Measuring Software Engineering

CS33012 Software Engineering

Barbara Flora Molnar

18336339

Table of contents

1. How can software engineering be measured?

1.1 Why should we measure software engineering?

This is an important question to look at in the context of this topic. Why is it important to measure software engineering and the productivity of developers? Is it possible to collect data that will give an insight into what is happening and what can be changed to improve the process? These questions will be answered and discussed in detail in this essay. Every business should aim to continuously improve their offering and business operations. Therefore, it is important to find

new ways to measure performance, so that areas needing improvements can be identified and new approaches can be introduced to help drive the company forward. This will also lead to an increase in income, which is ultimately the goal of every business operating for profit. Analysing these metrics will also lead to lower costs for the company as there will be less wasted time, so overall projects will be completed faster. Other than monetary gain, it will also help workload be administered more efficiently, leading to better time management and reduced overtime for the workers.

1.2 What can be measured?

Another important aspect to look at is what information can be measured that will provide useful insight into the company's workings. Is it each individual developer that should be assessed based on their performance or should we also be looking at a wider team and how people work together? Afterall, a company is a larger team of workers who all contribute to a common goal, with everything being connected to each other.

Measures represent important data in all engineering areas as they show how things work and how to make changes to produce desired results. In software, there are two main types of metrics. These are product metrics and process metrics. Product metrics describe a product's characteristics such as size, complexity, design features, performance, quality and reliability. Process metrics refer to various aspects of the product development process, such as efficiency or fault detection, focusing on methods, techniques and tools that are used. In the following sections, there will be examples given to demonstrate these ideas.

1.3 Measurable Data

Measurable means that it is quantifiable and data refers to facts and statistics collected together for reference and analysis, based on definitions from Oxford Languages dictionary publisher. In software engineering measurable data can

signify a variety of aspects, depending on what area the company is interested in improving.

Measuring engineers' productivity

As mentioned earlier, what our metrics will be depends on what we are interested in. There can be a lot of different definitions for productivity, so it is important to first have a clear understanding of what it is the company is trying to measure. One approach to defining productivity could be to look at inputs, all the resources that go into developing software, and outputs, the end result in the form of the final product. However, it may be difficult to quantify the finished project, as it is a lot more than just code.

Counting the Lines of Code

One of the first ideas may be to count the lines of code a developer writes. However, after that initial thought, it should become obvious that it is a really bad way to measure performance. Doing something with more words or more lines of code does not make it better, if anything, it makes the quality worse. Fewer lines of optimised, readable code are a lot more valuable than hundreds of lines of inefficient code. This approach would only encourage engineers to write worse quality code. More than this, the same program implemented in different languages will differ in code length, because of the nature of the programming languages. For example, Java would likely use more lines than Python for the same functionality. Another issue would be that someone testing and maintaining the code would have a lot fewer lines attributed to them, than those initially developing the software. This would indicate that one is less valuable than the other, when in reality it may not be a case. Based on these examples, we can see that even though this is something we can measure, it does not mean that we should draw conclusions based on this metric. Maybe a way to be able to use this metric is to look for fewer lines of code per task, but also add a readability measure to it to ensure that others can understand what is being written as well.

Hours Worked

This is another example of something that is easy to measure and compare. However, similar to the previous example, it is not a good indicator of how productive an engineer is. The developers who complete tasks and solve problems in a shorter amount of time are better than those who take longer. Not only that, it has also been proved that the more hours are worked, the more the average productivity level per hour is reduced. The focus should be more on "quality over quantity", as the popular phrase goes. The amount of time spent on a project should be balanced out by the final result. Related to this, it is also important to keep in mind, for example, the level of experience of the engineer. Someone with several years of experience will be able to finish a task much faster than someone who has only been using the language for a single year. This could make it seem as though the inexperienced employee is laggard, based on the fact that they take longer to accomplish an objective compared to their seniors peers.

Code Quality

How can we define what good quality code is? Saying something is "good" or "bad" is subjective, therefore, what is seen as good practice and made the standard approach in one company may not be the same across the whole industry. However, there are some aspects that we can consider that software engineers seem to agree on. In an article on Perforce.com the key measures for code quality are reliability, maintainability, testability, portability and reusability.

Reliability shows how likely it is that the program will run without any failures over a period of time. Number of defects may be measured with a static analysis tool by looking at the mean time between failures (MTBF). Low defect counts are extremely important for writing reliable software.

Maintainability refers to how easy or challenging software upkeep is. Some areas to look at are the complexity, consistency and structure of software, among other things. One metric is not enough to measure maintainability, as it would not cover all that goes into it. The number of warnings about syntax errors and Halstead metrics for complexity are commonly used.

Testability means how easy it is to test different areas of the code to find faults. If testability is high, it shows that finding shortcomings of a program by way of writing tests is a good approach. A way to measure it is to see how many test cases are needed to find potential flaws.

