Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Measuring Software Engineering

Hasan Opiev
17331415

Professor Stephen Barrett

Contents

To deliver a report that considers 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 ethics concerns surrounding this kind of analytics.

Software engineering is one of the fastest growing professions in modern days. According to Evans Data Corporation, there were 23 million software developers in 2018, this number is expected to reach 26,4 million by the end of 2019 and 27,7 million by 2023 [1] . With this continuous increasing of software engineers in society, we have started identifying ways in which we can measure and compare the productivity of software engineers and distinguish those who are productive, and those who are not. In this report, we will discuss this measurable data, the platforms and algorithmic approaches that are available for us to use in performing this work, and the ethics concerned with gathering such vast and somewhat private information.

Measurement metrics

Measuring productivity can be tricky. There are a lot of factors that could potentially affect the work and productivity of a software engineer. The difficult part of measuring this productivity is deciding what data to track and analyse to form your conclusions. Many would argue that;

> *"there's no good way to measure software development efficiency and productivity" and that it's best to measure "things that have a positive or negative effect on productivity" -Vlad giverts, Head of engineering for Clara Lending LTD* [2]

Others will argue that even attempting to measure engineer's data can have negative implications in regards to employee motivation and productivity that will ultimately lead to poorer results for the firm. Regardless of your school of thought, there are still metrics out there that provide management with insight into a software engineers productivity and output.

Lines of code (LOC):

Lines of code is a metric used to evaluate a piece of software in accordance to its size (the amount of lines it contains). It is a simple way to measure programmer productivity. Although it is agreed across the board that this is a weak, simple and flawed metric, it is a metric that was used and is still somewhat used. Bill Gates famously said, in regards to this metric:

> *"Measuring programming progress by lines of code is like measuring aircraft building progress by weight". - Bill Gates*

There are different measures for productivity in regards to lines of code per day. They range from 10 LOC per day according to some, all the way up to 125 LOC per day (for small projects)[3] . But as mentioned, this is a flawed measure. An example scenario is as follows; Programmer A, B, and C are given a similar task. A and B sit down and start coding a solution. Programmer C goes for a walk, contemplates possible solutions for the task, has lunch etc, and then returns to the office to work on a solution he/she has figured out (e.g similar solution present in code base but slight adjustments required). Not much longer, he/she has changed the existing code and has presented a working solution, even though he/she has spent significantly less time and has done it in 2000 less lines of code. Who has been the most productive of the engineers. A, B or C? Of course it is C. This may be a made up example, but mirrors the real world in many ways.

In the context of software engineering, using the amount of code written as a metric for productivity could have detrimental effects. An excess amount of code may indicate inefficiencies and poor performance, where as too little may be an indicator for both lackluster code, or perfect efficiency and creativity. There is too much uncertainty surrounding this form of measurement. Hence, another reason why it is seen as a quite useless metric in today's standards [4].

> *"The speed at which a developer produces lines of code can be an excellent indicator — of keyboarding skills. When it comes to measuring the impact of their work, not so much."*

Commit Count/Speed:

Every developer has their unique style of working. Different software engineers have different tendencies and approaches. Some take longer to commit, or commit more often than others. Teams often have their own projects and set tasks. Often, we see that there is a competition between engineers to complete the set tasks as quickly as possible, as they may believe that this indicates that they are doing something right. In reality, commit counts or the speed at which commits are made is not a reliable metric when looking to measure the productivity of a developer. Some may finish quicker and attempt to move on to the next task, while others will take their time and ensure that their code will require less reworking. This form of measurement ignores the differences of the tasks of developers and gives no context as to why some developers may be taking longer to commit or not committing as often. E.g a project in the early stages of its life will have more commits and faster commits., whereas an already developed and late stage project will be the opposite. Looking at commit data will not provide this valuable insight.

There are tools available to accurately interpret this type of data. Tools such as gitClear analyze repositories to calculate output and productivity, using data such as commit counts and frequency to reach conclusions. We will discuss this tool later in the report.

