MEASURING SOFTWARE ENGINEERING

Brief - 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.

1 INTRODUCTION

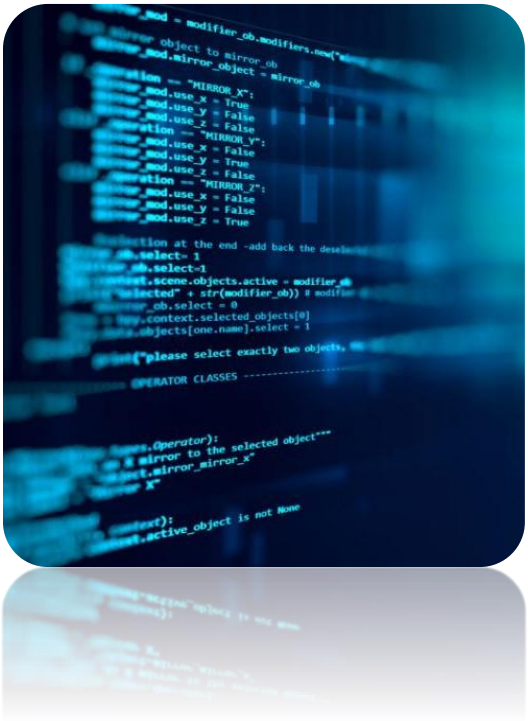
What is Software Engineering?

According to the Bureau of Labour and statistics, Software Engineering is defined as the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

In software there are 2 categories of metrics we use:

Software engineering metrics, also known as software development metrics, or software delivery performance measures how software is being built and the engineering teams' productivity.

Software or application performance metrics are the metrics of software delivered, response time of the application, etc. (Anaxi, 2019)



What does it mean to measure software engineering?

The Software Engineering Process can vary immensely depending on the project being undertaken. We need to consider the type of application that is being developed before we can attempt to judge or measure the software engineering methods and techniques being employed.

This report aims to shine light on the methods used to measure Software Engineering both today and historically.

2 MEASURING DATA

Software Metrics

Margaret Hamilton, lead software engineer of the Apollo Project, stands next to the code she wrote that was used to take humanity to the moon.

'Software metrics' is the rather misleading collective term used to describe the wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterise properties of software code through to models that help predict software resource requirements and software quality. The history of software metrics is almost as old as the history of software engineering. Yet, the extensive research and literature on the subject has had little impact on industrial practice (Norman Fenton, 1999).

Before looking at the current status of software metrics I find it important to consider the past and where we have come from. In the late 1960's the routinely used metric was simply 'Lines of Code' or LOC. LOC was being used to measure programmer productivity and as a metric for measuring quality. Quality was determined by calculating the number of defects per unit of LOC. Fenton and Neil detail the history of software metric and their implementation in the 1999 article 'Software metrics: successes, failures and new directions'. By the mid 1970's the obvious drawbacks of using LOC as a measurement metric became apparent. LOC was a crude method that rewarded inefficient code. The LOC crucially overlooks important aspects of code such as functionality and complexity.

The need for an accurate way of measuring software accuracy caused an explosion of interest in the field in the mid-1970s.

Collecting Data

Companies want to measure time and money to ensure that their processes are maximizing efficiency in these areas. Human capital in the form of a programmer's productivity is no different.

