Measuring Software Engineering

Michael Cullen - 16317188 - CS3012

Software Engineering itself is an art with various aspects to be discussed. The idea of measuring software Engineering is a difficult task in the first place, just like measuring the influence of artwork. According to Vlad Givert (former CTO at Identified) “The truth is, there’s no good way to measure software development efficiency and productivity. But you can measure things that have a positive or negative effect on productivity…” (https://stackify.com/measuring-software-development-productivity/). Therefore the task of measuring Software Engineering as a whole requires a deep insight into the position while also considering the effects of the data being used to display these metrics.

When Software Engineering first began to form into an entity, there was no previous attempts to formalise any measurable data as there wasn’t anything to compare it to. For the Apollo 11 mission to the moon, Margaret Hamilton and her team were credited for coining the term “Software Engineering” back in the early 1960’s. At this time, getting the computers to work in the first place was an achievement in itself. With the increase of teamwork, there needed to be a formalised structure to allow the clean and smooth operation for the mission. Therefore, key measurements were taken to analyze the team as a whole and on an individual level to measure areas such as work completed and code added to the final bundle.

There are many good reasons to measure Software Engineering as it provides a basis for future development and productivity. Nowadays, projects can consist of multiple team members, each providing a different contribution to it. For the employer, it would be very useful to have some sort of statistical knowledge on how that team is performing and where the strengths and weaknesses that lie within the team allow for quick and easy intervention. There are many areas of Software Engineering that need to be considered such as the development / productivity of the team, the quality of the work produced by the team, the efficiency of the team and the overall moral. These four things are required for the smooth running of the development of software. With these things in mind, it is a difficult task to display and measure these statistics as every team will have different tasks and roles to work on. Therefore the measurements need to be adaptable to work with all forms of Software Engineering while considering the ethical implications of the data in the first place. In this essay, I will discuss some of the ideas and possible solutions in existence and review on their impact on the measurement of Software Engineering as a whole. The main topics in discussion are; measurable data, computational platforms, algorithmic approaches and the ethics concerned with this type of data analysis.

Measurable data

There are many aspects of data collection available to parse using APIs such as Github's. Although many of them do not reflect fully the commitment of the work but provide an overall picture of the project with information such as contributors and commits made to that project and even the amount contributed. These statistics help show a generic view of how Software Engineering can be measured. As well as that, a lot of Software Engineering cannot be measured as it may be difficult to compute or is too time consuming to perform. This can include problems encountered within the project or even the quality of the code itself and how well it runs in different environments / platforms. In this section, two areas of Software Engineering measurement will be mainly discussed, measuring the code itself and measuring the work of the individuals that created / made the code.

The project itself can be seen as a source of measurement as it is a deliverable item, therefore it can have various measurable attributes such as the time frame to completion, bug handling, test case and many more. These attributes are key to the software and how they are implemented. Although these can be measured and recorded, it does not mean that they provide accurate and detailed data about certain aspects of the project. By just characterising Software Engineering as a whole by just the project is like judging the performance of a computer by its colour. Further in depth analysis is needed to break down the project into sub-measurable attributes as discussed below.

One of the main areas that are usually considered it the "Commits Made" to a project. This is ultimately seen as the main aspect to measure the performance of Software Engineering but it is not as accurate as it seems. According to Gitlab " Their size and frequency do not correlate with the work needed to achieve that change. At best, commits can be viewed as an indication of activity." This is true for the idea that a commit can be of any size, either one line of code or a large dataset of code altered. Commits are usually undertaken when a piece of code needs to be updated / inserted, therefore they come in all shapes and sizes and even scattered across the whole project. Therefore measuring commits alone may lead to displaying inaccurate measurements of the overall workload.

One inaccurate form of Software Engineering measurement is the "lines wrote" or "SLOC" for the project. Lines wrote refers to the physical lines included to the code bundle. According to Bill Gates (founder of Microsoft Corporation) "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs." Lines of code are not necessarily good indicators of how good or bad the software is and its performance in the real world. Different languages can have different lengths of code to perform the same task. For example Haskell vs Java.

There may also be parts of the code that may never be executed leading to redundant code committed to the bundle which is not an accurate view of the project itself. Along with this, size is another inaccurate measurement of Software Engineering. Although very important as it needs to work within the specifications of the system its running on it can vary from language to language and with different techniques can alter the size of the bundle.