Line Impact:

"Line impact is the single, reliable metric we use to clarify developer work." These are the words of the gitClear team. According to them, it is a performance metric that offers visibility into what gets done in regards to code, and who codes them. It is a distant cousin of the "Lines of code" metric, in that each line of code has the potential to accumulate line impact. The major difference between these two metrics (Lines of code & Line impact) is that one holds no real meaning or value, while the latter was built with the goal to measure meaningful changes [5].

A metric such as this, may prove extremely valuable to management. With gitClear's use of the line impact metric, the productivity and efficiency of developers will be on full blast for management to see, critique, and evaluate. Unlike the Lines of code metric, Line impact takes all/most important factors into the equation. As such, it is safe to say that it is a more suitable method of measuring the software engineering process. Implementers of the metric generally leave positive feedback, therefore reinforcing the fact that it is what we believe it to be [5].

> *GitClear has been an invaluable tool for us as it allows my leadership team to have in-depth insight into the performance of their developers who are geographically spread throughout the world. I would have far less confidence in my understanding of my organization without it. - Rocket WIlliams, VP Engineer Trintect* [5]

These are only some of the methods used to measure a programmers productivity and output. After management identifies what they want to measure, a new conundrum emerges. How do they measure it?

Computational platforms:

In order to achieve the desired results in measuring engineering, the selected data must be gathered, classified, and analyzed. Management is always looking for ways to gather, classify, and analyse this data in a manner that is cost effective, and produces (accurately) the desired results. There are many computational platforms available to management to aid in the measurement of the software engineering process.

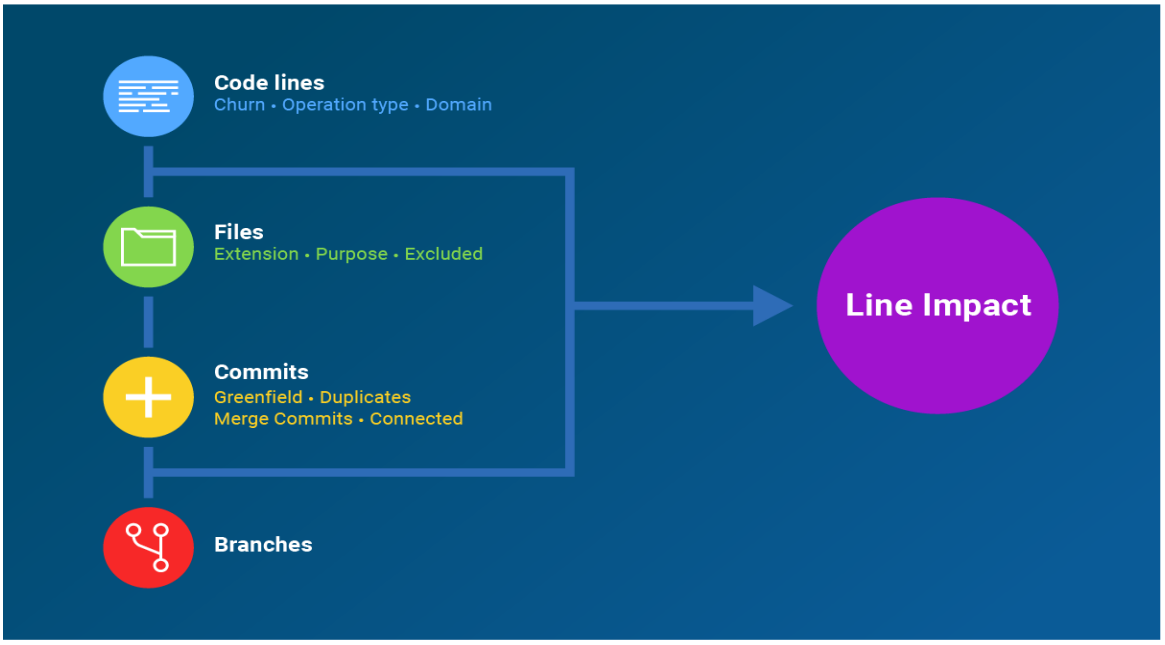
gitClear:

In early 2017, GitClear debuted with a single-minded focus on being the best solution for technical managers and developers who review code. To best serve the needs of its technical users, GitClear is the only company to offer robust code review tools, including a diff viewer that groups similar commits together [6]. GitClear is a tool that analyzes Git repositories to calculate team output and actionable opportunities. GitClear uses several different data points to get a complete picture of the value of a commit. It combines all of these data points to produce a new metric which they call "line impact" (discussed above). The platform provides managers "visibility into what's getting done in the code, and by who".

It provides insight into individual and team performance and productivity by providing metrics and data such as;

- Matching developers to their strengths
- Pinpointing technical debt
- Spotting productivity dead zones
- Evaluating new hires without the wait
- Giving better annual reviews
- Inspect pull request activity and trends

[6]



[6]

PSP (personal software process):

The norm of today is the test and fix methodology, which encourages developers to write code, test it, and fix it until the written tests are passed. This is often argued against, as many state that excessive amounts of time are spent doing this, and to no real avail, as code would still be littered with bugs and errors. Enter Watts Humphrey.

Watts Humphrey is the creator of the Personal Software Process. This was a new way of providing software quality while simultaneously increasing the productivity of developers. The Personal Software Process (PSP) is a software development process that is created to help software engineers better understand and improve their performance by keeping track of their predicted and actual development of their source code [7]. In the abstract of his report, Humphrey gives a summary of what the PSP is:

The Personal Software ProcessSM (PSPSM) provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work...The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects. When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. When used with the Team Software ProcessSM (TSPSM), the PSP has been effective in helping engineers achieve these objectives. [7]

The concept of PSP is based around Humphrey's belief that developers should aim developing quality code from the get-go, instead of relying on test code to do the task. PSP puts forward a number of principles, as seen in humphrey's report:

> *The PSP design is based on the following planning and quality principles:*
>
> - *Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.*
> - *To consistently improve their performance, engineers must personally use well-defined and measured processes.*
> - *To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by mistake; engineers must strive to do quality work.*
> - *It costs less to find and fix defects earlier in the process than later.*
> - *it is more efficient to prevent defects than to find and fix them.*
> - *The right way is always the fastest and cheapest way to do a job* [7]

The process has been taught and implemented in many universities and firms throughout the world. The results confirm/ point to the fact that the PSP can indeed increase productivity.(Using PSP has been measured to: PSP can raise productivity by 21.2% and quality by 31.2%) [7]

Hackystat:

As good as the Personal Software Process sounds, it is still a long and painful manual process. Hence, to no surprise, an automated alternative is available.

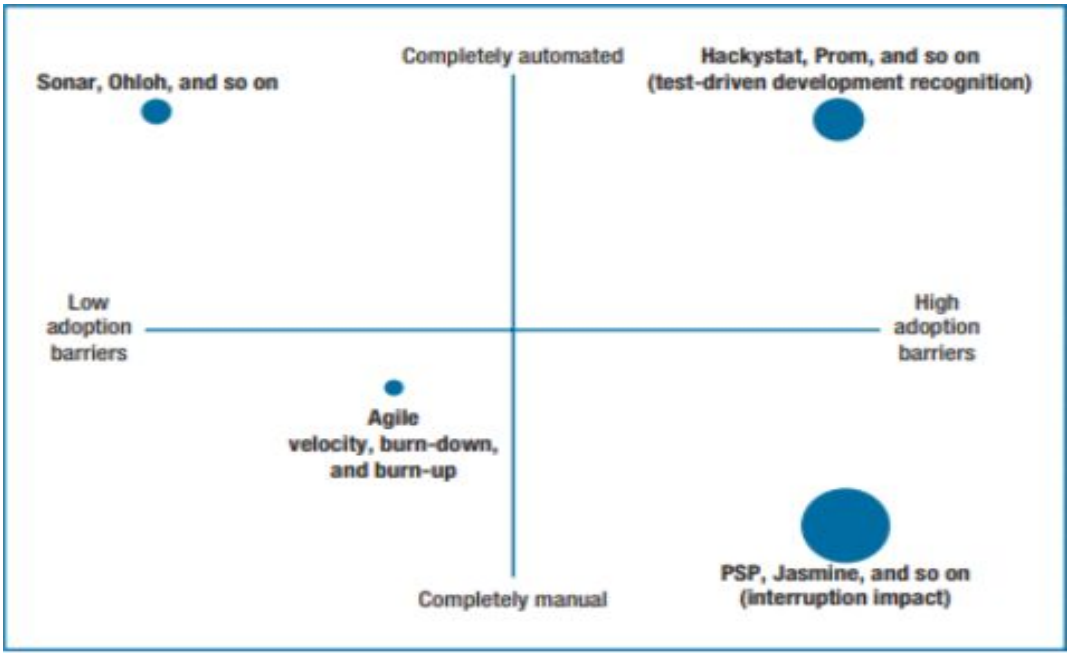
Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. The Hackystat Framework supports three software development communities:

