Ellen Whelan

CSU33012

29 October 2019

Measuring Software Engineering

Brief: to deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

INTRODUCTION

This paper will delve into the field of software engineering and the concept of measuring it. The first thing I wish to establish is a definition of software engineering; what does it mean. Software engineering is often used as a synonym for computer science, but in fact for many it is a branch of computer science that focuses more on an engineering approach to developing software. Fritz Bauer, the german computer scientist, coined the term in 1968 as part of the NATO science committee in response to the 'software crisis'. In 1972, Bauer published the following definition of software engineering:

"Establishment and use of sound engineering principles to economically obtain software that is reliable and works on real machines efficiently[1]."

While Bauer coined the term software engineering, Margaret Hamilton is also credited with the promotion of the term 'software engineer' during her time on the Apollo project[2].

Software Engineering inherently has essential and accidental difficulties to overcome[3]. The challenge of developing software that is well designed, well tested and well implemented is larger than ever in the modern day, as demand grows for software and as technology continually advances. With the growth of open source, and improvements in accessibility, it's becoming easier and easier for anyone to develop software. This means the measurement of software engineering is more necessary than ever, to ensure that software is of good quality and to ensure software engineers produce well designed software.

WHY WE MUST MEASURE SOFTWARE ENGINEERING

So with this ever growing challenge of developing and maintaining high quality software, and the idea of measuring software engineering to guarantee this, what exactly do we mean by 'measuring' it? It doesn't at first glance seem like a quantifiable object. Software engineering is not a physical object and can't even be thought of as a physical object, so how could we measure it. Software metrics (measuring the software engineering process and assessing it in terms of measurable data) uses a variety of methods to measure software, to provide a formal means of estimating software complexity and quality.

Software metrics and measuring the process of software development, is of course key to improving the software development process. For software companies, and indeed most companies in this computer age, if they can improve their software development process to develop better software, more efficiently, it will give them the much sought after edge on their competitors. In a world, bursting with software companies, an edge is an extremely valuable commodity.

In general, software metrics are useful in three ways: helping us to understand and model the software engineering process and product quality, aids in the management of software development, aids the improvement of the software engineering process.

THE CHALLENGE OF MEASURING SOFTWARE ENGINEERING

Intuitively, anyone can see the challenge of measuring software engineering: how on earth can we measure something that seems so unquantifiable. Indeed it certainly is a challenge.

One problem encountered by software metrics is the vast amount of development methods that exist on software engineering. Different projects undertaken by software engineering teams will utilise different design methods such as the waterfall method, the incremental method and the verification and validation method. With different design methods, it can be difficult to come up with a method to measure the software engineering process that will fairly and accurately compare, software developed under different design approaches[4].

Unforseen size and complexity are one of the main reasons for failure of software engineering projects. Therefore the measurement of size and complexity of software is a very important metric to evaluate. Perhaps the most common method to measure the size of software is to count lines of code, but this method can be dependent on the software engineers building the project. For example, if the method is used in its purest form, the LOC figure may contain lines of ‘dead code’. Furthermore, if the coder wrote inefficient, long unnecessary function, this can also affect the LOC figure and the accuracy of the metrics.

Another problem that arises is choosing what goal to focus on when measuring. If a company decided to focus on productivity, software engineers working for the company may

begin to commit large volumes of garbage code in order to meet targets. Instead of improving software produced this would have the opposite effect and therefore illustrates the challenge of choosing the correct goal and method

Finally, not everything that can be measured is of value to a company. Therefore, the company embarking on a software measurement task has the additional challenge of identifying those metrics that add value to the project.

HOW CAN WE MEASURE SWENG

With the what and why discussed, it's now time to focus on how software engineering can and is being measured in industry. More and more companies are investing in analysing their software engineer employees performance using algorithms and computational intelligence ideas. These companies use various techniques to discern what a good software engineer is, and what makes them a good or bad engineer. We can consider lots of factors in the quality of an engineer: how many languages they know, how often they commit code to repositories, how many projects they work on, their ability to meet deadlines and so on. With all these factors to consider there is various methods to measure the success of a software engineer or team. These include algorithmic analysis, number of lines of code, bugs per 1000 lines of code, level of testing over the code, number of classes/interfaces, and the complexity of the code[5]. Most of these are easily calculable but may not be the most accurate method of measurement. For example, counting the number methods may cause issues where developers start adding unnecessary methods that aren't even run by program, in order to 'improve' their performance.

