack of transparency and one-way communication. [20]

5.3.GAMIFICATION

"Compared to people who are unhappy, it has been found that people who are happy have 37% higher work productivity and 300% higher creativity."

This was the findings of a study by the Hitachi. Some companies have acknowledged the negative impact monitoring their employs can have on their productivity due to it reducing their overall happiness in the work place and have started to introduce this idea of 'gamifying' the software engineering process. The idea basically revolves around the aim of making the software engineering process more fun while also encouraging productivity by turning the process into a 'game'. By creating a game out of the process, employees may be more likely to over look their ethical concerns about having their work monitored especially if a rewards or achievements system is put in place. For example, a paper by L. Singer and K. Schneider[21], they discussed how they encouraged students to frequently commit code by turning it into a game like system with weekly team updates, ability to gain achievements and a leaderboard. The results found that commits did increase which is a positive result. The aim of measuring software

[20] https://ethics.org.au/ethics-explainer-panopticon-what-is-the-panopticon-effect/

[21] https://www.computer.org/csdl/proceedings-article/gas/2012/06225927/12OmNyQYt7y

engineering is t find ways to improve productivity, and by gamifying the development process, productivity improved in this study.

6. CONCLUSION

In conclusion, there are a vast amount of metrics that can be used to measure software engineering as a process or as a product. The collection of such metrics has been made possible through the use of a vast variety of different computational platforms available in the industry that each use their own unique algorithms to calculate and compute useful information pertaining to the data collected. The growth in machine learning as a technology is on the rise and will o doubt have an impact on the way in which algorithmic approaches are used to analyse data and metrics used to measure software engineering. However, this does not fix the ethical issues that surround the idea of constant and automatic data collection that seems to be the central point for obtaining the data in question.

The whole idea of measuring software engineering is improving the process. However, it can be argued that the measurement of metrics is counter intuitive in that the act of measuring such metrics takes away from the work itself. Whether it be taking up a developer's time due to having to manually enter the information like with PSP or the fact that employees may be distracted by the fact that they are over-aware that they are being monitored and this may have a negative effect on their focus and productivity. In any case, it can be said that no process can be fully measured to its fullest simply because measuring the process will lead to a lower performance resulting from attempting to measure it in the first place. In other words, it is a waste of time and resources attempting to measure software engineering and companies would be better off allocating such resources to software engineering itself rather than the measurement of it.

MEASURING SOFTWARE ENGINEERING

AMY PIERCE, CSB, 17330305

BIBLIOGRAPHY

L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," Games and Software Engineering (GAS), 2012 2nd International Workshop on, Zurich, 2012, pp. 5-8.

Christe, Anthony & Negrashov, Sergey & Johnson, Philip & Nakahodo, Dylan & Badke, David & Aghalarpour, David. (2017). OPQ Version 2: An Architecture for Distributed, Real-Time, High Performance Power Data Acquisition, Analysis, and Visualization. 1415-1419. 10.1109/CYBER.2017.8446111.

Citeseerx.ist.psu.edu. (2019). *Download Limit Exceeded*. [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.3505&rep=rep1&type=pdf

Code.google.com. (2019). *Google Code Archive - Long-term storage for Google Code Project Hosting.*. [online] Available at: https://code.google.com/archive/p/hackystat/wikis/Tutorial_GuidedTour.wiki

Codes), C. (2019). *Essentials of Machine Learning Algorithms (with Python and R Codes)*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

Codescene.io. (2019). *Code Churn — CodeScene 1 Documentation*. [online] Available at: https://codescene.io/docs/guides/technical/code-churn.html

Cs.utexas.edu. (2019). *E.W. Dijkstra Archive: On the cruelty of really teaching computing science (EWD 1036)*. [online] Available at: https://www.cs.utexas.edu/users/EWD/transcriptions/EWD10xx/EWD1036.html

GeeksforGeeks. (2019). *Software Engineering | Halstead's Software Metrics - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

GitPrime. (2019). *Why Code Churn Matters | GitPrime Blog*. [online] Available at: https://blog.gitprime.com/why-code-churn-matters/

Ibm.com. (2019). *IBM Knowledge Center*. [online] Available at: https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.2/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

Pdfs.semanticscholar.org. (2019). [online] Available at: https://pdfs.semanticscholar.org/4935/410e22756a262e41614747e0d94291cf860b.pdf

Perforce Software. (2019). *What Is Cyclomatic Complexity? | Perforce Software*. [online] Available at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity

Stackify. (2019). *What are Software Metrics? Examples & Best Practices*. [online] Available at: https://stackify.com/track-software-metrics/

Trionfo, M., Silverman, A., Mundy, J. and Morris, J. (2019). *GitPrime Pricing, Features, Reviews & Comparison of Alternatives*. [online] GetApp. Available at: https://www.getapp.com/business-intelligence-analytics-software/a/gitprime/

Tutorialspoint.com. (2019). *Cyclomatic Complexity - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

Tutorialspoint.com. (2019). *Software Design Complexity - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm

Blog, S. (2019). *Ethics Explainer: The Panopticon - What is the panopticon effect?*. [online] THE ETHICS CENTRE. Available at: https://ethics.org.au/ethics-explainer-panopticon-what-is-the-panopticon-effect/

Computer.org. (2019). *CSDL | IEEE Computer Society*. [online] Available at: https://www.computer.org/csdl/proceedings-article/gas/2012/06225927/12OmNyQYt7y

Creative.artisantalent.com. (2019). *Guest Blog: Why Jack Welch's 10% Rule is 100% Ridiculous*. [online] Available at: https://creative.artisantalent.com/why-jack-welchs-10-rule-is-100-ridiculous

En.wikipedia.org. (2019). *Vitality curve*. [online] Available at: https://en.wikipedia.org/wiki/Vitality_curve

dzone.com. (2019). *What is Scalable Machine Learning? - DZone Big Data*. [online] Available at: https://dzone.com/articles/what-scalable-machine-learning

Raeng.org.uk. (2019). [online] Available at: https://www.raeng.org.uk/publications/responses/algorithms-in-decision-making