- Researchers
- Practitioners
- Educators

[8]

It works by allowing users to attach “sensors” to their development tools, which then proceeds to collect and deliver “raw data” about development to a similarly named web service under the name ‘Hackystat SensorBase’ for storage. The Sensorbase repository can be queried to form abstractions of this raw data, and integrate it with other mechanisms[8] .

A summary of several computational platforms may be found below. The platforms are positioned in accordance to the level of automation of the platform in collecting the data, and the difficulty in adopting said platform.



Algorithmic approaches available:

You have now reached the stage where you have collected 'raw' data with the use of your chosen tool or platform. Now what? How is this data interpreted? In order to draw some conclusion from the interpreted data, we need to have some comparison. It is of no use to management to know that developer A spends X amount of time doing Y unless we have something to put up alongside it for comparison. The numbers themselves hold no value. Hence, why there are algorithmic approaches available for analyzing software metrics.

McCabe's Cyclomatic Complexity:

McCabe's cyclomatic complexity is a software quality metric that quantifies the complexity of a software program. Complexity is inferred by measuring the number of linearly independent paths through the program. The higher the number the more complex the code. [9]

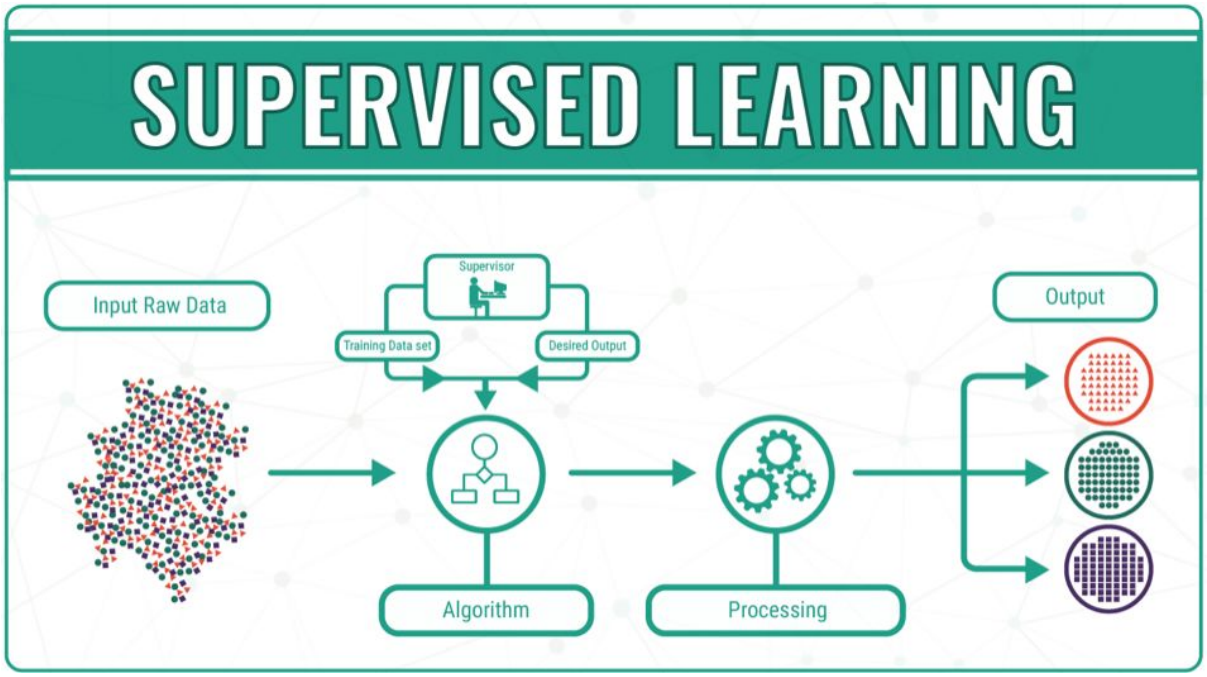
Measuring McCabe's number ensures that the engineers are aware and sensitive to the fact that code that inherits a high McCabe number is difficult to understand and therefore has a higher chance of containing a defect. The complexity number also indicates the number of test cases that would have to be written to execute all paths in a program.[9]

