# Measuring Engineering Report

By Pascal Raos

## 1. Introduction

Science over the course of the last one hundred years is truly a spectacle to behold. All fields of science, maths, business, agriculture, and countless others have been irrevocably changed by technology and at such incredible speeds as to warrant the arrival of problems unique to this technological conversion.

Documentation, communication, and measurement methods have been abandoning a previously paper and physical filing-oriented method in favour of the digital. This has opened up avenues previously unattainable, specifically efficient storage, and technological analysis of data, that can be achieved automatically without user input. One immediate thing we can notice is the application of these technological processes towards the measuring of employee efficiency and quality within different work sectors. For the sake of the report, I will state the three work sector theory that encompasses the roles of manufacturing and business.

Work sectors can be divided into three distinct groups: Primary, that deals with collection of raw materials/foods, Secondary, that deals with the creating of products and utility-based services (electricity, gas), and the Tertiary, which focuses on the selling or manipulation of created or established products (Shops, Banking), primarily focused on person-person interaction and retail.

Now that we have established the areas of interest, we must discuss how this technological foothold within these areas can affect the measurement and qualification of the work done by individuals in these sectors. Firstly, we can intuitively think that for the more physical Primary sector, we can classify performance by quantity and quality of resources acquired over time. Similarly, in the more physical aspects of the Secondary sector we can identify the units of products created (such as electricity, or items on an assembly line), and in the Tertiary, the rate of sales and profit from direct transactions.

Software Engineering, the focus of this report, can be considered a special case when looking at these measuring methods. This is due in part to the process of software engineering. Multiple people work independently and cooperatively to create the very rules to which technology adheres to. Due to this the nature of quality assessment can be quite subjective unless certain code standards and commit procedure is set up. Code assessment is also not always a black or white matter, as many factors need to be considered. It is my aim in this report to highlight the effort put into attempting to standardise these measurements, by method of the creation of unique metrics, and through the use of highly complicated analysis software. For reference details please see the final page of the report.

## 2. Metrics for Measurement

In this section we will explore the different metrics employed by different software companies and contemplate their effectiveness in determining an employee's efficacy, and examine potential benefits and negatives for each. These methods can be split into basic and complex assessments

**Basic Assessment:**

This area focuses on simple metrics that do one job in particular, which can be used alongside others to provide a comprehensive performance analysis.

### 2.1. Quantity of Code Commits

One method of qualitative assessment can be through the number of code commits a programmer pushes over the course of a certain period of time. While I believe this can be a good indication of how much work is being done, it is only effective when combined with other forms of assessment. This is because using only this method, the assessment process can be exploited by meaningless code commits, causing large quantities of poor code being favoured over small amounts of quality commits by another software Engineer.

### 2.2. Quantity of Lines Committed

Similar to "Quantity of Code Commits", this one looks at the amount of lines committed by a software engineer. While it suffers from the same problem of the quality of code being undetermined, quantity of lines is still a better indication than number of commits as you can have hundreds of one line commits, whereas you cannot hide the number of lines.

### 2.3. Structure of Code

Many software companies will provide their software developers with some sort of document stating a company coding standard, this is to ensure indentation, and other code formatting is constant across all separate divisions of the company. This allows a more fluid reading and debugging experience in the peer review process. This is a very effective metric in assessing the competency of a software engineer as an engineer that cannot

follow basic procedure, may be evidence for some lack of skill or competency.

### 2.4. Performance of Code

The most common method of performance measurement when it comes to code is asymptotic notation. It is a form of notation that inspects the cost of every line of code and only acknowledges the highest order of cost. This is known as asymptotic notation and is talked about in terms of Big-O, Theta, or Omega. Where Big-O is the upper bound of the worst-case runtime, Omega represents the lower bound of the software runtime, and Theta is a measurement relative to the other two. Analysing the performance of code is paramount in assessing a person's capabilities, as efficient code is usually indicative of a good software developer. One thing to consider though is that efficient code can be very complex, and sometimes in large group projects where code understanding is important, having overly complex code at 100% efficiency is worse than having simpler code at maybe 60-80% efficiency that your peers can interpret.

### 2.5. Relevant Commits

This is a more advanced form of assessment that would require more setup for measurement and would take place over a period of time. Once an engineer commits a piece of code, it can be measured based on certain criteria and assigned a score. This score will be modified if another engineer or the same engineer edits that specific region of code. This shows evidence that the engineer's initial code had some issues surrounding it. A level of score deduction in this assessment area can be affected by whether the engineer needed someone else to correct their mistake, or if they did it themselves. I believe this is a solid method of assessment, but can have issues in when exploring the meaning of code rewriting. Maybe the initial code fit the specification perfectly, however a change of specification required a change in code. Certain tact must be taken when implementing this form of assessment in order to not wrongfully assess an engineer.

### 2.6. Test Cases Passed

A self-explanatory metric that measures how many test cases that are implemented have passed. The effectiveness of assessment is directly correlated to the quality of tests created, and the edge case coverage of the tests.

**Complex Assessment:**

This area focuses on complex systems of metrics that are either extremely common or have been used by leading tech companies in the field.