A hidden aspect of Software Engineering is software testing. Code that runs in the outside world and needs to be tested for every eventually and error scenario. If not, the code might cause serious problems and then may have to be reviewed again adding time and cost to the business. Although this a major area of Software Engineering, it may not necessarily be the most enjoyable. To aid the speed of testing code, test cases are created created, which is a piece of the code that tests the functionality of the deployable code to make sure that it can handle its requirements and any error case it may encounter. Is also provides useful feedback that can easily be reviewed and tracked such as the line where the error occurred or the result expected for the test with the result returned from the code.

Bugs are an inherent part to Software Development, Measurement cannot be solely characterised by the amount of bugs introduced by the code but rather by the time spent fixing these bugs. This includes the process of first identifying where the bug is, creating an adequate solution to fix it and and then implementing a solution to finally rectify the bug. If there is a large proportion of time dedicated to fixing these bugs it may reduce the productivity as it is a time consuming and tedious task to complete. Therefore the measurement of bugs can be shown as the percentage of the uptime the product is to the public without any errors occuring.

Measuring the code is one aspect of Software Engineering, but reviewing the Software Engineers is another area to assess. At the end of the day, these are the people who create and develop the software. There are many factors to be considered when measuring Software Engineers such as their surroundings, daily routines, personal engagement and their involvement in teamwork.

One measurable aspect of Software Engineering is the time spent on a particular task. According to tasktop.com, On Average 6.2 (+-3.3) minutes are spent on each task per hour. Graphs like Burndown Charts which is a graphical representation of work left to do versus time can be used to accurately measure the time per task in terms of the whole project. Along with this Slack can be used to order and prioritise these tasks so that more of the time can be dedicated to the higher priority tasks rather than the lower priority tasks in the project. At the completion stage, the overall project time can be measured and reviewed and this can give an overall picture of the areas where the most time was dedicated or left out. Along with measuring time as a

whole, the time spent context switching (Task switching) can be reviewed Task switching refers to the amount of times the task was changed to a new task within a period of time. According to tasktop.com, On average Task switches occur 13.3(+-8.5) times per hour. At this rate, it shows the complexity of Software Engineering as a whole and it shows the time spent per task is very minimal with work not being completed in large loads but as small incremental commits.

Performance and productivity felt among the Engineers is a key measurement of Software Engineering. The work they produce needs to be of the highest standard so their time and resources need to be allocated in an efficient manner that provides the best results for the project. In a review carried out by tasktop.com, they consulted a sample group of Software Engineers on when they feel most productive within a given day. The table below shows the results of the most common answers and the percentage of people who felt that the given task was productive and worth their time.

Task	**Percentage %**
Task completed on time	53%
Attending Meetings	50%
Given Clear goals	20%
Coding	~75%
https://www.tasktop.com/blog/measure-productivity-software-development-team/	

As shown by the data, there are many areas of Software Engineering that need to be considered outside of Software Development itself.

The work of the individual can also be measured. This can include their own personal projects and their contribution to group projects. This data can show the diversity of the Software Engineer and how they interact with other group members. As Software Engineering is expanding, the requirement for group work has become more a necessity then a want. Therefore the contributor size to a project can be measured as a whole and be broken down into the sections that each individual has worked on.

Computational platforms

There are a vast array of platforms available for software engineers to help aid the development of Software Engineering. These platforms provide a service for mainly gathering and representing data it in a user friendly fashion. The metrics discussed above are the main topics of these platforms as they are they provide only factual information that can be processed from the code bundle and any attribute in relation to it.

GitHub

GitHub is one of the main Computational platforms as it can gather a wide range of analytics from the code and the users associated with it. GitHub consists of a website with a cloud-based service that helps developers store and manage their code, as well as track and control changes to their code repos. It provides analytics on the amount of contributors, dates and times of commits and even the parts that were updated by each person for each repository with the frequency in the system. By doing this, they can provide useful graphs that visualise the data in an aesthetically pleasing manner. Github also provide an API that allows users to parse their own data if the ones provided are not sufficient enough. This data would be extremely useful for team based projects as they provide insight to the efficiency and the contribution of the team by showing the areas of the code altered by each team member.

