Measuring Software Engineering Report

(By Sean Murphy, #19335002, Computer Science and Business)

Introduction

The software engineering process can be measured in several different ways in terms of measurable data, quality and completeness of work and performance of employees etc. Productivity and quality of work of a company's employees is of serious priority as of late. Often managers or even external companies or technologies are contracted to ensure that employee performance and productivity remains at a high level, constantly rising and never plateaus or dips. So, we ask ourselves, *which is better, human measuring or AI?* It really is a heated debate. Recently, it's become increasingly common for software to be used to measure productivity and performance of employees. The software is installed on the employees' machine or in some cases on their person in the form of a bracelet or necklace. This way of collecting data causes major ethical concerns over employee welfare as they are constantly being monitored around the clock unlike a human can. Over the years, numerous studies have taken place to measure productivity of employees and

that of a software engineer and time and time again the result has boiled down to the equation that:

Productivity = Output ÷ Input

This is a simple equation to understand but the main take from it is *how do we measure input and output?* It is a very difficult question that people have been trying to answer for many years. Is it how quickly a developer completes a task that they are given or how many lines of code they can write per hour or per day? But what if the code they have written is full of bugs and errors and does not fit the specification of the task they were assigned. Well then is the software engineer really being productive? These are some of the questions I will discuss and hopefully be able to conclude on in the following report.

Data Measuring and Gathering Standards

As previously stated, the ability for one to measure productivity is quite difficult. There are endless ways to do it, but every way has both its pros and cons. We said that one way to measure is by the number of lines of code a developer has written. The disadvantage of this is that the code may be full of bugs and not optimized. A more efficient method for developers is to practice Test Driven Development (TDD) and Unit Testing.

Although, this method of development is certainly more time consuming, the number of defects in the code is massively reduced. In a study by Boby George and Laurie Williams it was found that TDD can increase development time by up to 35% but this increase in development time can decrease defects in the code by up to 90%. 78% of developers said that TDD improves their overall productivity as a developer and 92% said that TDD allows them to write higher quality code with less bugs and errors (George & Williams).

In a lot of cases TDD is not cost friendly enough for businesses to take this method on board with their developers. Code complexity is often much greater and could be remodeled into a simpler program with the same function for the piece of technology that the company is working on. The middle ground businesses find is to have levels and standards that all code written must adhere to.

Agile development involves development, which is much less likely to cause errors, it increases performance and enhances both the productivity and knowledge of developers. It involves developers always working in pairs on the same code where they can share ideas with one another and give their opinion on how their partner can improve. Code is always tested before it is pushed to a repository on a common codebase that is shared between everyone on the team to allow everyone to view everyone's work. This development

method is easier to measure as although team members are expected be competitive with one another to outperform other pairs outperform other pairs within the team. Pair members are reliant on one another to be productive so that their tasks are completed in good time and to a high standard to impress managers and hopefully earn bonuses and gain better status within the company.

As I spoke about previously, bugs are very prevalent in development. They are pretty much impossible to avoid, and they take an indefinite amount of time to fix as the scope is unknown and a lot of painstaking work is needed to fix the bugs. Due to this, bugs are not suitable for estimation or data measurement.

So, although it easy for developers to be measured on how many lines of code they write or how many commits they make to a given repository. I think it is more suitable for developers to take a more conservative approach when writing code, one in which they are less likely to make mistakes or produce bugs and errors such as TDD or Agile development. This is due to the fact the time to fix bugs in programs is indefinite meaning that, realistically, while bugs exist, we cannot accurately measure the productivity of a developer. Speed and volume of work is important, but I believe that businesses must prioritize correctness of work when measuring the productivity of their software engineers.

Platforms Available

Today, thousands of platforms exist for companies to measure the productivity of their employees. There are several applications that allow managers to track the productivity and time spent using company computers such as Desktime and WorkIQ. These applications categorize applications into productive and unproductive with social media apps like Facebook being labeled unproductive while Microsoft 365 applications like Word and Excel would be considered productive. These applications have been met with lots of scrutiny by some employees as some employers have begun paying employees for their actual hours worked rather than the time that they spend in the office. Platforms more suited for employers who want to measure the productivity of their employed developers are GitPrime, TestRail and Jira.