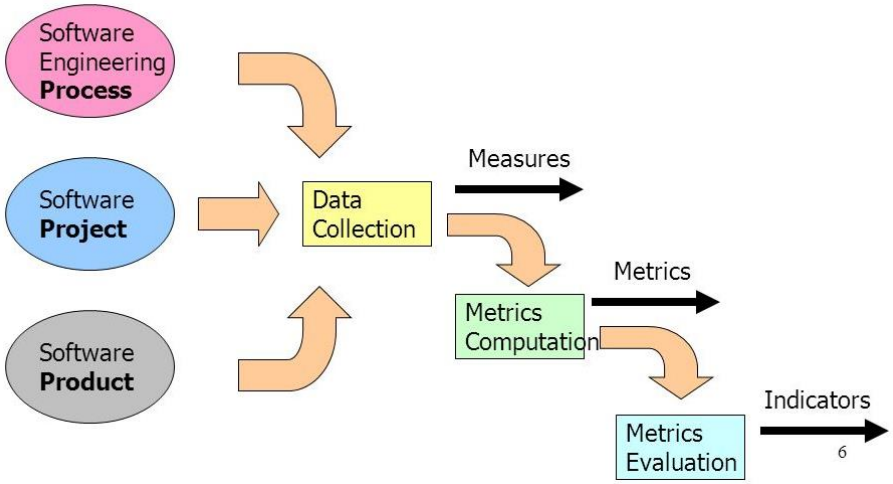
> *The productivity of very good programmers is ten times better than average. For these reasons, it is very important to understand how top developers work and to encourage all developers to adopt a process that helps them to achieve the best possible results.* (A. Sillitti, 2003)

Data must be collected in order in order to monitor and optimise productivity to achieve the collective goals. One of the biggest issues a company faces is how to collect data and what data to collect. In the early days of software engineering manual data collection was commonplace. Manual data collection is time consuming and prone to human error. It was agreed that data

collection should provide information on the product, process and project but the process of collecting data should not cause a strain on resources (such as time and money). (Kan, 2002)

There are **three main measures** required in order to analyse the software engineering process. These were laid out by Kan in 2002 and are as follows:

- Product Metrics
- Process Metrics
- Project Metrics



Measuring Product

Product metrics are mainly broken down into 4 areas - Specification, Design, Coding and testing. These metrics allow engineers to gain a better understanding of their processes and assess the quality of their work.

Measuring Process

When the priority of the assessment is the performance of projects processes/workflow, process software metrics are used. Process metrics involve looking at the steps involved in creating a software product. The internal attributes of a process include effort, time and quantity of changes.

Measuring Project

Project measurement encompasses looking at the resources of a project. It is the easiest metric to evaluate as it is like any other project measured in the workplace. The three main entities which are analysed are personnel, tools and environment.

The Question of Importance

Google Product technology Manager, Steven A. Lowe discusses the modern importance of metrics in software engineering, or lack thereof in his article “Why metrics don’t matter in software development”. Lowe suggests measurements should be designed to answer business questions not software engineering ones. “What matters most is success metrics. To get there, you can indeed use several specific and objective measurements, in combination, to answer real business questions” (Lowe, n.d.). Even though technology has developed massively, the basic ideas that Norman and Fenton touched on in 1999 (that these metrics have very little impact on industrial practice) still holds true.

In a follow up article, Lowe detailed 9 metrics that can make a difference in modern Software Development Teams. Again he stressed that measurements should only be designed to answer business questions and never “How many KLOCs are we up to now?” (Lowe, n.d.). This is comparable to the adage, “Quality over quantity”.

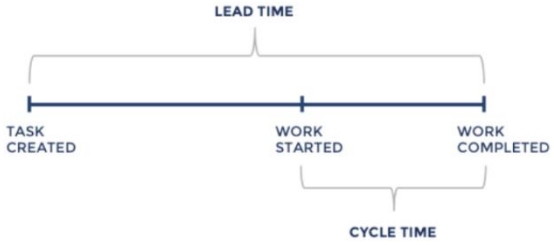
The metrics were as follows:

- **Agile Process Metrics**
  - Leadtime
  - Cycle Time
  - Team Velocity
  - Open/Close rates
- **Production Analytics**
  - Mean Time Between Failures
  - Mean Time to Recover/Repair
  - Application Crash Rate
- **Security Metrics**
  - Endpoint incidents
  - MTTR (mean time to repair)

Agile is a process by which a team can manage a project by breaking it up into several stages and involving constant collaboration with stakeholders and continuous improvement and iteration at every stage. Principles of these methods are different from traditional methods and so there are some different processes and activities required to compare them to traditional methods. ASD methods require different measurement practices.