In this section, I will discuss some design approaches to measuring the performance of software engineering, as well as the use of computational intelligence in software metrics.

Design Approaches

Software engineers can take particular approaches when developing software in order to analyse and improve their performance. One such approach is the personal software process (PSP)[6].

Developed by Watts Humphrey, it was meant to apply the Software Engineering Institute's Capability Maturity Model to the software development process of individual developers. PSP disciplines the software development approach of individual developers in order to aid in the improvement and understanding of that developers performance. It also structures the way that developer tracks their predicted and actual software build. PSP enables engineers to analyse their work by reviewing the software's development time and issues or bugs. This analysis is key to improving their performance, the main goal of software metrics, by aiding developers in software quality management, software issue reduction, developing realistic commitments and in the improvement in the developers planning and prediction skills[6].

PSP has a structured level system, that software engineers integrate level by level into their existing design process in order to improve their software design process and the resulting software. The system has three stages split over multiple levels: planning, development (design, code, compile, test) and a 'post mortem' or review. In the review the software engineer records all data for the project and comes up with a coding standard and a plan for further improvement. This plan is called a 'Personal Improvement Plan' and includes ways to improve the software development process further for that individual software engineer.

Computation Intelligence Approaches

Definitely the more interesting approach is the approach of using computational intelligence, neural networks, deep learning and the like to measure and analyse the performance of software engineers and software engineering teams. The field of 'Artificial Intelligence' is developing extremely rapidly but is still in some ways quite primitive, essentially clustering large collections of data points into smaller groups - no where near the computational capabilities of a human mind. Computational intelligence is a subset of artificial intelligence, based on soft computing techniques and fuzzy logic, as opposed to the hard computing techniques of artificial intelligence (i.e. the idea that everything has a boolean value of 0 or 1). The computational intelligence idea of softer techniques is much more amenable to the natural,organic way human brains operate with partial truths.

Computational intelligence is certainly the technology that companies are currently queuing up to invest in, so it comes as no surprise that software companies (and companies with software divisions) are investing in computational intelligent approaches to measuring the performance of their software engineers. After all, if the company can have a machine analysing employee performance that's at least one less person they have to employ. If companies can input data on all their developers into some system and have the system 'intelligently' provide insights into which areas need more resources or changes in order to improve software quality and productivity.

Using computational intelligence to automatically calculate software metrics and assess the performance of software engineers, can and does help companies to achieve higher quality,

more reliable software. It can provide valuable insights and predictions of where problems and delays may arise and take steps to counteract those. Intelligent analysis of software engineering processes is certainly the more interesting and more effective way of improving software engineering processes going forward as demand for software only seems to grow.

EXISTING TOOLS AND PLATFORMS

Many platforms exist in industry at the moment for the measurement of software engineers performance. There is both code based platforms and time management platforms.

Code Based Platform

SonarQube is an open source platform that statically analyses code to provide insights into code quality. It automatically reviews software and code to detect bugs and vulnerabilities. Such vulnerabilities include duplicated code, coding standard deviations, unit test failures, code coverage shortages, code complexity , comments, bugs and security vulnerabilities. SonarQube records metrics on the code and provides evolution graphs for the user. Support for a wide range of languages is available with SonarQube, including Java, c/c++/c#, PHP, JavaScript, Ruby, SQL, Go, CSS, HTML, Python and more[7]. SonarQube also integrates with eclipse, visual studio and intellij IDEA integrated development environments, as well as with some other external tools such as GitHub. This wide range language support and integrability with such widely used tools means SonarQube is a widely used tool for software metrics, with 173 companies reportedly using SonarQube for code analysis[8].

Along with monitoring code quality SonarQube can be used for gaining insights into the technical debt of projects and can estimate the time it would take to resolve project problems.

SonarQube can also provide data visualisation to the user, to enable understanding of code quality. It is a valuable management tool for visualising the relationship between the software engineering process and how it affects the quality of the software produced.

Time Management Platform

Another tool that focuses on monitoring time management in software engineering projects and other projects is Toggl. Toggle has a web, mobile and desktop application which can be used to monitor time based on projects either automatically or based on manual user entries. The tool allows users to track the time budget of a project as it's spent and to make insights into productivity of software engineers working on the project. Toggl was originally built using Ruby on Rails and later transitioned to using Go in the backend with javascript running the front end.