Gitprime refines data from git into easily readable metrics and graphs for managers and engineers alike to optimize coding practice within the company. It will tell you a lot about how a developer is performing commits and pull requests to a project. These logs offer a quick view for managers on the productivity of their developers. Git have stated that this service has correlations with increases and decreases of developer

productivity. If new integration or deployment tools are added, then overall productivity is increased. This service does not just refine lines of code but all technicalities in terms of creating projects and getting company products onto the market. The reason productivity is measure is to hopefully be able to increase it in the future, therefore GitPrime is useful as it provides data on when developers were least productive and helps mangers to try to understand why this is happening.

Other tools exist that are plug-ins for IDE's and the repositories that store a project base. These can be useful for the purpose of quality of code and assurance. An example tool is TestRail, it integrates with GitHub to hold source code and Jira which hosts relevant and Agile development boards. TestRail allows teams to monitor testing metrics, provides detailed user-friendly interfaces that allow managers to monitor testing quality, code coverage and test results relating to the task assigned to an individual developer. It does not gather metrics such as lines of code written. These metrics are useful for managers as they can significantly reduce the number of bugs in each program which increases productivity as there is less time needed to fix bugs and issues that may arise in the future (TestRail – Test Case Management).

Jira is a piece of software especially used within organizations that practice Agile development. It allows

project managers to map and design ‘sprint’ tasks and assign these tasks to developers using a scrum board. The tasks can then be tracked by the manager until they are completed by the developers. It gives developers within a team full transparency into the task at hand and allows them to bring maximum code output in a minimum amount of time. It then offers metrics and graphs into the performance of the developer throughout the life cycle of the given ‘sprint’ task. Jira is completely integrated with GitHub so code can be reviewed almost instantly, and feedback can be reported which increase overall productivity and reduces the time for a product to transition from development to production (Jira – Software Features).

Approaches available

In my introduction I stated that productivity is often measured as output divided by input, but this can be rather vague as we also must account for the software developer themselves and include factors withing their life which may cause this equation to change. Historically software engineering has been measured as:

Productivity = ESLOC / PM

Where, ESLOC = Effective source lines of code
PM = person month

ESLOC must be greater than or equal to the number or source lines created or changed by the developer. Measuring ESLOC is straight forward when all the lines of code are new. Other factors must be considered when modifications are made such as that thorough understanding of the code the code to be modified, along with knowledge of the associated systems architecture. Reverse engineering of code must be done if existing documentation is insufficient. Modifications can't break any existing interfaces and code written in a new language must be compiled with new compilers. To account for the above complexities an adaptation adjustment factor (AAF) should be applied to modified code:

AAF = 0.4Fdes + 0.3Fimp + 0.3Ftest

Where

Fdes = % of the reused software requiring redesign and reverse engineering,
Fimp = % of the reused software that has to be physically modified (technically called "recoded"),
Ftest = % of the reused software requiring regression testing.

Using the AAF, ESLOC can be re-written as:

ESLOC = Snew + Smod + Sreused*AAF

Where

Snew = new lines of code,
Smod modified lines of code,
Sreused = reused lines of code.

Although ELSOC has now been defined, it is not easy to measure ELSOC for individual developers within an organization. So, ELSOC is calculated by getting statistics off a company's existing code repository and then tools must be used to get the final value for ELSOC (Chatterjee, D.*)*.

Another approach available is the Halstead complexity measure to measure the productivity of a developer producing a program. It was first established by Maurice Howard Halstead in 1977 as part of his research into the empirical science of software development.
In a software problem, let:

N1 = number of distinct operators
N2 = number of distinct operands
X1 = total number of operators
X2 = total number of operands

We can derive several measures using these numbers and when we combine these measures, we can analyze the problem at hand from mathematical approach.