Agile teams often do their projects in the simplest and most effective way, meaning measurement practices in agile methods are more important than traditional methods, because lack of appropriate and effective measurement practices, will increase the risk of a project (Taghi Javdani, 2012).

Leadtime is used to measure the time taken for an idea to propagate from the requirements planning phase all the way through to deployment. If a software engineering company wants to be more responsive to its customers, it must work on reducing its lead time, typically by simplifying decisions.



Cycle time starts when the actual work begins on the unit and ends when it is ready for delivery. Modern agile teams using continuous delivery can have cycle times measured in minutes and seconds instead of months.

Velocity measures how many units of software a programmer or team typically completes in an iteration. Comparing team velocity would not make sense because the metric is based on non-objective estimates. Instead it is a metric used to plan iterations.

Open/Close rates measure how many production issues are reported and closed within a specific time period.

Mean time between failures (MTBF) is measured as the average of the times between each failure. A high average means that software is performing well, whereas conversely a low average means that the software is failing to perform and requires work. The calculation of this is as follows:

$$MTBF = \frac{\sum(\text{start of downtime} - \text{start of uptime})}{\text{number of failures}}$$

Mean time to recovery (MTTR) is the average time to recover from a failure.

$$MTTR = \frac{\sum(\text{start of uptime} - \text{start of downtime})}{\text{number of failures}}$$

Application crash rate measures how many times an application fails divided by how many times it was used. This metric is closely related to MTBF and MTTR.

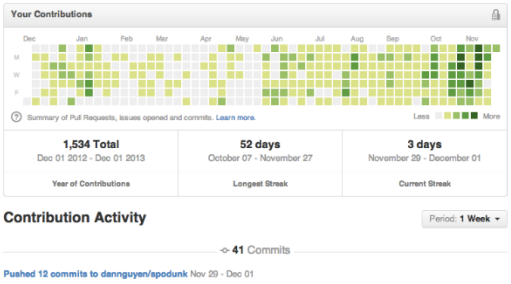
(Lowe, n.d.)

3 COMPUTATIONAL PLATFORMS AVAILABLE

This section addresses the interpretation of data in order to measure the performance of a software engineering project. Once data has been collected, it must be analysed. Computational platforms have been released which collect and analyse software engineering metrics.

For the vast majority of companies, product metrics are defined and dealt with internally. Consequently, we will only be looking at platforms that allow for the analysis of Software Engineering metrics. The desire for companies and managers to quantify and improve productivity has created a lucrative market for companies that can accurately and efficiently calculate insightful and relevant metrics.

GitHub



A screenshot of GitHub visualising a user's 'commit' history.

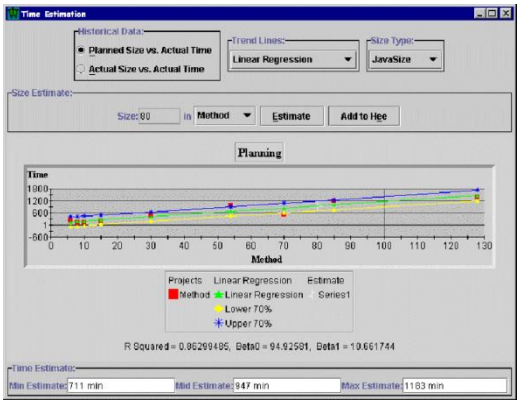
GitHub was developed by Chris Wanstrath, P.J Hyett, Tom Preston-Werner and Scott Chacon in February 2008. In 2018 Microsoft acquired the company for $7.5 Billion.

GitHub is a version control tool which offers all the features and functionalities of Git (the open source brains behind the service) along with its own services. These services include bug tracking, feature requests and wikis. GitHub reports having more than 37 million users and is a clear market leader in the field. There exist projects available on GitHub which utilise the publicly available GitHub API to extract and analyse user project data. The REST API can be downloaded to allow the number of 'commits', number of contributing developers and the frequency of an individual contributors input to be analysed using an analytics software package such as R studio.