## 2.7. Functional Points

Function Point Analysis was initially developed by Allan J. Albercht in 1979 at IBM and its further development can be attributed to the International Function Point Users Group (IFPUG).

Functional Point Analysis is a method of qualitative assessment for software engineering based on the analysis of the coded systems. The process can be explained using five assessment points: external input types, external output types, logical internal file type, external interface file types and external inquiry type. This combined with identifying the complexity (simple, average or complex) and quantity of each point creates five reliable and standardised metrics, that are ultimately combined into a final score.

An **external input type** is a unique user control or file system data input, and its complexity is based on the size of the input and its effect or reliance on the internal system, either through parsing or manipulation of current processes. Similarly, an **external output type** is a unique control or data output and its complexity is assessed in a similar way. A **logical internal file type** is an instance of the application generating its own data or control information that is maintained by the application itself. An **external interface file type** is an instance of another application managing data relevant to the application, and this information can only be retrieved to the original application by an interface. **External inquiries** are the transaction functions of retrieving data.
For every instance of the above functional code, the quantity of each instance and complexity of each instance is tallied and stored in a table.

| *Function type* | *Simple* | *Average* | *Complex* |
|---|---|---|---|
| External input | 3 | 4 | 6 |
| **External output** | 4 | 5 | 7 |
| Logical internal file | 7 | 10 | 15 |
| External interface file | 5 | 7 | 10 |
| External inquiry | 3 | 4 | 6 |

Fig 1. Example Table from source: https://ecomputernotes.com/software-engineering/what-are-function-points-how-are-they-computed-explain

Once the data has been collected, it is analysed following complex formulae (that I do not understand to the extent of being able to communicate properly) and a score is computed to assess the effectiveness of the designed system.

## 2.8. Story Points

Story points are a measurement commonly used in agile software development. In the Sprint Planning Stage of software development, each task is split into inherent subtasks and assigned difficulty points known as story points. These story points are relative measurements such that a task of 2 points takes twice as long as a task assigned 1 point. Similarly, a task of 2 points takes two thirds of the time that a 3 point task takes.

These points are assigned through a method known as a planning poker session.
Each member of the development team is given a set of cards taken from a series of increasing numbers (the Fibonacci sequence is common). For each task the team is asked to present a number that they think relates to the task difficulty. If many people choose the same number a consensus is reached, and that difficulty is applied to the task before continuing with another. If a consensus is not reached, people discuss their reasoning for picking their numbers before another attempt is made. If the general point score for a task is too great, the task may be split into subtasks and their scores evaluated through the same process.

Once all necessary tasks have been assessed work for the sprint can begin. The way that this can be related to a software engineers performance measurement, could be from their contribution to task with proportion to the story point score of the task completed.

## 3. Platforms for Data Analysis

This section focuses on identifying existing software in the marketplace designed for the specific need to measure the software engineering process. From my research I have noticed that most of these measuring platforms look at three similar metrics for their main performance assessment: Cycle Time, Review Time, and Throughput.

Cycle time is defined as the total time from the beginning of a task to the end, and can be applied to the project as a whole, or individual tasks. Review Time is the time spent reviewing system functionality in order to assess the need for change or approval. Throughput is a measure of how much work is being done by the system over a given time, and can be a good measurement for system performance.

### 3.1. Swarmia

Swarmia is a software system dedicated to measuring software engineering, with a final goal to decrease cycle and review time in a company, and increase the overall throughput of the software system.

Swarmia achieves these goals through data collection sent to the managers and CTOs of a company, while informing engineers of goals and necessary tasks through slack notification integration. The company sets up specific software development rules, for example focuses on how long to spend reviewing code, basic testing standards that must be met, keeping up with pull requests and bug ticketing.

The application graphs average time spent on these processes, including cycle times, review times, pull request backlogs and sizes, and provides the data in a streamlined fashion to higher ups within the company for review. Overall, the software helps to identify problems during the development cycle so they can be rectified quickly, while keeping the engineers proactive with useful reminders through quantifying performance and giving them a tangible goal.

### 3.2. HayStack

Haystack is another platform for managing a companies software engineering. It helps track active tasks such as pull requests, cycle time, and sprint progress by analysing repositories using the GitHub API. The system uses heuristic methods (which are not talked about in detail) to assess “each team, repository and member”. If necessary, changes in long term performance can be identified by managers quickly, in order to rectify productive deficiency.

Some of the advertised features are weekly reports using real-time analysis, and the prevention of employee burnout through the use of daily reports and alerts to help give engineers a sense of progress and tangible goals to work towards.

## 4. Methods for Data Analysis

There are many different methods to understanding the received metric data from a company's employee performance assessments. These metrics can range from quite simple to complex systems trying to ensure fairness. Some of the most notable complex methods are Machine Learning and Expert Systems. While simpler methods are counting methods and peer reviews.

### 4.1. Expert Systems

Expert Systems falls under the category of Artificial Intelligence. Expert Systems is a method best used when all known possible inputs into the system can be recognized and fall under a certain amount of pre-specific rules.

A company can predefine set requirements that an engineer must meet across a range of metrics, and an expert system can deduce based on collected metric data, the effective performance of an employee.

