Measuring Engineering

A report on the current state and advancements in the measurement of workplace productivity for Software Engineers

Daniel Weng Yang Fong

CS3012

CONTENTS

INTRODUCTION

The Software Engineering industry has always struggled with means to identify productivity and quality of the code produced by developers, as such there is a lot of controversy surrounding the topic of software metrics and how their use might aid or impede the development and innovation of better software.

So what are software metrics?

‘A software metric is a measure of software characteristics which are quantifiable or countable.’ - (4)

In recent years many software developers have moved away from traditional methods to quantify their productivity as it has proved to be ineffective. Some claim that metrics are useful but are being interpreted in the wrong way, and that they should be interpreted in the context of the ‘success’ of the software being developed (2).

The reason for this shift is based on the unique nature of software engineering, whereby analysing and quantifying ones work without considering the value or ingenuity of their ideas results in an incomplete picture of the software engineers contribution to the project.

MEASURING SOFTWARE ENGINEERING PRODUCTIVITY

Why Measure?

As with many other industries the benefits of measuring and determining factors related to employee productivity are being increasingly valued and analysed. This is especially important in software engineering due to the intertwining nature of a software application and how issues caused by poor contributions of a certain subset of the team developing the software could go unidentified until later at release.

Increasing productivity would also obviously lead to a more efficient use of resources within the workplace allowing for better value to employers as well as better and increasing opportunities for successful developers.

Difficulties in Measurement

The issue with measuring ones work in Software engineering is fairly simple to understand. For example when assigning the same task to two employees, one might labour hard and write many lines of code but take a week to complete the task, the other may attempt to 'hack' the task and find a quick way to do so whilst taking a couple hours. In this case it's easy to say the employee who 'hacked' the task deserves the merit as he was so much more efficient, however in the case of coding large software applications finding a 'hack' to an issue may then cause far more issues down the pipeline. A balance needs to be found between being efficient yet reckless or thorough and slow, this is quite subjective, and as some might call it, an art.

Measurement Methods

Although there are numerous metrics being used, these would seem to be the most used ones:

Time Logged

As a mode of measuring work this is probably the simplest and most used across various industries. This simply involves measuring the amount of time the employee puts towards his job and can be used to calculate cost effectiveness in terms of hours taken to complete a task assigned.

Lines of Code

This is a logical measure to use in software engineering yet a deeply flawed one in terms of measuring productivity. A software developer may simply space their code more or over complicate the code and thus generate more lines of code whilst deteriorating the quality of the code.

Code Coverage

Code Coverage means how much of the developers code actually comes into use. In Layman terms it tells managers/employers how much of the developers code is essentially useless to the project. This is quite a useful measure, however it incurs additional cost due to testing and can sometimes be flawed.

Commits

Software Version Control Systems are almost universally used in Software Development teams as a means to share code and track who contributed what. Measuring the commits of a developer as well as the frequency of them can be used to understand how much a developer has contributed to the project as well as his frequency to make small adjustments or improvements.

Bugs

Ultimately a measure of counter-productivity, this means measuring a developer based on how many issues their code causes to the overall project. Obviously the more the worse the quality of their code.

Code Churn

Code Churn is a slightly more advanced metric which combines Commits and Lines of code to measure how much of a certain developers code has been added to the codebase, and how much of it has been rewritten as a series of additions/deletions during the project. This metric shows the robustness of a developers code and provides an insight into the quality of the developers code, as a developer who seems to be constantly rewriting his own code is unlikely to be producing high quality code efficiently.

ALGORITHMIC APPROACHES

Algorithmic approaches tend to take a more holistic approach to measuring software engineering team productivity by combining metrics and assessing some qualitative factors.

Halstead Complexity Measure

As part of his treatise to establishing software engineering as an 'empirical science', Maurice Halstead developed the following(appearing in the image below(6)) metrics to measure and combine various aspects of the software development process and extrapolate estimations as to the time taken, bugs developed, lines written and etc. metrics. Companies have implemented and used these models whilst comparing them to their own employees to better understand the utility of their employees.

Measure	Symbol	Formula
Program length	N	N= N1 + N2
Program vocabulary	n	n= n1 + n2
Volume	V	$V= N * \log_2 (n)$
Effort	E	E= V/2*n2
Halstead Program Length	H	$H = n1*\log_2(n1) + n2*\log_2(n2)$

Agile

Agile is a Software Engineering philosophy which aims to combine elements of planning and teamwork within software engineer teams as well as feedback from users and testers to develop a fluid and efficient pipeline to develop high quality code. The Agile Manifesto is based upon these 12 principles(7):

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential

11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

This adaptive development method has been used in some form for numerous large projects throughout the years with examples such Lisp and Java.

ETHICS CONCERNS

Privacy Boundaries

One ethical issue to consider is how the collection of certain data might intrude on an employee's right to privacy. For example some technologies are being used to track physical attributes of the employee at work to better understand how his conditions may be affecting his productivity, acting as a ‘big brother’ presence to observe the employee. Other methods of tracking could intrude on an employee’s privacy on his work computer, for example tracking his work habits and how he may have his own manner of approaching a certain task. These issues are continuously evolving not only in software engineering but also in other industries as the move towards a more interconnected future where more and more data is handled digitally.

Use and Implementation of Productivity data

The issue with software engineering metrics as we have discussed is that there are flaws within them, and that this may cause an incomplete picture to be used regarding an employee’s productivity and quality at work. This leads to the danger of these metrics being used by managers to handle employees and make serious mistakes (such as firing a member of the team) due to misleading information.

CONCLUSION

The risks of overusing software metrics is quite apparent due to the many flaws within the measurement of them and the lack of a more qualitative judgement on the code.

Another issue which I have not touched in this report with using metrics is that they are also so far ineffective in terms of understanding how an employee might contribute in a more social manner to the flow of the team. Methods to measure social workplace interaction are likely to be explored further in the future, however care is needed as to not cause ethical issues regarding employee privacy.

I believe that moving forward into the future systems to recognise not only the bare statistical effectiveness of a software developer, but also the creative design element shall become possible with the implementation of Machine Learning Systems, these systems should also become able to recognise the social impact of team members on each other and represent a holistic overview of how ‘good’ a developers work is.

REFERENCES

1. https://techbeacon.com/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals
2. https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams
3. The Journal of Systems and Software 47 (1999) 149±157: “Software metrics: successes, failures and new directions” by Norman E. Fenton *, Martin Neil
4. https://stackify.com/track-software-metrics/
5. https://en.wikipedia.org/wiki/Halstead_complexity_measures
6. https://www.researchgate.net/profile/Selcuk_Sevgen/publication/319481865/figure/tbl1/AS:631652569063467@1527609116687/Halstead-Complexity-Metrics.png
7. https://en.wikipedia.org/wiki/Agile_software_development#The_Agile_Manifesto