Measuring

Software Engineering

Report

Author: Samuel Alarco Cantos

Contents

Introduction

Software engineering has become one of the most dominant and relevant engineering disciplines of the modern era. Traditional industries such as transportation, communication, entertainment, and hospitality are gradually being transformed or outright disrupted by the software revolution. As software engineering is ingrained into many companies and processes, becoming a vital part of countless businesses and enterprises, a simple but profound question has been raised: can we measure it?

It is the ubiquitous temptation of entrepreneurs, managers, investors, and team-leads alike: can we measure the performance of a team, quantify the quality of a product, calculate the efficiency of an employee. The success of the modern business and managerial models are arguably based on the usage of useful and actionable metrics. As software engineering quickly develops and becomes such an important discipline, it is inevitable that the same questions are asked. Even software engineers themselves are usually interested in their own performance.

However, is this possible? Are these useful questions to ask? Can they even be answered with any degree of accuracy or actionability? What are the ethical consequences of measuring software engineers as individuals and teams with the data and methods we have available? What conclusions can we draw from these measurements and how should we act on them? On this report, we will explore these questions and try to spark a discussion on the most important topics of interest.

Measuring Engineering

In this section we will explore what is meant when talking about measuring software engineering. I will discuss several aspects and dimensions that could be measured, together with the motivation and relevance for such measurements.

Measuring Code

The first aspect that comes to mind when measuring a process such as software engineering is measuring the outcome or product of said process. This can be thought of in two ways: the end-product that is the software shipped to the client, and the source code which is the raw artifact created by the software engineering process. The multiple ways of measuring the efficiency, impact, and success of a product in the market have been thoroughly researched and discussed in literature. I will not dwell in them since they are also dependent on extraneous factors such as business planning, product design, marketing, etc. These are outside the scope of this report. The analysis of the source code is of much more interest to the measurement of software engineering.

When thinking on how to measure software engineering through the source code generated, several metrics quickly come to mind. We could measure things such as lines of code written, or changed by an engineer, quality of the code, idiomaticity, or complexity. Some of these, such as the lines of code per commit, are readily available from version control systems such as git. Others, such as code complexity, can be calculated through certain mathematical or statistical methods such as cyclomatic complexity, even if the accuracy is arguable.[1] In general, these metrics are easy to calculate and can give a general picture on the work of a software engineer or team. However, they are rather useless without further context. Different programming languages, because of their syntax and idiomatic practices, are more or less verbose and have different average line counts. These can further vary due to specific company style guides and best practices. Complexity can be difficult to properly quantify, and often depends on a balance between efficiency, readability, and functionality.

Finally, these measures are easy to game. If a developer is being measured by lines of code written, they will write more verbose and possibly inefficient code. If they are being judged by the number of commits, they will just make more micro-commits. In general, using these metrics as a main source of evaluation for software engineers has unexpected and undesired effects, rendering them useless by themselves beyond giving some nice statistics about the work of a developer.

Measuring the Process

It is far more useful to measure software engineering as a process, not just as its end-product. By measuring the process through a wide variety of metrics, we can identify strengths and bottlenecks at individual, team, and inter-team level. All together, they can give us crucial actionable insights into the performance of engineers and engineering teams.

[1] https://www.perforce.com/blog/qac/what-cyclomatic-complexity, accessed 20/12/2021

The software engineering process can be framed in terms of **cycle time.**[2] This is a term that comes from the manufacturing world. When referring to code and specifically to software features, it represents the time it takes for a feature to go from design/inception, through implementation, code reviews, testing, integration, to the moment it is released into production. Each of these steps have their own sub-metrics which give further insight into performance of the team. The cycle time can provide an actionable metric describing how well an organization is working, of if there are bottlenecks.