Bitbucket and Agile

Atlassian Bitbucket is another version of Github with the integrated support of Agile. Agile refers to an approach to software development where the project is developed in incremental steps rather than a complete finished product. “It does not matter how slowly you go as long as you do not stop.” – Confucius. (https://blog.versionone.com/10-agile-quotes-from-the-worlds-most-brilliant-minds/)
Scrum is a form of agile software development that break down Software Engineers work into actions that can be completed within timeboxed slots, called "sprints", within a timeframe of two weeks upto a month, that can track progress and re-plan in 15-minute stand-up meetings called daily “scrums”. Within Bitbucket, they provide “Burndown” metrics on this data recorded throughout the sprint. These sprints can be also combined to show an “Epic and release burndown chart” that shows all the points per sprint accumulated by the team to give an overall view of the scope of the project. One of the problem with this is the changing nature of the project such as an injection of more requirements into a previously-defined project.

Scrutinizer

Scrutinizer is a web based application that lies on top of Github and Bitbucket that provided an extra layer of data analytics that both Github and/or Bitbucket do not provide. These include analyzing the code for issues ranging from critical security issues, bugs, to coding-style issues, and fix them before the code goes for production use.

Integrated Development Environments

Integrated Development Environments (IDEs) provide their own form of parsing metric data. For example Eclipse has a plugin called EclEmma that checks the code coverage of a program. This can include code coverage with tests, time of execution and error scenarios such as run time errors with the line number at fault. These tools can be used with the development process as it provides instant feedback to the Software Engineers to allow them to fix the errors in real time rather than down the line which would add extra time and money to the company. But the data is lost when the project is re-compiled or changed, therefore it is only useful for “on the spot” measurements.

Hackystat

Hackystat is an open source framework that uses the data it collects to analyze multiple metrics of Software Engineers. Hackystat uses “sensors” to extract the raw data from the development and sends it to its repository where it is interrogated and abstracted to any view that is required. To do this the “sensors” are incorporated into the work environment of the Software Engineer and the data is collected as the project is updated. This provides a more in depth analysis of the data as Hackystat can show more information then Github or Bitbucket as it has direct access to the work environment of the project. These results are then processed to generate graphs that can be reviewed and discussed. Therefore Hackystat can access data such as the number of build attempts, number of runtime errors, active time which allows the Software Engineer to analyze the statistics in real time rather than after committing the bundle.

Personal Software Process

Finally, Personal Software Process (PSP) is another form of a software development process that is aimed to help Software Engineers better understand and improve their own performance by tracking their own predicted development and their actual development of the code. PSP works incrementally by providing a detailed view on how to plan, measure, and manage their work. It is designed for work with any programming language and for most aspects of software work, including requirements, tests cases, defining processes, and repairing defects within the code itself.

Algorithmic approaches

With the measurements listed above, Algorithmic processes can be applied to them to allow the data to be turned into useful information. When computing these, security and accuracy need to be topmost priority as these deal with sensitive data. Many approaches can be taken such as the simple analytics of commits, overall time, and lines of code. These may suffice for some analytical processing but these do not suffice as an overall measure of Software Engineering. Computational Intelligence on the other hand may provide an overall solution that may “best fit” Software Engineering as a whole rather than piecing parts of it together and trying to measure it.

Many of the statistical analysis such as the frequency of commits and times can be easily calculated by the adding up all the committed sections of that user. Similarly, Lines of code can be calculated in the same fashion by adding up all the lines of code present in the file. This form of process is bland as it only parses the obvious data that is given to it and does not provide any real indication of what the real indications of measuring Software Engineering are. As discussed above, as well as being a simplistic calculation, the use for these metrics are also very limited as they only provide a one dimensional view of the code by only analysing data that is too broad to be characterised as solid facts.

A code churn is another approach of data processing. Code churning is a measure of the rate that the development is evolving. This is the next step up of statistical analysis as not the calculations are made on all the lines of code, all the modified lines of code and deleted lines of code for every commit of the project. This form of analysis is more intense then before as there needs to be records of the previous modified files. Another simple measurement is to implement a “lines of code executed” checker is to run the given tests provided by the Software Engineer and map each line of code that needs to be executed. At the end of the program, all the lines should be marked as hit, otherwise there exists a section of code that is not covered by the tests. Although useful, these forms of data analysis are very one dimensional as they only provide a service for one area of measurement rather than an overall calculation. With this, the overall result of measurement may not be as definitive as the results are only pieced together to form different sources rather than an overall processing of the data. This is where Computational Intelligence comes into light.

