Measuring Software Engineering

Sanil Gupta

21/11/2020

Table of Contents

Introduction

In the modern day, it has become more and more important for software engineering to be measured. In the past, companies have tended not to put much effort into analysing the efficiency and accuracy of projects and product development processes. However, it soon became obvious that this was an absolute necessity in order to compete with the most influential companies. The issue with this is, ever since the idea of software engineering was introduced, experts have been discussing and arguing over the best and most accurate ways to actually measure and assess the software engineering process. To this day, there is still no correct answer to this question.

Measurement is defined as 'a manifestation of the size, quantity, amount or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard'. Measurement is so necessary in a business because it allows the management to make decisions based on statistics and calculations rather than just observation. The measurement of software improves estimations of future qualities of the product or service, upgrades the product or service, and regulates the state of the project with respect to the budget and the schedule.

What Can Be Measured?

Measuring software developer output quantitively tends to be considered both misleading and irrelevant for many different reasons. Below I will talk about some of the most commonly mentioned statistics that are available to managers and employers in order to assess the productivity of a software developer and explain why using these statistics may not be a good idea.

Lines of Code

Lines of code is one of the most talked about quantitative metrics, which goes by the idea that the more lines of code written the better, as well as meaning that developers that write more lines of code are more productive. I do believe that the number of lines of code can reflect developer performance in some ways. It has the ability estimate the amount of time and effort that is required to finish a program. However I do believe this metric has its flaws, an obvious one being that it encourages code redundancy which leads to low quality

software. Developers may add unnecessary lines of code to their program in order to appear more productive, even though this actually does quite the opposite. On top of that if a developer adds to another program and makes it twice as big in terms of lines of code, the code may introduce bugs into the system having a negative effect on the program.

Commit Frequency

Commit frequency is another metric commonly used perhaps due to the easy of extracting the information. If the employees of the company all publish their work on an online platform such as GitHub, data such as the number of commits in a program is available to the employer. Commit frequency is typically used to reward teams with a high frequency. In my opinion this metric is more accurate than using lines of code, and I have many reasons for this. Mainly, it is not susceptible to to redundant code which is one of the main problems of using lines of code as a metric. It encourages smaller and frequent commits which can result in greater collaboration and continuous delivery. You can often identify a team that is struggling with their project if they have not committed in a relatively long time.

However, I still believe there are flaws in using commit frequency as a metric and these flaws are too large to justify using the metric as a tool of measurement. The main problem is the same that is of using lines of code as a metric – commits can be poor or redundant, and is not necessarily good code. Furthermore, two developers could write identical code but if one developer commits more often than the other they will be seen as being more productive even though this is not the case.

Pull Request Count

A more recent way of measuring productivity in software engineering is counting pull requests. This measuring process is more known for analysing the process rather than the individual programmer. It can be a very useful measurement tool for managers in order to understand the dynamics of a team and make sure that the team is collaborating in a healthy manner. However in the grand scheme of things using pull requesting to measure software development I not very useful as you can not assess individuals. It would be unfair to use the amount of pull request of a programmer to assess that individual as a pull request count does not take into account the size, or difficulty of the pull request.

Using the pull request count as a metric encourages individuals to create pull requests unnecessarily in order to look more efficient. This creates unwelcome overheads across the team.

Platforms Available

Due to the fact that there are so many metrics that need to be taken into consideration in order to assess the performance of a software engineer, there are now many platforms that were developed for managing developers that are available to the public in order to perform analysis on a project. Since GitPrime, one of the most popular performance measuring software, was released in 2015, there have been many platforms that have followed suit such as GHTorrent, Pivot, Velocity, VizzAnalyser, GitClear and Code Time.

GitPrime

GitPrime is an organisational tool which automatically tracks team process, highlights development work patterns, and identifies areas to five feedback. GitPrime documents and examines many different metrics including an individual's metrics such as defect rate, amount of technical debt, and details on time and number of individual commits. GitPrime mines git repositories and processes this raw data to produce analysis on developer efficiency. This software helps managers understand which programmers have low or high churn rates, the time it takes a developer to provide 100 lines of code, and whether the group meetings have been productive or not. According to a study GitPrime conducted with their clients, they all had an increase of at least 5% in productivity with some clients having up to 100% improvements. Some of these clients include Tesla, Storyblocks and 128 Technology.

In my opinion, GitPrime is a very useful software for both managers and software teams to use in order to analyse the software efficiency. It lets the managers know which individuals may require help and take action soon, rather than slowing down the teams progress by allowing the individual to stay stuck on a certain problem. It helps team set development goals which I think is a great way to maintain progress and make sure the team's standards do not drop in terms of efficiency. However, I still do not believe that GitPrime is the perfect way to measure software engineering. As I have mentioned, this organisational tool uses metrics such as number of lines of code, commits per day, and impact scores which I do not believe is an accurate way to measure efficiency of a programmer.