Several seminal studies into these methods were carried out by DORA, the DevOps Research and Assessment team. They were acquired by Google in 2018 and have as their goal "to understand the practices, processes, and capabilities that enable teams to achieve high performance in software and value delivery."[3] In the book *Accelerate*, they expounded 4 metrics which they claim can measure the performance and delivery capabilities of software engineering teams. These are:

1. **Lead Time:** The time it takes for a feature or change to go from inception to deployment. This is a close parallel to the cycle time mentioned above. Slow LTs indicate bottlenecks and inefficient processes in a team or individual.
2. **Deployment Frequency:** How often are new versions deployed in production. This is especially relevant in the context of agile programming, as it indicates how fast software is being developed and (hopefully) improved. DORA suggests that high deployment frequencies indicate fast development with small changes pushed often to production. This is generally considered safer and better practice than slower, bigger deployments.
3. **Change Failure Rate:** How often deployments of changes and new features lead to incidents. It is useful to indicate the quality of the code being produced and pushed to production. It also shows the quality of the testing pipeline being used before deploying code. The aim should always be to reduce this metric.
4. **Mean Time to Restore:** Complementing CFR, MTTR measures how long it takes to recover from a failure. This can be business critical as failures can cause partial or total

[2] https://www.swarmia.com/blog/measuring-software-development-productivity, accessed 20/12/2021.
[3] https://jellyfish.co/blog/dora-metrics-101, accessed 20/12/2021

blackouts, leading to economic losses and customer dissatisfaction. As with CFR, MTTR should always be minimized.[4]

These 4 metrics have proven to be quite efficient at measuring the workflow of software engineering teams. They also help to position single developers in the workflow and measure their impact at different points of the development pipeline. Many companies such as Linearb.io[5] and Swarmia[6] have adopted these metrics as part their main products, helping software organizations measure and optimize their software engineering processes.

Measuring Collaboration

Collaboration and related concepts such as “siloing” can be crucial aspects in a software engineering organization. Collaboration is one of the main ways in which software engineers build cross-domain knowledge of the organization´s products. It is the main way of avoiding knowledge being lost when engineers leave a team or company. It is also the source of new insights and creative solutions. Therefore, it is in the main interests of organizations to measure the level of collaboration in and across teams. We can identify several dimensions of collaboration which can be measured:

1. **Collaboration across Features:** Measures the number of developers working on a single feature. The optimal number depends largely on the feature being worked on. Working together on a single feature has its communication and organizational overhead, but it also contributes to sharing knowledge across the team and to bring different viewpoints to the same problem.
2. **Collaboration across a Codebase:** If large swaths of a codebase are only touched by a single developer, the organization is at a high risk of losing precious knowledge if that developer is lost. It is also more difficult to onboard more developers as the codebase grows but knowledge is not properly shared across the organization.[7]

The interpretation of these metrics is highly dependent on the product being built, company culture, the current situation of the organization etc. Having said that, these are metrics that

[4] *Accelerate, The Science of DevOps,* Nicole Forsgren, Jez Humble, Gene Kim, 2018.
[5] https://linearb.io/dora-metrics, accessed 21/12/2021
[6] https://www.swarmia.com/blog/measuring-software-development-productivity, accessed 21/12/2021
[7] https://www.swarmia.com/blog/measuring-software-development-productivity, accessed 21/12/2021

any manager should take into consideration when analysing the interaction between members of a team.

Measuring Health

A metric that is not often considered when talking about software engineering is the health of engineers and the team. This is by necessity a more qualitative metric than those discussed, as they largely depend on feedback given by team members and management. The measurements are often collected in the form of survey answers given at regular intervals. The questions might vary across organizations, but they should likely include inquiries into issues such as motivation, feeling of burnout, manageability of the workload, levels of stress, communication with other members of the team, perceived bottlenecks, conflicts with management, etc.

In software-driven organizations, the software engineering teams and most importantly the software engineers they are made of are one the most important assets. It is therefore of the outmost importance to make every effort possible to regularly evaluate their working health. This makes sure they are working at their full potential, or if anything could be improved. Failure to do this will often lead to bad company culture, toxic working environments, and ultimately a decrease in productivity and the outflow of talent elsewhere.

Existing Platforms and Resources

After having explored the various ways in which software engineering can be measured, we must focus our attention into the technologies that make this analysis possible. Services such as GitHub and Gitlab have made it possible to access immense amounts of data on the software engineering process. Furthermore, the rise and availability of utility computing make it possible to process vast amounts of data with a very low upfront investment. Finally, numerous companies have started offering such services to companies in the form of SaaS products, making it easier than ever to implement sophisticated metrics into every organization´s pipeline.

