2021

Measuring Software Engineering

CSU33012

THEO STEPHENS KEHOE – STUDENT NO: 19334945

INTRODUCTION

In this report I will be describing the software engineering process and the ways in which it can be assessed. I will describe the types of measurable data that can be extracted from the process and then will discuss some of the platforms available today that make use of the data. I will then discuss the ethics surrounding the measurement of the process.

THE SOFTWARE ENGINEERING PROCESS

Before we can delve into analysing and describing the Software Engineering process, we must first define it. There are many definitions of software engineering that can be found, but it can be best described as "the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines" [Fritz Bauer]. Put simply, it is the creation of software used in industries through engineering principles. The discipline differs from traditional computer science as the software created is focused on usability and real-world application, rather than coming up with algorithms or designs for theoretical concerns.

The discipline was first introduced around the 1960s by Margaret Hamilton, who came up with the term to give the discipline legitimacy, as the idea of it had been around for a period of time. The NATO sponsored conferences on software engineering in 1968 and 1969 in Germany is seen by many to have marked the official start of the profession of software engineering. It is therefore a relatively young discipline and profession. However, as we know in the present day, it has evolved quite significantly – the meaning of being a software engineer nowadays differs quite drastically from what it meant back in the 1960s. Software engineers now are concerned with all areas of the production of the software – from the initial design to how it is deployed to industries. They are also required to be able to work in a team, or at least be able to communicate with other software engineers, as most software today is created through companies or small teams. It is a software engineer's job to engage with other engineers to move software from one production stage to the next – this is what we define as the software engineering process. It is imperative then for the engineers to be able to measure their own individual performance and the performance of their team as they carry out this process, so that they can find their weaknesses and strengths and restructure themselves accordingly. Through this, they can improve their efficiency and produce a better service for their clients. This analysis can be done by observing and performing calculations on a number of quantifiable, measurable data. In this report I will be discussing this analysis and the challenges that come with it.

MEASURABLE DATA AND THE ANALYSIS OF THE SOFTWARE ENGINEERING PROCESS

It has been known since its inception that the software engineering process was going to be difficult to accurately measure and predict. After all, we are not measuring the performance of a machine or an algorithm, but a person. Many argue that it cannot be measured at all as software engineers do not produce singular objects that can be easily defined, and that software engineering does not follow a simple production line. It has been observed that as the development process is so complex, there is no simple metric that perfectly describes the productivity of a software engineer. However, this does not mean that there are no metrics that can be used to gauge some outline of how a software engineer performs.

Before we can observe some of the most common metrics used, we must first understand the Software Development Life Cycle (SDLC) – a model that is commonly applied to projects/productions in order to divide the development process into distinct phases. The model is as follows:

1. Planning: Usually performed by the senior members of the team, information is gathered on the goal of the project and the functionality of the software being designed. The information is sourced from domain experts and market surveys. The information is then used to set out a plan of action.
2. Defining: Once the basic plan of the project is outlined, the software being produced is clearly defined and documented. This is usually done through a *Software Requirement Specification (SRS)*.
3. Designing: The phase in which by the architecture of the software is designed and specified. Usually more than one design approach for the software is created and the entire approach is proposed in a *Design Document Specification (DDS)*. The DDS is then reviewed by all important members of the project for approval.
4. Building: The phase whereby the actual development of the project starts, and the software is built. Code is generated as per the DDS.
5. Testing: The code is tested rigorously for errors and defects, which are all reported, tracked, tested again, and fixed.
6. Deployment: The software is finalised and is then deployed to the market/customer.

Although this is a cyclical waterfall model, it is key to note that in reality many stages of this model may take place simultaneously for a project. As stated before, software engineering **does not** follow the form of a simple production line. Testing may take place at the same time as another component of the project is being deployed or defined. Errors and inefficiencies may be corrected as a component is being further developed upon. The definition of the software being required may change as the project progresses, in response to newfound knowledge about the performance of the software. The software engineering process is dynamic and is always changing, and rarely follows a simple linear progression. It is however still beneficial to apply such a model as it gives us a rough path of how the software should generally develop. It also allows for all engineers within a team to know what stage the team is at, regardless of whether if they individually are working on tasks attributed to that stage or not.

As we now know the most common layout for the software engineering process, we can now discuss the measurable data that can be extracted to measure the software engineering activity. This measurable data can be divided into two categories: quantitative and qualitative. I will be discussing a few examples of each.