Cyclomatic complexity is derived from the control flow graph of a program as follows:
Cyclomatic complexity (CC) = E - N + 2P
Where:
P = number of disconnected parts of the flow graph (e.g. a calling program and a subroutine)
E = number of edges (transfers of control)
N = number of nodes (sequential group of statements containing only one transfer of control)[9]

Unsupervised learning:

Unsupervised learning is a machine learning technique, where you do not need to supervise the model but rather you allow for it to work on its own to learn and discover information. It typically deals with unlabelled data. [10] Such algorithms allow you to perform more complex tasks compared to supervised learning even though it can be more unpredictable than other learning methods.

Unsupervised machine learning finds all kinds of unknown data patterns from the dataset. In the realm of software engineering, this comes in hand with the data output of developers. Such algorithms are able to derive patterns and regularities and so provide management with necessary information. Unsupervised learning takes place in real time, meaning that data is analysed and labelled in the presence of learners.



[11]

There are different types of unsupervised learning. The most popular of these is Cluster analysis. It is used to locate patterns and grouping from the data analysis (grouping ~ clustering). Clustering algorithms will process the data and find natural clusters if they do actually exist within your dataset. The different types of clustering include:

- Exclusive clustering
- Agglomerative clustering
- Overlapping
- Probabilistic clustering [11]

Halstead's metrics:

Halstead complexity measures are software metrics introduced by Maurice Howard Halstead in 1977 as part of his treatise on establishing an empirical science of software development Halsteads's metric, which is a static analysis of code, is used in many of today's current commercial tools that count software lines of code. Halstead's metric takes into account the length of the code based on the number of operators and operands. By counting tokens and determining which are operators or operands, the base measures that are collected are as follows:

- n1 = Number of distinct operators
- n2 = Number of distinct operands
- N1 = total number of occurrences of operators
- N2 = total number of occurrences of operands[12]

From this, the following definitions were defined;

- Program length: N = N1 + N2

- Program volume: V = N log2 (n1 + n2) (represents vol. Of information necessary to specify a program)
- Specification abstract level: L = (2 * n2) / (n1 *N2)
- Program Effort: E = (n1 + N2 * (N1 + N2) * log2 (n1 + n2)) / (2 * n2) (interpreted ass number of mental discrimination required to implement the program) [13]

Halstead's goal was to identify measurable properties of software, and the relations between them. This is similar to the identification of measurable properties of matter and the relationships between them. Thus his metrics are actually not just complexity metrics.[14]

There is a large number of other available approaches that can be used to interpret the data. These methods will continue to improve and develop, and with this, the grip and control of software developers by management will tighten and increase, possibly creating more problems than solving.