Version Control Platforms

Platform such as GitHub and Gitlab offer cloud hosted version control functionality to individuals and companies. They have recently become crucial parts of software collaboration and deployment pipeline, and now count with hundreds of thousands of individual repositories, many of them public. These platforms come with an additional advantage both for organizations and researchers: they make a number software engineering metrics very easy to obtain or calculate by making a wide array of data features readily available through APIs.[8]

Cloud Computing

Code repository and version control platforms like GitHub calculate certain basic metrics for you. These include frequency and size of commits, or developer profiling (proportion of commits against code reviews, discussion etc). However, the true power lies in the data they can provide through their APIs on practically every aspect of the software engineering process. To take advantage of this data we need to access to considerable computing power. This would have been difficult to achieve for the average individual or company a couple of years ago. Nevertheless, thanks to the advent of cloud computing providers, companies and individuals now have cheap access to considerable computing power on a scale never seen.

Cloud providers such as Google Cloud, Amazon AWS or Microsoft Azure offer many different options to rent their vast computing resources. The most basic are in the form of virtual machines (VMs) which can be accessed remotely and configured with the desired hardware requirements. More advanced options include clustering solutions using technologies such as Kubernetes. These can be used to automatically scale machine clusters depending on the analysis tasks being done. Finally, most prominent cloud providers offer purpose-built machine learning products that can help companies and individuals efficiently analyse their data using machine learning techniques.[9]

Dedicated Products

While cloud providers offer developers and companies unprecedented computing resources to exploit at a very competitive price, they are mainly providers of infrastructure. In order to extract, process, analyse, and draw actionable insights from software engineering metrics, solutions need to be built on top of that infrastructure. This is of course software. Depending on the sophistication of the analysis being carried out, such solutions will probably be

[8] https://docs.github.com/en/rest, accessed 21/12/2021
[9] https://cloud.google.com/vertex-ai, accessed 21/12/2021

sophisticated and require specialized talent, and substantial inversion of time and resources. This is especially true if software engineering is being measured at a company-wide scale. Large companies might have the human capital and resources to spend, but small organizations and individuals likely will not. Even if large companies possess the necessary resources to accomplish this task, it might still be a considerable time investment, time that could be better used somewhere else. This where specialized products come into play.

Given the demand for proper measurement of software engineering, several start-ups and companies have started offering products that accomplish just this. They can integrate into a team's pipeline, automatically collect data on a continuous basis, and provide teams and management with detailed insight. Companies like Linearb[10], Swarmia[11], or Haystack[12] are good examples of such services. Each service offers slightly different metrics and emphasize different approaches, but they all aim to easily integrate into a team's existing pipeline to make onboarding as simple as possible. Using these tools has the advantage of utilizing robust, heavily tested, market-proven products without investing considerable resources and time into developing your own. They are often the best choice for most companies, unless very specific requirements need to be met. In that case a company will likely have to develop their own software.

Methods and Algorithms

We have discussed several metrics of software engineering and gone over available resources that enable us to do the necessary computations. Before discussing the ethical considerations and consequences of such an analysis, it is important to give a small overview of the different methods and algorithms that are used to calculate these metrics and give actionable insights.

Counting and Aggregation

Among the simplest methods available in software engineering measuring, we find counting and aggregation of data. These are the methods involved, for example, in counting line changes, commits per developer, features completed, and calculating cycle time average. Even if straightforward, they form the basis for most of the other more sophisticated methods

[10] https://www.linearb.io, accessed 21/12/2021
[11] https://www.swarmia.com, accessed 21/12/2021
[12] https://www.usehaystack.io, accessed 21/12/2021

such as machine learning or expert systems. In fact, most of the metrics available in the commercial platforms discussed in the previous section would fall under this category.

A level beyond this, we can start using more complex compound calculations to distil deeper knowledge from the data. A good example of this is the already mentioned cyclomatic complexity of the code, which is a quantitative metric of the complexity of a piece of code. Specifically, it calculates the number of independent paths present in source code.[13] This involves the counting and aggregation of a series of features in the code, such as the number of decisions present in the form of if-else statements. Algorithms such as these can provide us with more complex and nuanced metrics. However, they should not be interpreted outside the context of the piece of code or team being analysed.

