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

CSU33012: Software Engineering

Measuring Software Engineering Report

Khushboo Jain, 19309169

Contents

1 Introduction

Software engineering refers to the process of creating, designing, developing and maintaining a software systems. The intent of software engineering is to provide a framework for building higher quality software. Software engineering is no royal road of perfection, rather is a road to a great magnitude of improvement. There are several definitions used to describe software engineering and methods in place to cover, what it is but none is perfect. It is fair to say software engineering is a complex process, with an evolving end product to meet the changing demands.

2 Software Engineering Process

Software engineering is a layered process.There are several processes like waterfall, spiral, cleanroom and agile process. One software engineering process is not meant to suit all organizations or workplaces. Each software development cycle is unique with some personalization as every team and deliverable has its own requirements.

The basis for defining a software process can be time consuming but is pretty straightforward :

- define the steps and sub steps of the process, i.e., what people do;
- define the style(s) of enactment, i.e., what order things get done in;
- define the artifacts, i.e, what gets produced

2.1 Measurable Data

"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind. If you cannot measure it, you cannot improve it."

~ Lord Kelvin(physicist,Baron Kelvin)

Software engineering is not an easily quantifiable thing which can measured by using a single metric, it has to be looked from varied angles and measure each aspect differently. Like it is hard to measure productivity and quality of code just looking from number of commits. So, why measure when there is room for error and various interpretations ? If everything is left to subjectivity then the ambiguity rises. Personal concerns arise as to whether one is contributing enough and delivering the required quality or not.

There is a need for quantifiable measures as it leads to a mutual trust and team knows how everyone is working along with what area needs to be looked into. The issue has been the measures in the past which weren't ideal representatives, like lines of code, butt in seat time, measuring business value, bug counts.

Metrics are important for understanding of the health of the software, whether the project is delivering value with consistency. Whether the people involved in the process responsible enough to make progress and adapt to changes quick enough. It is really important to think about the right fundamental questions.

Without an actual measure one will be lost as to what defines as a term "well" or "bad". For one's development it is essential to know what measures as "well" and what constitutes of "well". It is not about only how long it takes to do something but also the quality and the purpose of the content being evaluated.

There is a pool of data present and the infrastructure to collect data exists. There has always been a lack of descriptive enough benchmarks. The bigger issue is how the data is interpreted and for what purpose, as to what question is being looked at and regarding what. The appropriate pool of data needs to be evaluated to make improvements and decisions. It's like the data being collected and evaluated for measuring productivity is completely different from the one for measuring managerial activities. Key Performance Indicators when set clearly will allow to understand how everyone is doing in a team and to set clearer goals.

So, engineering is measurable if we know the purpose or the question "why" behind what is being measured, using the appropriate pool of data and the appropriate set of metrics. It needs to be measured for management and leadership purposes. After determining what and why we are measuring something we need to look at the right set of metrics, Measuring doesn't need to take place by one metric alone. Tool generated data, using the wrong tools and looking at single tool metric in the pipeline can lead to incomplete measure. Aggregating data is necessary. So, using a constellation of metrics will allow to get a clearer picture. Having too many and mismatched metrics without a predetermined objective can lead to confusion and skew results. But what makes a good metric:

In the book Lean Analytics, Croll and Yoskovitz define a "good metric" as something that's

a) comparative,

b) understandable,

c) uses a ratio or a rate, and

d) changes the way you behave.

That last one is interesting, because using metrics invariably changes the way a team behaves. That's why tracking the wrong things is so toxic, and measuring the right things is so important.

A few issues while answering the question "why" and "what data is being collected" are the human errors in the data being used, fairness and misuse of the data. Lack of contextual data when data is collected from automated metrics. Gathering more data using the correct set of metrics which shows impact or improvement based on the metrics will lead to a better engagement from not only developers but management as well. Impact is necessary to look at as softwares are becoming a commodity.

2.2 Development Analysis Platform

There is a large number of metrics out there for myriad interest which might be complementing or conflicting, so several assumptions are made while concluding. Earlier, only the technical issues were looked at or addressed primarily, while other social factors impacting the productivity were not measured really. The metrics mostly worked on the Maslow's hierarchy of needs, in which higher level needs are only addressed when lower-level needs are satisfied, the fact that technology is the dominant preoccupation. Lower-level needs meant the technological factors being the primary/dominant need. Meaning looking at the code base or repositories was deemed enough, but the contextual aspect was missing. So even with automation there is a need to look at the social and contextual factors affecting the quantitative data being collected. Looking at a few way of collecting data:

Personal Software Process provides an empirical support for improving time quality and quality assurance. Developed by Watts Humphrey, it was to apply the SEI's (Software Engineering Institute's) Capital Maturity Model to the software development process of

individuals. In PSP the data is manual entered/logged by the user. PSP is a structured process in which one starts from the base level to the top level and the process is cyclic. The data entered can be broadly categorized in to the 'product' or 'development' phase. At the earlier phase, the data was manually entered on a survey sheet by the developer. Later, there was the use of a tool which had dialog boxes in which the user can record the effort, size and defects. The tool also displays various analysis if requested. In PSP gathering the contextual switching while collecting data is really hard to capture. It leads to personal development and improvement of the developer.