Computational Intelligence is another approach that can be further discussed. Computer Intelligence refers to “the ability of a computer to learn a specific task from data or experimental observation” (https://en.wikipedia.org/wiki/Computatonal_intelligence). As there are many aspects of Software Engineering to be measured, this can provide an overall “blanket” solution to the idea of measuring Software Engineering as a

whole. The two main forms of Computational Intelligence I will discuss are; Fuzzy Logic, Neural Networks.

1. Fuzzy Logic - Allows the system to leave the bounds of binary and to reasonably approximate answers within a range rather than a definitive answer, as Software Engineering is no right or wrong ways to measure it. It all lies with the interpretation of the results and what aspects are more interesting than others. With these approximations, it allows the system to develop and overall “best fit” answer rather than a right / wrong answer at the end.

2. Neural Networks - Allows the system to develop and learn from the data and adapt to predicting future variations of the data. While not being an algorithm, but more of a framework to allow connected noted to be aggregated to give a route to the answer. Neural networks can be compared to natural life forms such as plants and animals, as they too adapt to their surroundings and make logical interventions based upon them. This is very useful in the area of Software Engineering measurements as it takes into account all the data and determines what data to use to interpret with practice and time. As it if was a lifeform learning a new habit.

When looking at Computational Intelligence from a wider scope, it can be seen to fit into measuring Software Engineering as a whole as it gets rid of the dependencies of single point sources of data and computes an overall picture of Software Engineering. With Neural networks, it allows the process of measuring Software Engineering to adapt to the changing nature of it in the real world. By doing this, not only is the Software engineer measured, but also their work and any other measurable attribute that can be computed. Therefore, with all the information gathered, comparisons can be made such as “What constitutes a good Software Engineer” or “What measurable attributes are more concerning than others”. This can shed light onto the higher order questions posed by measuring Software Engineering rather than the easy “low hanging fruit” questions. Therefore companies may use this technology to use this statistics and data to form their own interpretation of an “ideal Software Engineer” as up to this point only single attributes were taken into account rather than the suitability as a whole for the role.

Finally, Cyclomatic complexity is an algorithm that determines the complexity of a program. The overall aim of Cyclomatic complexity is to calculate the amount of paths within the code and it deciphers it complexity by converting the program into a tree structure. The less paths, the easier the code is to understand. The formula to calculate Cyclomatic complexity is :

$$M = E – N + 2$$

where ;
C is the Complexity
E is the edges of the graph
N is the nodes of that graph

This form of measurement is very useful as it is independent of the lines of code wrote in the program and also the language that writes it. This Algorithm provides an unbiased measurement on the paths available for that particular program. Therefore, it can only measure the if / else statements, branch statements and switch statements, so along with the algorithms above, it provides only one service to measure Software Engineering rather than measuring Software Engineering as a whole.

Although, some of the solutions listed above provide simple straight forward solutions while others provide an overall picture. It all comes down to the interpretation and requirements of these results. Companies may only be concerned with certain aspects of Software Engineering rather than a generic solution. Therefore, although simplistic, these calculations may be the only algorithmic approaches businesses can afford to compute as large Computational Intelligence systems may not return their investment for small businesses.

Ethical concerns

Ethics is concerned with moral values and concepts of the conduction of an activity. Ethics play a key role in the development of software Engineering as product that is being delivered will be directly in contact with the outside world. These can range from privacy to personal information and to the use of the data stored. These concepts need to be discussed beforehand to analyze whether they are suitable for the requirements provided by the project.

One aspect of ethical concern is the wellbeing of the Software Engineers, if treated in an unfair manner or not being abided by their rights as a human, the workflow within the company will drop. Therefore both employers and employees need to stand on common ground so they both can develop the task at hand. If the employees are constantly being monitored, it may lead to a pressurised environment within the workplace as the employees may feel like the work that they produce needs to be of the highest standards and anything less may result in punishment. For example, if a software team comes short of its targets, it may reduce the motivation for the team as the main focus was on the given task rather than the mental wellbeing of the team creating it.