Portability describes how platform independent a software is, how easy it is to convert onto different environments without it affecting the functionality. It may be difficult to find metrics to test this aspect of the program, if there are any at all. The best way to ensure portability is to continuously try out the code on different platforms and devices, so problems are identified early on, when they are less challenging to correct.

Software reusability looks at how reusable existing assets are within the development process, such as components, design, documentation, test suites and code. This area can be measured by looking at how interdependent different parts of the program are. Static analysis methods are a good way to test this.

Using these different approaches to try to determine and improve the quality of code is arguably a good way to measure productivity. It takes into account a variety of aspects that may differ between people, teams and companies. This means that it gives a better understanding of the performance of individual developers. Rather than looking at something that is easy to measure such as number of lines of code written or hours worked, the value of the work is evaluated within its context.

2. Computational Platforms to Collect and Analyse Data

2.1 Hackystat

Hackystat is an open source “framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data” according to their official GitHub Pages page. Hackystat is the third generation of Personal Software Process (PSP) data collection tools developed at the University of Hawaii.

PSP is a rigorous software development methodology for all steps of the process, from the early stages to the final product. It helps developers track their work, making it easy to be analysed at any stage in the process. This continuous observation helps trace performance and productivity, compare them to forecasts and plans and find areas needing more attention and come up with ways to improve them. In PSP, metrics are used to collect data which can then be analysed to come up with estimates for quality and length of development, how to plan the project and improve the development process and find any problems. Hackystat is a framework that "allows developers to collect process data and analyze them automatically" (Sillitti et al, 2003). It is done by a way of attaching software sensors to development tools that collect data that can then be queried by other web services to do higher level analyses. Hackystat services are created in a way that they can work well together with other components in the "cloud" of internet information systems and services available for modern software development (Hackystat, 2020).

2.2 GitPrime - Pluralsight Flow

GitPrime was a tool for tracking and improving the productivity of developers. It was acquisitioned by Pluralsight for $170 million in 2019. Pluralsight CEO Aaron Skonnard said that "marrying Pluralsight's skill measurement and skill development capabilities with GitPrime's developer productivity capabilities provides technology leaders with the most complete platform to improve efficiencies and speed up product development to deliver their digital transformation strategies".

GitPrime was connected to every major code repository service in use, including GitHub, Bitbucket and GitLab. Due to having access to such a wide variety of platforms, GitPrime captured data from a large number of repositories and gave unbiased insight into how developers use their skills. GitPrime made analysing data simple by visualising work habits, team's contributions and commits. GitPrime was used to measure the efficiency and performance of software development teams.

According to their website, Pluralsight is now the leading technology to help companies and teams create better products by equipping them with critical skills, improving processes and gaining insights through data, as well as providing strategic skills consulting. Pluralsight Flow provides engineering teams with data and visibility into workflow patterns to accelerate and improve the delivery of their product offerings. It transforms historical Git data into easy to understand insights and reports, giving objective data. They recognise that managing large teams is challenging as the volume of activity is overwhelming and stand-ups are not efficient. They have a feature called "Work Log", which shows the team's contributions, work habits, makes it possible to view commits and pull requests in one dashboard and helps identify bottlenecks early on.
By seeing who is contributing what and how much, it is possible to evaluate how they are doing, but also to plan better for future projects, sprints and reduce development time.

2.3 Trello

Trello is a web-based kanban-style application to make lists, especially for groups to organise themselves, to make collaboration easier and to keep track of the progression of work. Kanban comes from the Japanese word meaning "billboard" and it was Toyota car manufacturer that came up with the idea. Their thought was to focus on the demand of customers, instead of just putting products out into the market. This is part of what is known today as *Lean*, a set of principles that aim to minimise waste (defect, overprocessing, overproduction, waiting, inventory, transportation, motion, non-utilised talent) in the workplace and improve productivity. Although it originates from manufacturing, it has become popular in other industries, such as software development.

Trello is an application that supports these ideas and creates value to businesses by having it in a digital platform. They make it possible for workers to see a lot of information at a single glance. This means both the big picture but also details

in the project. Everyone on a team has access to these boards, so all members are on the same page in what tasks need to be done, what is in progress and what has been completed.

They also give users the option to “power up”, which refers to extra features that can be added to the application. It includes the integration of some other tools such as Bitbucket, Jira, Confluence or Google Drive. Other functionalities could be emailing, limiting the number of current tasks and adding an approval system between employees, among other things. This is all contained within the same application.

3. Algorithmic Approaches

3.1 Halstead’s Software Metrics

In 1977, Maurice Howard Halstead introduced some metrics to measure software complexity. This approach was developed to measure complexity directly from the source code, with emphasis on computational complexity. The idea is to determine a quantitative measure of complexity directly from the operators and operands.