Personal Software Process (PSP)

The Personal Software Process is a structured software development process that is designed to help individual software engineers better understand and improve their performance. The PSP was created by Watts Humphrey 'to apply the underlying principles of the Software Engineering Institute's Capability Maturity Model to the software development practices of a single developer'. Usually the PSP analyses and continuously monitors manual data collection however there are many different variations available. Some of the things the PSP is said to do is reduce the amount of defects in a project, improve predictions and planning skill, and managing the quality of projects.

Some metrics that are used in the Personal Software Process are logged using the 'PROBE' method (proxy-based estimating), an estimation method introduced by Humphrey which uses the estimated size of a system to find an estimated time of development based on past projects. This PROPE method works under the assumption that the project a developer is working on is similar to ones that they have worked on in the past, therefore data can be used from these past projects in order to make estimations about the current and future projects.

I do believe there are benefits of using the Personal Software Process such as the fact that it improves planning skills, manages the standards of projects, reduces the number of faults in a project, and can higher software productivity in generally. Nevertheless, I once again believe that this approach is not fully effective or reliable. Due to the manual nature of the data collection required, not only is the process very time consuming, it is not very flexible and can produce huge overheads for the developer.

Velocity

In 2018, software engineering company Code Climate released Velocity, a now well known performance metrics software that focuses more on the project rather than the individual. When there is a problem in the program, Velocity is used to identify where the problem is within the code. Velocity provides additional metrics around impact, pushes per day, and coding days per week to help monitor developer activity. Velocity is similar to GitPrime however I believe it is slightly more flexible. It allows managers to filter out the metrics that they do not feel are beneficial, and they are also given the ability to create their own reports with sections that they find most useful.

Velocity is another performance metrics software that I find can be useful but can also be misleading. Velocity is simply an output and may be able to help business make estimations however there is no way of the software proving itself to be accurate.

Pinpoint

Pinpoint is an advanced analytics platform for software engineering which applies machine learning to git repositories in order to give managers an idea of cost, efficiency, results and performance. Pinpoint focuses more on system data rather than the code itself. Pinpoint uses metrics such as backlog change, throughput, cycle time, workload balance and defect ratio, rather than straightforward metrics like lines of code. Pinpoint's AI uses a team's work to learn how the team operates during a project, and it modifies itself to suit different team.

In my opinion, the metrics Pinpoint uses are much more relevant metrics compared to those that the software's I mentioned above use. These metrics give an intuitive cooperative set of measures to convey the performance and efficiency of a software development team. Out of all the approaches to measuring software engineering that I have researched and mentioned, Pinpoint is both the most accurate and the most useful performance measuring tool.

Algorithmic Approaches

For many years now, researchers have been trying to find the best ways to come up with algorithms and models to analyse collected data. Some of the most popular models are The Constructive Cost Model (COCOMO), the Bayesian Belief Nets (BBN), and the Function Point Analysis.

Constructive Cost Model (COCOMO)

Published in 1981 by Barry Boehm, The Constructive Cost Model is a procedural software cost estimation model, a regression model based on the number of lines of code from previous projects. It is a model made to assess the effort, cost and schedule of software development. COCOMO consists of three levels of evaluation with increasing accuracy and detail. Any of these three forms can be used, as each is useful for different complexities of

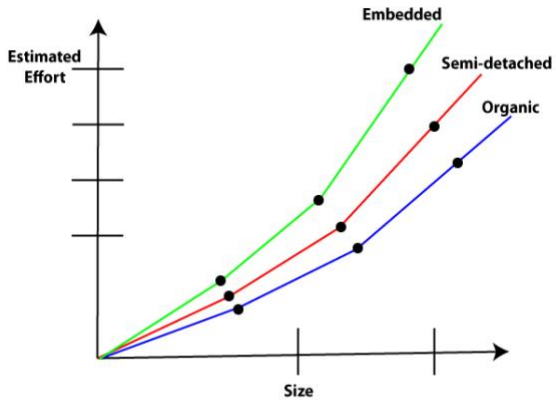
projects. Different formulas exist, and it has been adapted and altered multiple times

The first level is the Basic COCOMO Model, which would usually be used for quick and rough calculation of software costs. In this level, Kilo Lines of Code is the only software metric used, which is put into the formula with the coefficients corresponding to the project type, to calculate the effort, development time and required people.

The next level is the Intermediate COCOMO Model, and this form takes cost drivers into account. As well as evaluating Kilo Lines of Code, the Intermediate COCOMO Model also evaluates 15 other factors, with each factors value being multiplied to calculate the Effort Adjustment Factor (EAF). The EAF, along with the Kilo Lines of Code will then be used in the regression formula to calculate the result.

The last level is the Detailed COCOMO Model, and this model takes into account both the cost drivers and the influence of individual project phases. The development process is divided into 6 phases, where for each phase the effort is calculated. They ae then summed up to give the result. This level produces the most accurate result.

The model is based off two key parameters, effort and schedule. Effort is the amount of labour required to complete a task, and is measured in person-months unit. Schedule is the amount of time that would be required to actually complete the job. Schedule is proportionate to effort and is measured in time, for example weeks.



Effort versus product size

As we see in the above diagram, there are 3 types of systems that were defined by Boehm when he came up with this model: Organic, semi-detached and embedded. Here are the following definitions of each given by Boehm:

Organic – 'A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.'