GitHub is widely used in professional settings and allows for easy development of measurement applications to record and interpret bespoke metrics regarding Software Engineering performance and the realization of business goals.

Pre-made GitHub addons such as 'Screenful Metrics' allow the creation of visual dashboards that create automated reports based on the metrics obtained from GitHub.

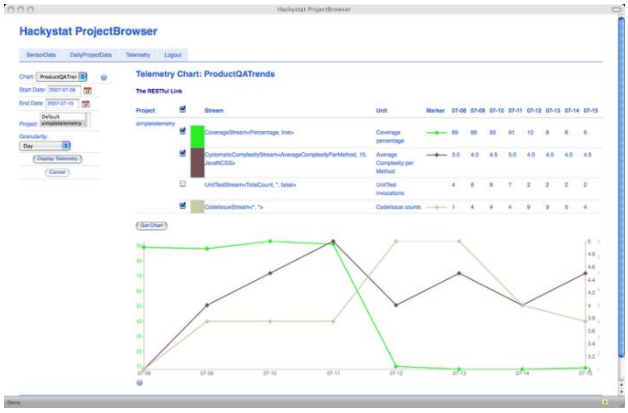
Leap Toolkit



LEAP toolkit is another (albeit older) platform available that aims to automate the data analysis process of an even older process called PSP (Personal Software Process). LEAP stands for Lightweight, Empiracle, Automated, and Portable. This technique still requires a level of manual input from the developer, and therefore is prone to human error.

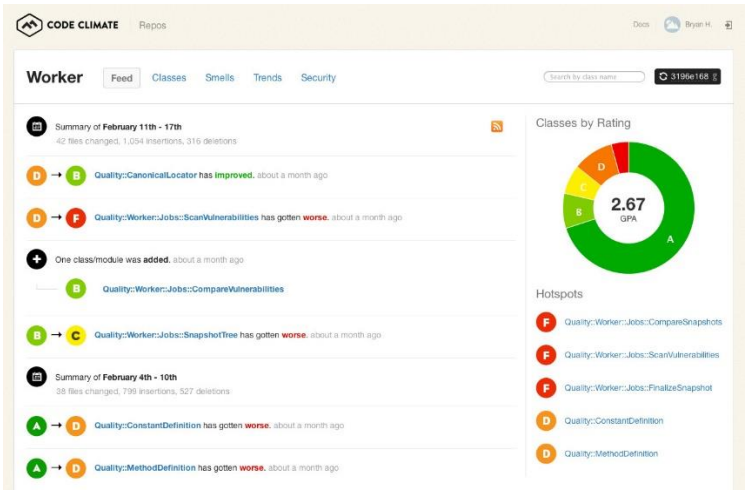
Hackystat

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of the Software Engineering Process. Hackystat works by attaching sensors to the developer tools which unobtrusively gather raw data which is sent to a webservice called the Hackystat "SensorBase" for storage. The automation involved is what set Hackystat apart from tools such as PSP or LEAP. This system and method of data collection gave employers very accurate and concise data regarding productivity but raised many ethical concerns about data collection and privacy. I will touch on these ethical concerns later in the report.

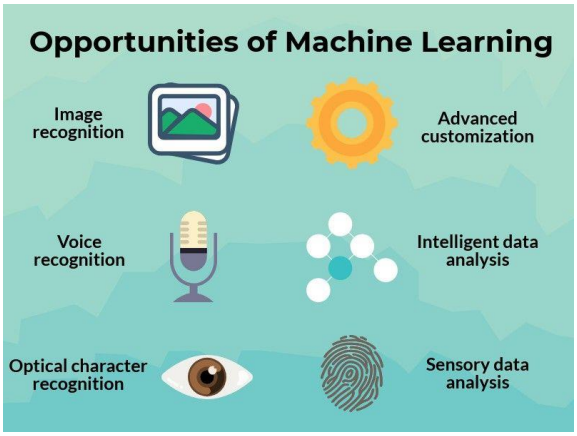


Code Climate

Founded in 2011, Code climate is a one of the relatively modern computational platforms available. Code climate also works in conjunction with Git, and is used to determine the quality of code. After each git push, code is analysed for complexity, duplication and other changes to determine changes in quality and technical difficulty. It sends automated code review comments on the users pull requests, which allows for assessment and potential improvement. The automation of the platform reduces the chance of human error. Code Climate provides an objective analysis of the quality of your code and gives you the information and tools you need to fix it. (Code Climate, 2019)