Quantitative Data

Lines of Code: A generic measurement, this metric simple analyses how much code a software engineer has produced for a given project. This metric may be useful in defining who has generally contributed the most code for large projects, but it gives very little indication of the impact of the code. One engineer may produce hundreds of lines of code while another may only produce a few lines but achieve the same functionality – they have both made the same impact.

Lead Time: Defined as the time taken between the creation of the product specification and the fulfilment of said specification, this is a useful metric as it can be used to see how fast a team or individual completes a project. The problem with this metric however is that it does not encompass how difficult it is to implement the specification or the quality of the software produced. A team may be able to complete a project relatively quickly, but the product produced may perform worse compared to a team who developed it slower.

Burndown: Almost an antidote to the problems of lead time, this metric is developed by splitting tasks into 'story points', where each story point is estimated to take the same amount of time to complete. This is useful as if a team/engineer is seen to be taking too long to complete a set amount of story points compared to usual, it could indicate poor development or problems occurring. The problem with this metric is that it disregards the development style of the team/individual, as they may develop code in non-linear amounts of time.

Commits: If a team or engineer is making use of a repository, statistics can be measured on the commits made. It can give an indication of the changes made and the pattern of an engineer's development style.

It is clear from the examples of some of the quantitative data that each metric presents problems of their own but can give a general view of how a software engineer or team performs in some regard.

Qualitative Data

Qualitative data is used in measuring the qualitative aspects of software engineering – it is specific to the impact of the code on the project. Metrics such as number of tests passed for committed code, code coverage and the percentage of lines of code that are executed all give good indications of the performance of the code an engineer/team produces. It is also possible to measure the number of bugs that present themselves after code is deployed and how many are discovered by the user. Qualitative data is mainly useful in regard to describing the performance of the software itself rather than the performance of the development team, however it can still be used in the latter.

Therefore, an approach to understanding the performance of an individual/team is to therefore use several metrics from both quantitative and qualitative data types and to create a profile for the engineer or team. However, there is some ethical problems that arise from such an approach which I will discuss in a later section.

It is also key to note that there are many non-code and non-quantifiable factors that affect the performance of a team/individual software engineer. Job satisfaction, workplace activity, day-to-day attitude, and the personality of the individual - just to name a few - all play significant roles in how a software engineer performs. It is when these factors are attempted to be measured and quantified so that they can analysed is when ethical concerns are raised – as some of the methods to obtain this data can infringe on privacy. As well as that, who can access this data (if retrieved) is also of concern. I however will discuss these ethical concerns later.

PLATFORMS FOR METRICS

Today there are plenty of services available for companies to track and analyse their software engineers so they can gain insight into their work habits and their performance. The data they provide, such as the metrics described previously, are useful to the management of these companies as it enables them to make more informed decisions on how to structure teams or who to assign tasks to. I will be describing three example platforms that can be utilised that make use of the measurable data provided by the software engineering process.

1. Pluralsight Flow

Pluralsight Flow (formerly GitPrime) is a service that aggregates data from git repositories, ticketing systems, and pull requests and transforms them into easy to understand insights and reports. The service has four sections to it: Code, Review, Collaborate and Upskill. The code section gives data regarding lines of code, commit risk, and code churn. It can also be used to recognise bottlenecks in the project. Review provides a visualisation of a team's code review dynamics and can measure how collaborative a group is. The collaborate section allows for objective measurement of impacts to cycle times and code quality – giving details about who is giving feedback and the percentage of the team that is involved in giving feed back or the likes. Upskill then gives reports on a team's effectiveness by programming language, commit efficiency, project timelines and sprint productivity. It also gives the ability to view a work log – a report detailing the work habits of employees and their contributions. The number of tools provided by Pluralsight is extensive, and gives a very clear overview of the general productivity of a software engineering team.

2. Code Climate

Code Climate is a specialised service that allows the tracking of software engineering projects and their progress, similar to Pluralsight. It comes with two applications: Code Climate Velocity and Code Climate Quality. The first application, Code Climate Velocity, mainly deals with performance analytics of the project – tracking progress, spotting bottlenecks, giving breakdowns of activity, and seeing trends in languages and commits. It provides concise, objective metrics for the project's team so that they can improve their efficiency. Code Climate Quality is mainly concerned with the quality of the code produced in the project – it gives detailed, automated code review for test coverage and gives information of files with high technical debt. The combination of the two applications allows for the engineers within a team to make better informed programming decisions. It differs from Pluralsight in that it used more by the engineers opposed to the managers of the project.