Ethical concerns

In the realm of measuring, collecting, analysing, and interpreting personal data, the topic surrounding ethics is a constant one. What is ethical to track and measure? What can others know and learn about you, without you having much a choice. Sure one can just decide to not work in a firm that monitors workers constantly, but it is not that straightforward. What if you are already working in a firm that then suddenly adopts new, strict employee tracking measures? Do they have the right to know how much value a developer adds to their firm? How far can they push it? All of these are hotly debated topic. At the end of the day, the question that remains is, What is fair game and what is not?

There are different types of metrics. Some metrics are perfectly fair and ethical, such as the ones that are publicly available. These metrics are often not invasive. They metrics do not track or measure the developer at all times, and so pose no real ethical concern. Problems start to arise when we start seeing a constant monitoring of workers. What they are doing, how they are doing it, what did they spend their time doing, where are they doing it and so on. There is a lot of potential to violate an individual's right to privacy. This in turn may have negative consequences as developers may lose motivation which leads to a downfall in productivity and results. Some will argue that employers are entitled to know the ins and outs of their developers in regards to their time spent at work, on company hours and pay. They should have the right to know how much value an individual adds to THEIR company and therefore make necessary changes to ensure the flourishment and prosperity of the firm. This debate is still on going and fiercely contested by all parties. The question of where the line should be drawn will continue for the foreseeable future.

The field of software engineering is not regulated yet. It is not like other fields or professions where there are specific qualifications and certifications necessary to practice. There are some groups and associations that provide an ethical guideline for developers and firms to follow, but these are merely guidelines and are not enforced by a governing body.

It is difficult to determine if measuring engineering processes is ethical or not. There are two sides to the coin again. It is inevitable that firms must collect data on developers in order to improve productivity and results as this would give valuable insight. Platforms such as PSP take a lighter approach to collecting data on developers, where you can have these developers put forward the data they would like to be analysed. Platforms that are not automated face issues, such as inaccuracy and inefficiency. Automated platforms such as hackystat take s different approach. The somewhat orwellian approach is often looked down upon by developers as it allows others to see the working behaviours of others, while simultaneously providing rich insight to management. However , this data may violate developer privacy and confidentiality. This in turn decreases motivation and productivity, as well as happiness. There are countless studies that state that there is a direct correlation between happiness and productivity. Happy workers are more motivated and have and average of 35% higher work productivity. It is the responsibility of the firm to decide on what route to take. Tracking or happiness?

Conclusion

A lot of variables go into the process of measuring software engineering. It is a vast and complex topic that involves analysing and assessing different metrics of measurement, the platforms to collect this measurable data, computing and interpreting this data to derive conclusions, as well as many sensitive ethical concerns with the whole field in general. Many would argue that it is not possible to measure software engineering, due to the large number of intangibles. I find myself disagreeing with this. It is clear that firms that collect, interpret, apply and learn from these metrics, gain a significant advantage over those who do not. It may prove to be challenging, but that is the name of the game.

Bibliography:

[1] https://www.daxx.com/blog/development-trends/number-software-developers-world

[2] https://stackify.com/measuring-software-development-productivity

[3] https://successfulsoftware.net/2017/02/10/how-much-code-can-a-coder-code/

[4] https://www.developer.com/java/other/article.php/988641/Its-Not-About-Lines-of-Code.htm

[5] https://www.gitclear.com/line_impact_factors

[6] https://www.gitclear.com/measuring_code_activity_a_comprehensive_guide_for_the_data_driven

[7] https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf

[8] https://hackystat.github.io/

[9] http://www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php

[10] https://www.guru99.com/unsupervised-machine-learning.html

[11] https://medium.com/@chisoftware/supervised-vs-unsupervised-machine-learning-7f26118d5ee6

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

[13] http://sunnyday.mit.edu/16.355/metrics.pdf

[14] https://en.wikipedia.org/wiki/Halstead_complexity_measures