4 ALGORITHMIC APPROACHES



Engineering Analysis has changed from a manual approach to a more automated one over the past 10 years. The rise of machine learning has enabled computers to analyse information and make decisions themselves. Several different aspects of machine learning can be used in an effort to measure performance during development of software.

Today companies have access to massive volumes of data which can be utilised to aid the training of machine learning algorithms. Machine learning algorithms and techniques are often categorised as supervised or unsupervised.

Simply put, in **supervised learning** there is an input variable and an output variable. An algorithm is used to learn the mapping function from the input to the output. Once the algorithm learns to map well enough it can be used to predict the future outputs for given inputs.

In **unsupervised learning** there is no output data, only input. The goal is for the unsupervised models to explore the data and find some structure within the data in order to learn more about it.

These two exciting methods provide us with great tools for analysing data, such as the code written by a certain developer or analysing the commits on a user's GitHub account. In turn, machine learning is an exciting frontier for software metrics and improving the measurement of Software Engineering.

5 THE ETHICS

There are undoubtedly obvious benefits to business performance and productivity through the use of the methods mentioned above. Though these practices are successful they come with a question of ethics and employer responsibility. The monitoring and storing of employee performance information is common in the workplace today, though the legalities around collecting data are changing. In 2018 the European union released the General Data Protection Regulation which strengthen the rights of those who may be having their data collected. In the field of software engineering the distinction between right and wrong is not as clear as in other traditional fields such as law or medicine.

A striking recent example of the misuse of data was unravelled with the Cambridge Analytica scandal in 2018.

Data analytics firm, Cambridge Analytica, harvested millions of Facebook profiles of US voters and used them to build a powerful software program to predict and influence choices at the ballot box. The scandal brought the worlds attention to the ever-increasing amount of data that is being stored by companies about us (many without our knowledge). (The Gaurdian, 2018)

6 CONCLUSION

This report has outlined the ways in which the software engineering process has and is measured. I believe the field of Software Engineering measurement is a 'No Silver Bullet' situation. There is no perfect way to monitor the productivity of an employee or team, ensuring they are not interrupted - all while being morally conscious. The methods I have discussed however are all in industry use in some form or another, with the aim of encouraging '10X' performance and optimising team efficiency. Efficient, productive teams create better work in a shorter time frame and make companies more money.

The focus of any company that measures their software engineering should be how the analysis will help their employers. It is vital that this monitoring and measurement is unobtrusive, effective and that there is full transparency between management and employees.

7 REFERENCES

A. Sillitti, A., 2003. Collecting, integrating and analyzing software metrics and personal software process data.. *Proceedings of the 29th Euromicro Conference: IEEE.*

Anaxi, 2019. Software Engineering Metrics: The Advanced Guide.

Code Climate, 2019. [Online]
Available at: https://codeclimate.com/quality/

Kan, S., 2002. *Metrics and Models in Software Quality Engineering.* s.l.:s.n.

Lowe, S. A., n.d. *9 metrics that can make a difference to today's software development teams.* [Online]
Available at: https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams

Lowe, S. A., n.d. *Why metrics don't matter in software development (unless you pair them with business goals).* [Online]
Available at: https://techbeacon.com/app-dev-testing/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals

Norman Fenton, M. N., 1999. Software metrics: successes, failures and new directions. *The Journal of Systems and Software.*

Taghi Javdani, H. Z. A. A. A. G. A. B. M. S. ,. R. M. P., 2012. On the Current Measurement Practices in Agile Software Development.

The Gaurdian, 2018. *Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach.* [Online]
Available at: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election
[Accessed 2019].