He defines several measures that can be collected:

n1: number of distinct operators

n2: number of distinct operands

N1: total number of operators

N2: total number of operands.

From these base measures, other measures can be calculated:

1. **Halstead Vocabulary (n)**: total number of distinct operators and operands.

   Calculated as:

   $n = n1 + n2.$

2. **Halstead Size (N)**: total number of operator and operand occurrences.

Calculated as:

N = N1 + N2.

3. **Volume (V)**: proportional to program size, it represents the size of space necessary for storage. The unit size for volume is bits.

   Calculated as:

   V = N * log2(n).

4. **Difficulty (D):** relates to how difficult it is to write or understand a program (such as during a code review).

   Calculated as:

   D = (n1/2) * (N2/n2).

5. **Effort (E):** measures how much effort it takes to translate an existing algorithm into implementing it in a specified programming language.

   Calculated as:

   E = D * V.

6. **Errors (B):** an estimate of the number of errors (bugs) in the delivered program.

   Calculated as:

   B = V/3000.

7. **Testing Time (T):** measures how long it will take to translate a program into a specified programming language.

   Calculated as:

   T = E/S seconds, usually S = 18.

   Note: The number 18 comes from psychologist John Stroud who defined a moment as the time it takes for humans to carry out the simplest decision. The value of S has been recommended to be 18 for programming applications based on empirical research.

Halstead's approach is useful because it has measures that are simple to calculate, look at the overall quality of the software, predict maintenance effort, help with scheduling and can be used for any programming language.

3.2 Computational Intelligence

According to the IEEE Computational Intelligence Society, Computational Intelligence is the "theory, design, application and development of biologically and linguistically motivated computational paradigms". Computational Intelligence is a subset of Artificial Intelligence, often considered as the more human-like approach called "soft computing", which is tolerant of imprecision, uncertainty, partial truths and approximation. Some of the main aspects of soft computing are artificial neural networks, fuzzy systems and evolutionary computation.

1. **Artificial Neural Networks**

   Artificial Neural Networks (ANNs) are inspired by the way the biological nervous system and the brain functions. They are usually used for specific situations, such as data and pattern recognition, image processing and compression, stock market prediction and security. The aim is for the computer to have a more human-like approach. They are suited for solving problems where there are no clear steps or rules to follow.

2. **Fuzzy Systems**

   Fuzzy logic is a concept first introduced by Dr Lofti Zadeh, as a way to represent human knowledge which is more imprecise by nature. Fuzzy logic is based on degrees of truth rather than the commonly used "true or false" values. There is room for partial truths. Values of variables can be anywhere between 1 and 0, both inclusive. According to Dogan (2016), a huge advantage of fuzzy logic is that it offers a practical way for designing and stabilising nonlinear control systems, which are difficult to do using traditional methods.

3. **Evolutionary Computation**

   Evolutionary computation is based on natural evolution. An evolutionary computation algorithm starts off with a population that can be fed to it or

created randomly. This first population represents individuals that offer different solutions for the problem at hand. Then, some operators inspired by things happening in natural evolution, such as crossover, mutation, selection and reproduction, are applied to individuals (Sen, 2015). Due to the population size having to be maintained, some individuals are removed. These steps are repeated until the end criterion is met. The most commonly used criterion is reaching the maximum number of generations given. The most competent individual is then selected from the population. This is a major area in research for adaptation and optimisation.

4. Ethics Concerns

With all of our devices in everyday life tracking our digital footprint, people are already wary of the continuous development of technology and the increase of companies using data collection and data analysis to back up all of their decisions. However, there seems to be no sign of slowing down, rather it is becoming more and more popular.

There are a number of questions and concerns that can be raised with regards to the ethics perspective of measuring software engineering. Are developers aware of the ways in which they are being evaluated by their employers? Do they know about all of the different metrics? Are they aware of all the data that is being collected about them and their performance? Do they know how much those matter with regards to their performance and position in the organisation?

Personally, I see where individuals and firms are coming from in trying to come up with ways to measure productivity in software, however, by doing the research for this assignment it has become clear that none of the options are ideal. All of the approaches I came across take away the human side of working with everything being measured and evaluated based on what aligns with an organisation's values and goals. No two developers are the same, they all have different skills and experiences, as well as personalities and preferences, among

other things. I think by focusing on these measures, valuable people who do not perform well based on the metrics could go underappreciated and even let go.

Then there is the concern about being constantly observed. This may put even more pressure and stress on each individual and team, possibly altering their behaviour in the process as well. The *Hawthorne Effect* is when an individual changes their behaviour due to their awareness of being watched. However, if this is the case, that means employees' performance and behaviour are not an accurate representation of how they actually work. This may also create a more hostile environment in the workplace, by making colleagues compete against each other and compare scores, instead of helping each other improve.