Computational Intelligence Approaches supports software analysis, design and implementation at various levels, as the software complexity and diversity skyrockets. Automated collection for simpler context switching between "product" and "development" processes, as seen in use of Hackystat (mentioned in Beyond the Personal Software Process). CI is a coherent and symbiotic collection of information technologies, namely fuzzy sets, neural networks and evolutionary computing. The automation of data collection leads to less human errors while inputting the data. The automation leads to lack of contextual information or the various secondary factors impacting the productivity or quality. But if further metrics are used to measure that it can be captured as well. The system allows for companies to just aggregate the right data to find the needed information. It allows for companies to improve more efficiently along the pipeline, as it analyzes the profound weakness and strengths quickly. It is essential to make sure to look at the right pool of data and to keep the data secure. As it is sensitive information and needs to be interpreted right.

Types of measures can be broadly categorized in to product and process measures. The former is focused on the product characteristics assessment and predictions. The latter looks at the pipeline of developing the software of a certain quality, time and cost estimations. Process measures can be difficult to measure.

There are a large number of metrics out there to collect data but it all ends up on what is being analyzed . Each metric has its own purpose and the right constellation of metrics along with a thoughtful analysis without incorrect causation and correlation. The metrics work well.

2.3 Algorithmic Approaches

Size oriented metrics often made up the primary metrics, based on looking at the source code like lines of code, complexity by number of operands and operators, function points, errors or defects per line, code churn and time to measure the quality of the software. It was is a highly flawed metric as looking at source code for quality is essential but measuring one's productivity is not accurate as 10 lines of code which could be written in 2 lines, is highly inefficient. Or measuring the time like maybe a developer spent 10 hours writing a piece of code(like 10 lines) which is really crucial while according to the measure that might be really inefficient. Measuring the size using especially the line of code is considered highly inefficient as it forces a change in the developers' behavior which will ultimately affect the quality of software in some way.

Design metrics are a pretty flawed metric but it still measured to see the design or adaptability of a software. There are a number of parameters like coupling between modules, number of modules and effects, cohesion of a program. Again, it is a lot to do with looking at the code base. It allows to find fatal errors and defects. What is considered or defined as a defect is not as simple as it seems, maybe a functionality relies on something which isn't part of the current sprint cycle, so the functionality which isn't being used or tested might be considered a defect. So, at times measuring in agile process development can get more complex. Another measure, cyclomatic complexity is used to indicate the complexity of a program, which will be a design metric. It is more like a special purpose metric. It is measured mainly for reliability and management. Defect metrics make a big part of design metrics.

There are several ways to analyze the data collected from metrics for different purposes mostly automated in the given age. Like maybe using gitprime to collect data about git repositories, or code climate to gather information about the quality of the software. None of the metrics in the broad categories is ideal by itself. Analyzing the data correctly after collection is really important.Each set of data needs to be analyzed separately. While analyzing shall one use the statistical approach or the machine learning techniques. There is some ambiguity left in the conclusion drawn.

Statistical approaches are driven by several standards and initiatives. The focus on statistical approaches has increased for better quality management. Several statistical approaches need to be used to identify and design process changes. One thing to be cautious of while using statistical approaches is not to assume correlation implies causation. If looking at the productivity of a team, which has dropped when a new developer joined 10 days ago don't immediately blame or doubt the new member.

Machine learning techniques can complement the pre-existing techniques. Machine learning has the capability to improve the process of software engineering as it has sound mathematical and logical justification. The major issue to look out will be the mismatch in the technique being used and a software engineering problem. We need to make sure that the contextual content or data is not missing when using a fully automated way along with machine learning techniques to measure.

2.4 Ethical Analysis

Collecting data for organizations is necessary to improve and remain competitive is necessary. But there are several ethical concerns around collecting data and the security issues about who has access, whether the data is secure or not.

Transparency is highly important, there needs to be an open disclosure of what data is being collected and how. Also if the developers and the management could decide on which metrics they think are fair and will be a good representation will be ideal. As humans nobody wants to be tracked unknown.

Data being collected especially after the introduction of General Data Protection Regulation(GDPR) in 2018 can only be done after individual consent. It makes the point in transparency important.

DevSecOps developed by Software Engineering Institute (SEI) proposes that there shall be data security at each step of the pipeline to make sure that the data collected is secure and not misused. It also enables automated security checks at each steps, by integrating security controls and processes. It is a cyclic process, and should be implemented iteratively.