These metrics need to be coupled with powerful visualizations in order to give decision makers a meaningful picture. This is the reason most platforms place so much emphasis on the visual aspect of their dashboards. After all, we are visual creatures, and we often need data to be represented in a visually meaningful and appealing way to make proper sense of it. While these methods are powerful, they need to be interpreted with a context to derive meaningful insights from them. This is where more sophisticated methods such as expert systems and machine learning come into play.

Expert Systems

Expert systems are considered by some to have been the first successful forms of AI.[14] They work by using if-else style rules which form part of a knowledge base, together with an inference engine, to work through a problem or dataset and derive meaningful conclusions. Research on these systems was done actively in the 80s and 90s to a measure of success.

Given the complexity of software engineering measurement, research has been done on the usage of expert systems to aid non-expert individuals, such as management personnel. An example of this is the 2003 paper by Yinxu Wang et al.[15] which introduced a web-based expert system just for this purpose. The user configured the system with the software metrics they wanted to obtain from a codebase, and the system then requested the necessary data from

[13] https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity, accessed 21/12/2021
[14] https://en.wikipedia.org/wiki/Expert_system, accessed 21/12/2021
[15] *A Web-based software engineering measurement expert system*, Yingxu Wang, Behrouz H. Far, Shuangshuang Zhang, January 2003

the user, processed it, and displayed the corresponding feedback and insights. The expert system contained rules representing a wide variety of software engineering metrics, together with rules on how to process codebases and interpret the goals of the user.

Expert systems had the advantage of being relatively easy to use by inexperienced user. However, the inference engines were often rigid and limited to the expert knowledge that could be expressed in the form of rules. Nowadays, inference engines similar to those used by expert systems are so integrated into larger solutions, such as the platforms mentioned in previous sections, that they are no longer called expert systems even though they implement very similar functionality. Their limitations are also being overcome by newer techniques, such as machine learning.

Machine Learning

Machine learning refers to a wide array of algorithms and techniques that can adapt and improve their results automatically through "experience", i.e. data.[16] It is considered to be a sub-field of AI. Many of the methods used are statistical in nature and strive to imitate to some extent the way the brain processes information and "learns". The aim of machine learning is to "find generalizable predictive patterns"[17] that can be applied to unseen samples, as opposed to inferring information on a sample which is the general aim of statistics.

Machine learning is being used more and more in the field of software engineering measuring, specially to find patterns in the data and uncover actionable insights. Given the massive amounts of data produced by the software engineering process, it is daunting for humans to go through the thousands of variables and uncover relationships without the help of computers. This problem is made even more difficult when we realize that the answers probably do not apply across teams. If we want to answer them algorithmically, we need a form of algorithm that can adapt to the data available from each organization and team, while still being able to use a general body of knowledge. This is where machine learning methods truly shine.

What separates an efficient team from an inefficient team? Can we predict developer burnout from the data available? Is it time to fire a developer? These are the types of questions

[16] https://en.wikipedia.org/wiki/Machine_learning, accessed 22/12/2021
[17] https://en.wikipedia.org/wiki/Machine_learning, accessed 22/12/2021

machine learning is already trying to answer or may answer in the future. Research is still ongoing on this front, but it seems very promising. However, machine learning methods also present a wide array of ethical issues that need to be carefully discussed and addressed. The following section will give an overview of these issues.

Ethical Issues

In the previous section, we have briefly touched on several methods to calculate and analyse software engineering metrics. On this final section, I wish to start a discussion on the ethical and moral consequences of this analysis, and specially the decisions made from these metrics. The ethical issues arising from software engineering measuring are made even more complex and nuanced given the fact that the measuring usually takes place in a corporate setting, and the subjects do not have much choice as to what data can be collected and how it will be used.

The first ethical concern is the issue of consent. As mentioned above, software engineering measurement and research usually happens in a corporate setting, not an academic one. Employers do not necessarily offer a choice to employees on how their productivity will be measured. Their work is the company´s property, and therefore it is difficult to limit the scope of what they can collect and analyse. In Europe there is the GDPR legislation which may put limits on personal data, but this is not always relevant to the metrics described above. If consent is not really a choice, companies should at least make clear what data is collected, how it is analysed and what conclusions are drawn from it. Care should be taken if employee data is being processed by third parties. In this case efforts should be made to anonymize and protect employee data, as a data breach of any sort could cause irreparable damage to the software engineers being measured.