Another area to consider is the projection of these statistics. If shown accurately, it may cause ethical issues within the workplace as the employees may feel pressured to keep to the same standards as their fellow employees. And by doing this, the higher achieved employees may get financial benefits / rewards and the others workers do not. This is not true for the case where the “high achiever” had an easy task to complete whereas everyone else may have had a tougher challenge taking more time to formalize an answer. Therefore, the overall measurement of Software Engineering may be difficult as every Software Engineer is different and may excel in one area and not in others. This is a major influence into the discussion of Software measurement as there is a standard for the code but not for the Engineers who create the code.

Storage of this data and access to that data is one of the biggest ethical concerns of this era. “Every day, we create 2.5 quintillion bytes of data” (https://bigdata-madesimple.com/exciting-facts-and-findings-about-big-data/). This data needs to be safely secured with secure access to it. If this data was to leak, it would be a major infringement on data security and the data holder could be faced with heavy fines. With so much data collected in our daily lives, it needs to be accurately parsed and stored so that only the people involved can access it. “Developers and researchers must weigh the trade-off between easily obtained analytics and richer analytics with privacy and overhead concerns”. (P. M. Johnson, "Searching under the Streetlight for Useful Software Analytics," in *IEEE Software*, vol. 30, no. , pp. 57-63, 2013. doi:10.1109/MS.2013.69). Therefore, the use of this data needs to be gathered within reason and not for use outside the use of solely measuring Software Engineering only.

One final ethical concern is the need for the collection of this type of data in the first place. If the employee does not consent to the collect of their own data, it will create anomalies with the final dataset. But the employee has the right to refuse. Therefore, tension can be created by the forcefull need of the collection of data making the workplace a hostile environment where it as seen the metrics are more important then the workers themselves. “a happy worker is a productive worker”. On that note, there needs to be a balance between the need for this statistical knowledge and the rights and wellbeing and security of the Software Engineers who are being measured.

Conclusion

In conclusion, there are many ways in which Software Engineering can be measured and displayed but none of the approaches above quite give a universal fit to an overall solution to measuring Software Engineering as a whole. Each of the outlined sections only seem to either focus on one particular aspect of Software Engineering and uncover their findings in that particular field or theoretically provide an overall solution. In this view, not all these sources of measurement techniques will suit the needs of every business. Along with this, if a business decided to use multiple aspects of measurement such as lines committed and bugs per test, they will require extra overhead as they are both computed on separate platforms leading to a data retrieval across multiple platforms. In my view, this can become very messy and unorganised for a large team as to gather all this data in a neat and orderly fashion would be a difficult task itself, not even taking into account the accuracy of this merge and the ethical impact of doing this in the first place.

In my opinion, Software Engineering can only be measured on the combination of some of the previously mentioned points such as, measuring the uptime of the code in relation to the bugs rate followed by the amount of code testing implemented within a controlled time frame. All these combined and stored securely to reduce ethical concern would help show the progression of Software Engineering as a whole. Software Engineering overall is a difficult thing to measure at the best of times as it is ever expanding and developing to the needs of the society, the business and the changing environment surrounding it. With this, The introduction of Computational Intelligence in the future may provide an overall solution but only time will tell. All these factors are a major external influence to Software Engineering and are very difficult to measure as they are ever changing. Therefore, although there is not a definite solution to the outlined question that masks this, it allows the companies themselves to decide and have the freedom to pick themselves what they find important aspects of Software Engineering while not being bombarded by endless pieces of useless data points that are not relevant to them.

Sources

https://bigdata-madesimple.com/exciting-facts-and-findings-about-big-data/
https://stackify.com/measuring-software -development-productivity/
P. M. Johnson, "Searching under the Streetlight for Useful Software Analytics," in *IEEE Software*, vol. 30, no. , pp. 57-63, 2013. doi:10.1109/MS.2013.69)
https://blog.versionone.com/10-agile-quotes-from-the-worlds-most-brilliant-minds/
https://www.tasktop.com/blog/measure-productivity-software-development-team/
https://stackify.com/measuring-software-development-productivity/
https://en.wikipedia.org/wiki/Software_measurement
http://ecomputernotes.com/software-engineering/software-measurement
https://pubsonline.informs.org/doi/10.1287/isre.8.1.95
https://en.wikiversity.org/wiki/Software_metrics_and_measurement
http://www.scitepress.org/Papers/2017/63144/63144.pdf
http://www.inf.puc-rio.br/~inf2921/2013_2/docs/aulas/INF2921_aula5.pdf
https://www.froglogic.com/blog/tip-of-the-week/what-is-cyclomatic-complexity/