3. Humanyze

The Humanyze platform is a data driven service that analyses collaboration and communication within a workplace. It is not specific to software engineering, but it can be applied to a software engineering workplace. It helps companies drive better business outcomes by informing management and workplace decisions. The company gives a case study of the service in use: a large European Bank had a performance gap between two of it its branches – one of them consistently outperforming the other by 300%. They began to analyse the communication patterns of teams and physical location patterns and segmented the data by compensation and tenure. It was found that the stronger performing branch had significantly more face-to-face interactions throughout the day, as well as that the employees were engaging with each other more than in the underperforming branch. The lack of interaction was then found to be based on the office layout, which lead to the redesign of the office space in the lower performing branch. This increased sales by 11%. This case study gives a good example of how analytical platforms such as Humanyze make a real impact on the performance of companies. It however is also an example of where analytical platforms enter the grey area in terms of ethics.

DISCUSSING THE ETHICS OF MEASURING SOFTWARE ENGINEERING

In this section I will be discussing the ethical concerns of measuring software engineering and I will be mainly giving my own opinion on the matter.

It is my belief that the metrics and methods used for measuring software engineering are becoming increasingly more and more intrusive and unethical. I can observe that the metrics may be useful in a lot of senses and are interesting to observe, but I feel that their application in a professional environment can be and would be exploitative on employees. They are fun to observe and to visualise, but when they are used to determine a person's employment or financial status, they can be used to paint an inaccurate representation of a software engineer. As we have previously discussed, there are many facets to how a software engineer contributes to a project or development process that are not necessarily quantifiable, so it would be unwise to take these values with reverence or using it as sole evidence of a software engineer's performance.

As well as that, reducing a person down to a myriad of statistics and values may have a negative impact on the atmosphere within a work environment. If the metrics are not treated with care and consideration, employees may see each other as competition – this would go directly against the central idea of an engineering team being cooperative. It could also cause management to see individuals within a team merely as only performance values rather than people, an aspect you would not like to have in someone who determines your living conditions.

Furthermore, the amount of data that can be collected on an individual software engineers is immense. It is possible for a company to not only track your work but also track your work, movement and communication patterns while working (see Humanyze case study). Although there are now GDPR guidelines to prevent such intrusive methods of data collections, many software companies have found side steps and have enacted methods that brush very closely to crossing the line. Other industries and disciplines such as medicine and law have strict ethical and regulatory frameworks in place to protect the individuals within the industry, and no such equivalent frameworks exist for the software engineering industry.

It is in my view then that representing an individual as merely a profile of statistics and values is unethical and removes compassion for a software engineer.

CONCLUSION

In this report I have defined what is meant by the software engineering process and we have observed the data and methods that can be used to measure the process. I discussed some examples of the platforms that can be utilised by software engineers today to analyse their performance or a team's performance, and the many advantages and functionality each platform provides. Observing the ethics of software engineering, I have concluded that although it is interesting to see the data that can be collected on individuals, we must approach and handle the information with care and carefully consider whether the information should be collected or not.

REFERENCES

https://www.youtube.com/watch?v=Dp5_1QPLps0

http://www.citeulike.org/group/3370/article/12458067

http://2016.msrconf.org/

http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf

http://s3.amazonaws.com/academia.edu.documents/35963610/2014_American_Behavioral_Scientist-2014-Chen-0002764214556808_networked_worker.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1479122144&Signature=pO6ZkNbSZgbNTqHMd7xyIdMV5iE%3D&response-content-disposition=inline%3B%20filename%3D2014_Do_Networked_Workers_Have_More_Con.pdf

https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

http://patentimages.storage.googleapis.com/pdfs/US20130275187.pdf

https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

https://www.cs.uct.ac.za/mit_notes/software/htmls/ch02s03.html

https://www.devteam.space/blog/how-to-measure-developer-productivity/

https://www.atlassian.com/agile/project-management/metrics

https://www.pluralsight.com/product/flow

https://codeclimate.com/velocity/

https://codeclimate.com/quality/

https://humanyze.com/case-studies-european-bank/

Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

W. Pan, W. Dong, M. Cebrian, T. Kim, J. H. Fowler and A. S. Pentland, "Modeling Dynamical Influence in Human Interaction: Using data to make better inferences about influence within social systems," in IEEE Signal Processing Magazine, vol. 29, no. 2, pp. 77-86, March 2012.

Dittrich, Andrew, Mehmet Hadi Gunes, and Sergiu Dascalu. (2013) "Network Analysis of Software Repositories: Identifying Subject Matter Experts." Complex Networks. Springer Berlin Heidelberg. pp. 187-198.