Expert Systems require more developer input if the system is to be improved upon, through the action of changing or adding more rules.

### 4.2. Machine Learning

Machine Learning falls under a branch of Artificial Intelligence also and is used in environments with a large amount of uncertainty. The systems can adapt to changes over time and create an overall improved system on its own without extra input.

I could not find any examples online as to how exactly Machine Learning is used for measuring software engineering, as, from what I could find online, most employee measuring software is secretive about the exact specifics of the data analysis. But in general Machine Learning is more malleable than Expert systems and can be used when a large amount of data is being analysed.

### 4.3. Counting Methods

Counting methods consist of very simple data analysis. Essentially, using specific metrics, an engineer's performance is measured solely based on the quantity of their work. This method pertains mostly to things like number of tests passes, number of commits, number of completed tickets, number of lines of code, and other similar metrics. This is a simple way of measuring an employee's performance, however it may not be a good indication of their actual worth in the company as these simple metrics don't usually paint the whole picture.

### 4.4. Peer Reviews

Peer reviews are a more direct method and probably one of the best measures of skills. This is due to the nature of software engineering in the public sector. Most software engineering takes place in team-based scenarios over the course of months to years. This creates a work environment where people are quite familiar the quality of their neighbouring workers, and can assess them on their teamworking, work attitude, and general competence. These are all metrics that are incredibly hard for a computer to deduce to a high level of certainty, and instead they infer on them based on collected data.

The only fault I can find in the peer review process are aspects such as internal corruption affecting the perceived worth of a person. This can come in the forms of personal bias, specifically when co-workers share a particularly fond relationship with each other, or in extreme cases, can be problems with nepotism.

Otherwise, I find that people are generally a good judge of character when it comes to software development, especially in cases where an employee is new and would be considered objectively bad by machine-based assessment.

Most of the automated assessment systems, specifically Machine Learning and Expert Systems have one major problem. That is, they cannot account for real world factors. They do not account for an employee's mental state or other external factors occurring in their personal life.

## 5. Ethical and Moral Concerns

With reference to an article by techbeacon (listed in the reference section), I will address certain concerns with the process of automated assessment, and the difficulty in fair measurement of an engineer's performance.

### 5.1. Weight of Measurement

One thing we must focus on is the weight that these metrics hold in certain companies. If a company hold a software engineers' skill solely to the review of his automatically analysed metrics, two important things need to be considered.

Firstly, that the metrics put in place are carefully chosen to properly convey his/her skill, and secondly, that the analysis of the collected metric values are rigorously analysed using intelligent formula. This is to ensure fair treatment of the engineer, and to assess strengths and weaknesses in such a way that the engineer will not have his employment on the line due to misgivings from subpar evaluation methods.

One of the biggest legal hurdles with using automated systems, is to justify to a court how you let go an employee based on a machine's deduction. A fear of being poorly assessed can have a negative impact on an employee's performance in the company.

In many cases it is good to have a two-step process when reviewing an engineer. A case where computer data is forwarded to dedicated staff that can truly understand it, and combined with some sort of human input, can decide whether certain metrics truly succeed in measuring skill. I believe until we are in some far future situation where technology can understand the human nature of an employee, that we will need some sort of user input to truly assess a software engineer.

### 5.2. The Future and Privacy

I believe in order to have truly perfect measurement of a software engineer, we would need to advance technology to a level where a computer can understand human nature. I believe this creates one big opportunity, but also comes with an extremely worrying price. We could create a world where an engineer's assessment could be entirely unbiased, and a perfect reflection of their skill. However, the means to reach it could be an extreme breach of what we currently accept as standard privacy measures.

Some of these problems could be camera monitoring to recognise facial expressions or detecting presence in online work environments. Using

artificial intelligence to parse company emails to deduce a person's nature and attitude towards their work. In extreme situations, an entire user profile can be accumulated on their mannerisms, preferences, and other personal details. While these could contribute to effectively measuring whether a person is an asset to the company, these invasions of privacy could be used harmfully in the wrong hands and create a feeling of an authoritative privacy-free world.

## 6. Conclusion

To summarise, I believe it is paramount that we create a fair work environment where a person's skills are assessed not only on black and white metrics, but that a person is treated as more than a set of data. Due to software engineering's complexity, there is definitely a need for a strong form of assessment, however a level of tact and circumstantial consideration is required.

Whether this be through human input or the improvement of human understanding by machines, we will not know for now. A certain level of planning must be put in place, along with established, government regulated guidelines when the time comes in order to hold true to our ethical human values.

## 7. References


Explanation of Function Points: https://ecomputernotes.com/software-engineering/what-are-function-points-how-are-they-computed-explain

Measuring Talk, Abi Noda, Senior Product Manager at GitHub: https://www.youtube.com/watch?v=cRJZldsHS3c

Explanation of Story Points: https://www.visual-paradigm.com/scrum/what-is-story-point-in-agile/

Swarmia Software: https://www.swarmia.com/story/matera/

HayStack Software: https://www.usehaystack.io/

AI metric ethics: https://techbeacon.com/app-dev-testing/should-you-use-ai-make-decisions-about-your-software-team