Semi-detached – 'A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.'

Embedded – 'A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.'

Basic COCOMO Model: Formula

$E = a_b \, KLOC^{\,b_b}$
$D = c_b \, E^{\,d_b}$
$P = E/D$

Where E is the effort applied in person months, D is the development time in chronological months, KLOC is the estimated amount of delivered lines of code for the project (expressed in thousands), and P is the amount of people required.

Ethical Concerns

As almost every major company in the modern day is using performance measuring software, the ethical boundaries around these companies gathering and analysing data on employees is becoming increasingly scrutinised. Ever

since performance measuring software has been used by companies, there have been reports of employees being against the software for many reasons including data transparency, inaccuracy of algorithms, and the relationships between workers. Our data is constantly being collected and studied in order to sell goods and services to us, however a line must be drawn between observing our data and the breaching of our privacy.

When some particular measuring software is being used on workers, it can make them feel uncomfortable which in my opinion should never be the case. It has been reported that the process of some of these approaches can feel intrusive in the way that it retrieves data.
Due to the fact that software engineers are so in demand by companies in the modern day, companies know that it is essential to make workers feel comfortable in their working conditions, otherwise they will work for the companies competitors instead.

The data being collected on employees in the workspace is often used in order to measure the employees performance and productivity, suggesting that a software can tell if an employee is not being productive enough. Many people feel like it is unfair to be judged by a computer rather than person. Although computers are able to assess certain statistics such as lines of code or amount of commits, they are many relevant things a computer is not capable of doing. A computer cannot assess some metrics such as how much effort a worker is putting in, or how helpful a worker has been to other members of the company.

The 1950 European Convention on Human Rights from 1950 states that "Everyone has the right to respect for his private and family life, his home and his correspondence." The EU is responsible for protecting this right and so in 1995 the European Data Protection Directive was passed, establishing minimum data privacy and security standards. However soon after this, technology became greater and more important in our lives, and so the increase in data collection was monumental. The Directive from 1995 was not enough anymore, and so it was agreed by the EU that modern laws had to be introduced to ensure the protection of people's data.

The General Data Protection Regulation was put into place in 2016, and in order to regulate data processing and protect the publics personal data. Personal data is defined by the EU as 'any information that relates to an individual who can be directly or indirectly identified. Names and email addresses are obviously personal data. Location information, ethnicity, gender,

biometric data, religious beliefs, web cookies, and political opinions can also be personal data. Pseudonymous data can also fall under the definition if it's relatively easy to ID someone from it'. There penalties for breaking the GDPR are huge, with the fine being up to €20 million or 4% of global revenue (whichever is higher).This prevents companies collecting 'unreasonable' information on the employee without the employees permission. All organisations were required to be compliant by 2018.

Conclusion

Although countless amount of researchers and engineers have done their best to come up with many different ways to measure software engineering, I believe there will always be flaws in the result. This does not mean that I do not agree that the collection of a workers data can be very useful to both the manager and the worker themselves as this is not the case. I think do think that some performance metric software is can be very useful to managers as it can give them a good idea of which workers require assistance. It is undeniable that if a worker has not written any code for a long time they are either struggling with an problem or there is another issue and without the software, managers may never have been aware of this. Software can let the managers know how much time a group has put into a project and how efficient their group meetings have been. These statistics can also help individuals as it can let them know aspects of their programming skills that can be improved. However, many of these statistics can be taken with a pinch of salt as there is no way of some of them being accurate. How can a computer truly know how much effort an individual is putting into a project? What makes a piece of code better than another piece of code? Statistics can help a software engineer improve their coding abilities, and can sometimes compare two different projects. However, I do not believe there is any way of simply 'measuring software engineering'.

Bibliography

(n.d.). Retrieved from https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

(n.d.). Retrieved from http://patentimages.storage.googleapis.com/pdfs/US20130275187.pd

(n.d.). Retrieved from http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf

A. Sillitti, A. J. (2003). *Collecting, integrating and analyzing software metrics and personal software process data.*

Di Penta, M. (2012). *Mining developers' communication to assess software quality: Promises, challenges, perils. In Emerging Trends in Software Metrics (WETSoM),.*

EU. (n.d.). Retrieved from https://gdpr.eu/what-is-gdpr/#:~:text=The%20General%20Data%20Protection%20Regulation,to%20people%20in%20the%20EU.

Fenton, N. E. (n.d.). *Software metrics: successes, failures and new directions.*

Geekforgeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/software-engineering-cocomo-model/

Geeks for Geeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/software-measurement-and-metrics/#:~:text=Software%20Measurement%3A%20A%20measurement%20is,an%20authority%20within%20software%20engineering.

Gitclear. (n.d.). Retrieved from https://www.gitclear.com/blog/the_4_worst_software_metrics_agitating_developers_in_2019

javatpoint. (n.d.). Retrieved from javatpoint: https://www.javatpoint.com/cocomo-model

Taghi Javdani, H. Z. (2012). *On the current measurement practices in agile software development, International Journal of Computer Science Issues.*

University, C. M. (n.d.). *Carnegie Mellon University*. Retrieved from https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283