Using the above numbers, let:

Program vocabulary: N = N1 + N2
Program length: X = X1 +X2
Calculated program length: X’ = N1 log2(N1) + N2 log2(N2)
Volume: V = X * log2(N)
Difficulty: D = N1/2 * X2/N2
Effort: E = D * V

Difficulty refers the difficulty for a programmer to write or understand the program. This accounts for a developer writing and performing a code review.
Effort can be translated into actual coding time by using the following relation:

Time to program: T = E/18 seconds

The time to test a program can also be measured using the equation:

Time to test: U = E/K

K is an arbitrary number that is defaulted to 18 but can be changed based on a developer’s conditions.

Halstead attempted to equate how to measure bugs in a solution:

Number of bugs: $B = E^{2/3}/3000$
This approach was developed all the way back in 1977 and can still be used somewhat today. Some of the equations do not work quite as well for modern programming languages which can be resolved much quicker than they could some 40 years ago but is still used as a benchmark for analyzing metrics within a project (IBM Knowledge Centre, n.d.).

Ethics

Ethical concerns are very prevalent in the measurement of software engineering. Developers must be treated equally in how their work is measured and analytics that are gained are all completely subjective. If different measurement techniques are used across different departments within a business, the accuracy of the measurement of each employee can differ greatly which can be both unfair and unethical.

There are many advantages and disadvantages of both machine measuring and human measuring of software engineering. In terms of using a machine or technology to measure the productivity of an engineer, a machine or technology shows no bias or prejudice whatsoever towards the engineer as the employee is just a number or name and are measuring them on the work that they

get done. A disadvantage of this is that a machine or technology cannot measure the engineer's attitude or the atmosphere that they create within a team which could negatively outweigh the amount of work that they get done. When looking at the ethics of human measuring it is the exact opposite to that of using a machine or technology to measure in that a manager could be biased or prejudice towards an employee based on their previous dealings with the engineer and may in turn not fairly measure the amount of work that they get done but on the other hand they can also see how the engineer interacts with his or her colleagues and the atmosphere that they bring to the office. It seems that a combination of both technological and human measurement may be the most effective way to ethically measure software engineering in a way that work ethic, personality, and attitude to work are all measured.

It is hard for companies to integrate ethical measures in a way which portrays the company in a good way to its shareholders whilst protecting the employees' rights to privacy and confidentiality. Companies' codes of conduct are built around good ethics along with behavioral rules, dress codes, accountability, and commitment to their work along with integrity for themselves and the company but this is hard for an employee if their rights to privacy and confidentiality are invaded through the measurement of how hard and efficiently, they work (McQuerrey, n.d.).

Conclusion

The ability to gather data from an employee by using methods like their code added, system knowledge, ability to track bugs, contributions which allows managers to create an overall view of the overall productivity of the employee through graphs, tables etc. However, these methods aren't very accurate and should be down to a manager to judge the performance of the employees and evaluate their productivity by this and their contributions to the outlined work.

The focus of corporations today seems to be one that does not promote individuality for their employees, they only seem to want people who will contribute in the right way and have the correct attitude that they are looking for. Despite this, businesses are at a huge advantage when they have identified workers who are the best at performing each task which will reduce overall errors and in turn make any errors easier to fix.

The practice of measuring employee's productivity clearly has many pros and cons as outlined above. It is hard to decipher whether the pros outweigh the cons and its evident that many people have different outlooks on the practice. It is unclear what the future will hold and if we will be constantly measured and compared to our peers. It is question that only time will be able to answer.

Bibliography

George, B. & Williams, L. *An Initial Investigation of Test-Driven Development in Industry*.

S. Malathi, & Seshadri, Dr.Sridhar. (2011). *An algorithmic approach for the implementation of analogy-X for software cost estimation.*

https://www.gurock.com/testrail – TestRail - *Test Case Management.*

https://www.atlassian.com/software/jira/features – Jira - *Software Features.*

Chatterjee, D. *Measuring Software Team Productivity* – http://scet.berkeley.edu/wp-content/uploads/Report-Measuring-SW-team-productivity.pdf

IBM Knowledge Centre – Halstead.

McQuerrey, L., n.d. http://work.chron.com/ethics-performance-evaluations-20998.html. [Online]