Of course, there is a clear issue when software engineers are made aware of how they are being measured: practically every metric can be gamed. If there are incentives to perform in a specific metric, software engineers will very probably find a way to tweak their work to exceed in that metric. The problem is that this does not necessarily translate to higher quality work. If the metric is line count, engineers will write more verbose work, if the metric is number of commits per issue, engineers will make lots of small commits, if the metric is number of Slack messages per feature request, engineers will extend their conversations at length on every little issue. This poses an ethical issue to those being measured: do they keep working honestly in the way they understand will produce a higher quality output, or do they instead work in a way to optimize the way they are measured. In the best case these two aims should converge, but generally they never do so.

Of all the issues present when measuring someone, maybe the most important ones are the issues dealing with how the insights are used. What actions can derive from that measurement? Is the metric accurate enough to make that decision? We live in a world of data-driven decision making. Using data to make decisions has the aim of being impartial and unbiased. The saying goes that "data does not lie". The promise is that if we base hiring, promotion, or firing decisions on data alone then we do not have to worry about personal biases. However, if decisions are being made on the results of a variety of algorithms then management needs to be very clear on what does results truly signify, and how the algorithms used have processed the data. Not only, the employees being measured should be made aware of and understand the reasoning behind decisions being made for them i.e. the standards must be public. As mentioned, this can lead to the "gaming" of the metrics.

Clearness on how data is being processed and conclusions being made is not always easy or even possible. This is one of the main issues with machine learning techniques. As discussed, machine learning methods can be extremely powerful in extracting patterns and insights from large amounts of data. Machine learning models are often used to automate decision making processes. The problem is that it is often very complicated or even impossible to understand how such analysis and decisions are made by the machine learning model, especially when neural networks of different kinds are used. These models are often treated as black boxes: data goes in, results come out. The way it works remains sort of mysterious and research is still ongoing. This is difficult to accept when important decisions such as firing or promoting someone are being made based on these models. Not being able to explain how the model is making recommendations could be seen as equivalent to not making the standards known to the parties involved. Moreover, research has shown that machine learning models can be highly biased depending on how they are trained. Even if a representative dataset has been used to train the model, bias can still be generated as a result of implicit biases present in the general population. Since machine learning models amplify patterns found in the data, these biases are usually amplified too. If the model is not properly understood, these biases could go unchecked and work unfairly in regards of certain software engineers or teams.

In conclusion, while software engineering measurement can provide very useful and actionable insights, care should always be taken to make sure decision-making bodies understand how these metrics are collected and interpreted. Software engineers and team should be made aware of the data that is being collected, how it is being processed, and what decisions are being made using it. If possible, consent should be requested to provide a trusting and open culture in the workplace.

Conclusion

We have gone over the different ways software engineering can be measured through different metrics and what computing resources are available to do such research. We have further explored different methods to process the data collected and derive actionable insights. Finally, we have discussed a variety of ethical issues that must be thought of specially when making decisions using these metrics. The recent global economic success has been in part due to the boom in data-driven decision making. I truly believe that measurement of software engineering, as with every process, is the future. It will benefit individual engineers, teams, and organizations to constantly improve and grow. However, care should always be taken to develop fair and unbiased methods that do not harm the parties involved.

Bibliography

1. https://www.perforce.com/blog/qac/what-cyclomatic-complexity
2. https://www.swarmia.com/blog/measuring-software-development-productivity
3. https://www.swarmia.com
4. https://jellyfish.co/blog/dora-metrics-101
5. *Accelerate, The Science of DevOps,* Nicole Forsgren, Jez Humble, Gene Kim, 2018.
6. https://linearb.io/dora-metrics
7. https://www.linearb.io
8. https://docs.github.com/en/rest
9. https://cloud.google.com/vertex-ai
10. https://www.usehaystack.io
11. https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity
12. https://en.wikipedia.org/wiki/Expert_system
13. *A Web-based software engineering measurement expert system*, Yingxu Wang, Behrouz H. Far, Shuangshuang Zhang, January 2003
14. https://en.wikipedia.org/wiki/Machine_learning