I personally believe the data needs to be collected but there needs to be a secure collection. Also, the data and measures shall be used for the well being and improvement

of an individual and the organization rather than for accusation purposes. Also, contextual and various social and mental factors need to be taken into analyzing the data. Rather than just purely looking at numbers alone.

3 Conclusion

This report looks at the importance of measuring software engineering for easier development and qualitative purposes of not only the product but also the developer. It can be concluded that the process of measuring software engineering is a complex and convoluted subject with loads of controversial and conflicting interests as well. If we know what we are measuring and why along with secure measures on each step. Then it will be easier to increase the efficiency and productivity as a whole. It is more opinion based rather than looking at just hard facts. The aim was to provide a perspective on what measuring software engineering is, how it is done and whether it's fair to or not.

4 References

Pressman, Roger S. Software engineering: a practitioner's approach. Palgrave macmillan, 2005.

JOUR, Brooks, Jr, Frederick. No Silver Bullet Essence and Accidents of Software Engineering. IEEE Computer. 2009.

"Software Engineering Processes - Users.csc.calpoly.edu." Accessed December 31, 2021. http:/users.csc.calpoly.edu/~gfisher/classes/307/textbook/2.pdf.

Templeton, Andrew. "Why Kpis Matter for Software Engineering." Pluralsight, December 16, 2019. https://www.pluralsight.com/blog/teams/why-kpis-matter-for-software-engineering.

Forsgren, Nicole, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. "The Space of Developer Productivity." *Communications of the ACM* 64, no. 6 (2021): 46–53. https://doi.org/10.1145/3453928.

"Se-Radio Episode 317: Travis Kimmel on Measuring Software Engineering Productivity." Software Engineering Radio. Accessed December 31, 2021. https://www.se-radio.net/2018/02/se-radio-episode-317-travis-kimmel-on-measuring-software-engineering-productivity/.

Forsgren, Nicole, and Mik Kersten. "DevOps Metrics." *Communications of the ACM* 61, no. 4 (2018): 44–48. https://doi.org/10.1145/3159169.

Croll, Alistair, and Benjamin Yoskovitz. *Lean Analytics*. Logroño: Universidad Internacional de La Rioja, S.A. (UNIR), 2014.

Bellini, Carlo & Pereira, Rita & Becker, João. (2008). Measurement in software engineering: From the roadmap to the crossroads. International Journal of Software Engineering and Knowledge Engineering. 18. 37-64. 10.1142/S021819400800357X.

John M. Roche. 1994. Software metrics and measurement principles. SIGSOFT Softw. Eng. Notes 19, 1 (Jan. 1994), 77–85. DOI:https://doi.org/10.1145/181610.181625

Humphrey, Watts S. *The Personal Software Process (sm)(PSP (sm))*. Vol. 11. Carnegie Mellon University, Software Engineering Institute, 2000.

Johnson, P.M., Hongbing Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, Shenyan Zhen, and W.E.J. Doane. "Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined." *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003. https://doi.org/10.1109/icse.2003.1201249.

Witold Pedrycz. 2002. Computational intelligence as an emerging paradigm of software engineering. In Proceedings of the 14th international conference on Software engineering and knowledge engineering (SEKE '02). Association for Computing Machinery, New York, NY, USA, 7–14. DOI:https://doi.org/10.1145/568760.568763

K. Akingbehin and B. Maxim, "A Three-Layer Model for Software Engineering Metrics," Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06), 2006, pp. 17-20, doi: 10.1109/SNPD-SAWN.2006.12.

Roche, John M. "Software Metrics and Measurement Principles." *ACM SIGSOFT Software Engineering Notes* 19, no. 1 (1994): 77–85. https://doi.org/10.1145/181610.181625.

Sillitti, Janes, Succi and Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," 2003 Proceedings 29th Euromicro Conference, 2003, pp. 336-342, doi: 10.1109/EURMIC.2003.1231611.

Card, David. (2004). Statistical techniques for software engineering practice. 722- 723. 10.1109/ICSE.2004.1317505.

Zhang, Du & Tsai, J.J.P.. (2002). Machine learning and software engineering. Software Quality Journal - SQJ. 11. 22 - 29. 10.1109/TAI.2002.1180784.

Atlassian. "DevSecOps Tools." Atlassian. Accessed January 3, 2022. https://www.atlassian.com/devops/devops-tools/devsecops-tools.

https://podcasts.apple.com/ie/podcast/software-engineering-institute-sei-podcast-series/id566573552?i=1000538699839

https://podcasts.apple.com/ie/podcast/software-engineering-unlocked/id1477527378?i=1000544238371

https://www.se-radio.net/2018/02/se-radio-episode-317-travis-kimmel-on-measuring-software-engineering-productivity/