At the same time, I do think providing individuals and teams feedback is useful, because that way they can identify areas in need of improvement, which gives them the opportunity to learn and grow. In my opinion, the problem arises more when the data is exploited or treated as the most important aspect of a worker's performance.

Another point to mention is that at the end of the day most businesses want to make a profit. They want to optimise their operations in any way they can, to save money, make money and keep their spot in the market, if not gain a larger market share. Therefore, the privacy concerns may get ignored to help the organisation move forward.

I do not know how to address these concerns, but based on my research other people are talking about them and the industry seems like it may be moving toward the right direction.

Bibliography

Altvater, A. (2017). "*What are software metrics and how can you track them?*" [online] Stackify.com, Available at: https://stackify.com/track-software-metrics/

Atlassian. (2020). "*Kanban*" [online] Available at: https://www.atlassian.com/agile/kanban

Bellairs, R. (2019). "*What is Code Quality? And How to Improve Code Quality*" [online] Perforce, Available at: https://www.perforce.com/blog/sca/what-code-quality-

and-how-improve-code-quality#:~:text=The%20number%20of%20defects%20%E2%80%94%20and,important%20metrics%20of%20overall%20quality.&text=Number%20of%20open%20defect%20reports,defects%20per%20lines%20of%20code).

Belyh, A. (2019). "*Overview of qualitative and quantitative data collection methods*" [online] cleverism.com, Available at: https://www.cleverism.com/qualitative-and-quantitative-data-collection-methods/#:~:text=These%20are%20data%20that%20deal,%2C%20price%2C%20and%20even%20duration.

Dogan, I. (2016). "*An Overview of Soft Computing*" Procedia Computer Science, Volume 102, Pages 34-38, Available at: http://www.sciencedirect.com/science/article/pii/S1877050916325467

Duffy, J. (2020). "*Trello*" [online] Available at: https://uk.pcmag.com/productivity-2/67278/trello

Gay, C. (2016). "*8 wastes of lean manufacturing*" [online] Available at: https://www.machinemetrics.com/blog/8-wastes-of-lean-manufacturing

GeeksForGeeks. (2020). "*Software Engineering | Halstead's Software Metrics*" [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

GeeksForGeeks, User pp_pankaj. (2020). "*Software Measurement and Metrics*" [online] Available at: https://www.geeksforgeeks.org/software-measurement-and-metrics/

Goldman, J.G. (2014). "*How being watched changes you - without you knowing*" BBC [online] Available at:https://www.bbc.com/future/article/20140209-being-watched-why-thats-good

Green Carmichael, S. (2015). "*The Research is Clear: Long Hours Backfire for People and for Companies*" [online] Harvard Business Review, Available at: https://hbr.org/2015/08/the-research-is-clear-long-hours-backfire-for-people-and-for-companies

Hackystat. (2020). "*Hackystat"* [online] Available at: https://hackystat.github.io/

Hakes, T. (2018). "*How to Measure Developer Productivity*" [online] 7pace.com, Available at: https://www.7pace.com/blog/how-to-measure-developer-productivity

IBM. (2020). "*Halstead Metrics"* [online] Available at: https://www.ibm.com/support/knowledgecenter/SSSHUF_8.1.0/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

Javatpoint. (2020). *"Software Metrics"* [online] Available at: https://www.javatpoint.com/software-engineering-software-metrics

Kanbanize. (). "*Kanban Explained for Beginners | The Complete Guide*" [online] Available at: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

Pluralsight. (2019). "*Pluralsight to acquire GitPrime"* [online] Available at: https://www.pluralsight.com/newsroom/press-releases/pluralsight-to-acquire-gitprime#:~:text=Pluralsight%20To%20Acquire%20GitPrime%2C%20the%20Leading%20Developer%20Productivity%20Platform,-May%201%2C%202019&text=Silicon%20Slopes%2C%20Utah%20%E2%80%94%20Pluralsight%2C,for%20%24170%20million%20in%20cash.

Pluralsight. (2020). "*Pluralsight Flow"* [online] Available at: https://www.pluralsight.com/product/flow

Sen, S. (2015). "*A Survey of Intrusion Detection Systems Using Evolutionary Computation"* [online] ScienceDirect, Available at: https://www.sciencedirect.com/topics/computer-science/evolutionary-computation

Sillitti A. et al. (2003). "*Collecting, integrating and analyzing software metrics and personal software process data"* [online] IEEE Proceedings of the 29th Euromicro Conference p. 336-342, Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf

TutorialsPoint. (2020). "*Software Measurement Metrics*" [online] Available at: https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm#:~:text=Software%20metrics%20is%20a%20standard,process%20metrics%2C%20and%20project%20metrics.

Wikipedia. (2020). "*Hawthorne Effect*" [online] Available at: https://en.wikipedia.org/wiki/Hawthorne_effect#Interpretation_and_criticism