Toggl's features are targeted towards management with an emphasis on relating time spent on aspects of projects and the effects of that time expenditure, in order to improve productivity. It can aid in the development of time budgets for projects and also in real time monitoring, and provide valuable insights into timings involved in software engineering projects.

Toggle has 1.6 million users as of 2016[9], and is used for time management by various companies including polish software company netguru, and UK company codecraft[10].

There are various other code based tools and time management tools available for measuring productivity and performance of software engineers and software engineering teams.

MORAL AND ETHICAL CONSIDERATIONS

As always with a new technology or new usage of technology the moral and ethics must be considered. These tools used for analysing the performance and productivity of software engineers, at the end of the day, are dealing with real humans. For example, ethics must especially be considered when you consider the fact that software metrics are used by companies to determine the people that are weakest and to terminate their employment.

We must consider what datapoints are ethical to track and what are unethical, and also how those datapoints are used. It must always be kept in mind that software engineers are people, and that the value monitoring a certain metric brings to a company, must not be at the expense of the real person involved. In my opinion, the value software metrics brings to companies and indeed to software engineers themselves, is too great to not utilise the tool but certain precautions must be observed to protect the data rights of the engineers.

For example, metrics like commit numbers and commit frequency may be beneficial to a company for identifying the productivity level of their engineers. Similarly, it might be beneficial for an engineer to monitor and maintain their own productivity. Furthermore, I don't think anyone would object to metrics of these nature as they are certainly not invasive and reasonable for your employer to want to keep track of.

However, on the other hand, tracking a metric like the location of a 'work from home' developer when they commit would be ethically problematic. In the software engineering industry, it is of course commonplace for developers to work from home and while it would be possible to track the location of these developers as they work, it would be ethically extremely questionable. In addition to being a violation of privacy, it would also be of no benefit to a

company to know whether their engineer was in their living room or the starbucks down the road when they committed that piece of code.

Transparency is also extremely important when considering the ethical implications of monitoring employees. Engineers must have knowledge of which metrics are tracked and also have trust in their managers[11]. This new way of monitoring software engineers productivity raises new digital ethics concerns, and therefore ethical considerations should be a more important part of the thought process, than ever before[12].

To sum up, software metrics to determine the quality of software engineers is a valuable tool to managers and software engineers alike, and I do believe it can be used in an ethical manner once the proper considerations have been made. Companies must take on the responsibility of ensuring only metrics that are valuable and not ethically problematic are tracked, in order to protect the privacy of their employees.

FINAL REMARKS

This report explored the uses and benefits of measuring software engineering, and the benefits are undeniable. Monitoring certain metrics associated with software engineering projects can help produce better software in a much more efficient manner. It can help reduce project failures by allowing managers to foresee time and complexity issues with projects, and which software engineers are best suited for certain tasks.

I do also believe, software metrics can assist software engineers to better themselves by highlighting areas they excel in and areas that require improvements. It can also help connect engineers with other engineers in order to help improve the skill level of the software community as a whole.

References

[1] Bauer, F.L., "Software Engineering", Information Processing, 71, 1972

[2] Lawrence, Snyder (2017). *Fluency with information technology : skills, concepts, & capabilities* ([Seventh edition] ed.). NY, NY.

[3] Frederick P. Brooks Jr, “No Silver Bullet- Essence and Accident in Software Engineering”. 1986

[4] Casper Jones, “Applied Software Measurement”

[5] https://diceus.com/top-7-software-quality-metrics-matter/

[6]https://en.wikipedia.org/wiki/Personal_software_process#targetText=The%20Personal%20Software%20Process%20(PSP,actual%20development%20of%20the%20code.

[7] "Multi-Language | SonarQube". Retrieved 2017-11-25.

[8] https://stackshare.io/sonarqube

[9] "Toggl Press Kit: Original Data, Infographics and Press Releases". *toggl.com*. Retrieved 2019-10-17.

[10] https://stackshare.io/toggl

[11]https://www.forbes.com/sites/williamcraig/2018/10/16/10-things-transparency-can-do-for-your-company/#4af55dd325d0

[12] Robin Boomer, https://www.gartner.com/smarterwithgartner/dos-and-donts-of-using-employee-data/