Measuring Software Engineering Report

Ajchan Mamedov

19335023

Contents

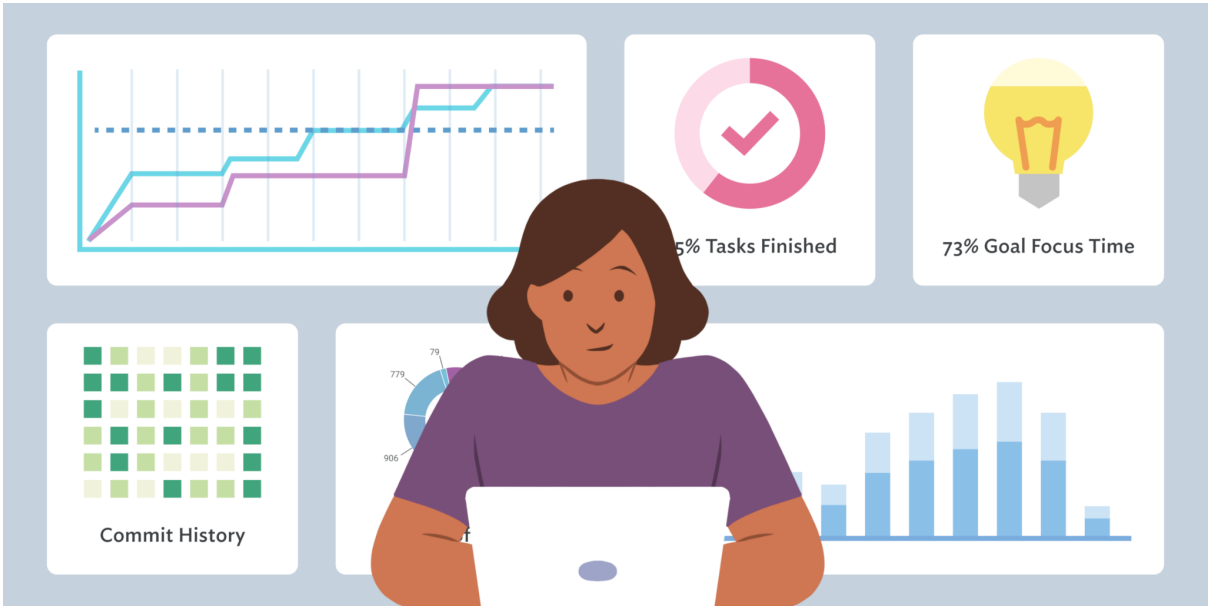
Introduction

Software engineering is one of the most is a relatively new field of engineering, that involves the steps that an engineer takes to design, develop and maintain software. It has gradually become one of the most important and relevant engineering disciplines of the modern era. As more and more industries have integrated software engineering, it has become a vital part of countless businesses.

The exponential growth of software engineering has brought a need to measure and evaluate this process. But can something like software engineering be evaluated and if yes, how do we measure it? As more and more entrepreneurs, managers and investors, want to evaluate the performance of software engineers and teams of software engineers, the question is how do we do this?

In this report, we will explore these questions and discuss how can we measure the process of software engineering, what data do we need to do so and is ethical to do this?

Measuring Software Engineering



In this section, we will take a look at what is meant by measuring software engineering. We will also take look at the software development life cycle and its phases and we will look at what can be measured in software engineering.

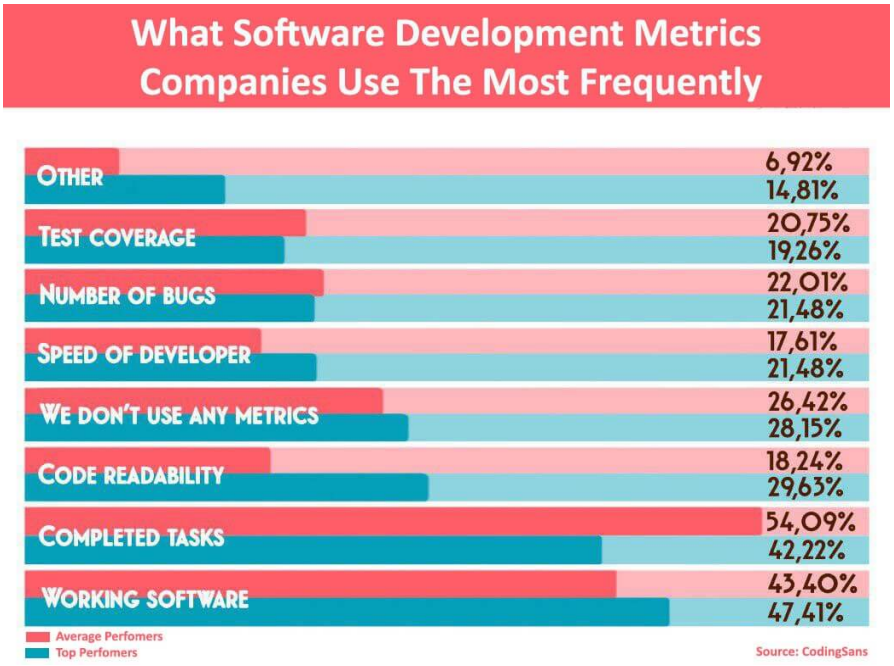
Measuring software engineering could be divided into two categories one would be measuring the end product and the other would be measuring the whole process.

Measuring the end product

Measuring the end product could be done in two ways, one would be measuring the source created by the software engineering process and the second would be the end product delivered to the client after everything has been fixed and honed. But no matter which way it would be done, the thing that would be measured would turn out to be the code written.

Some of the metrics that could be used to evaluate the code written would be:

- Lines of code written or changed by the engineers or the team
- Number of commits done by each engineer or the overall number done by the whole team
- Quality of the code, such as readability, the test cases passed, the number of bugs, etc.
- Gameability



The metrics like the number of commits and lines written are readily available by the version control systems, such as git. These metrics in theory should help organisations determine how productive an engineering team or an engineer is. However, this alone is often not enough, so organizations often combine these measurements, with the measurements of the quality of the code to get a more precise indication of persons or teams productivity. But because the programming languages are so different, their syntax can create more or fewer

lines of code on average. This further increases the difficulty of measuring the productivity of teams and people just by the code they have written.

These measurement criteria also can be easily gamed and be taken advantage of. If a developer or a team is pressured by the number of lines of code they have to write, they might just write more verbal code, which might end up very inefficient. If they are judged by the number of test cases their code passes, they might create more test cases or solely focus on passing test cases and not make the code efficient or readable. The number of commits can be gamed similarly, just by making more micro-commits - committing every little change done, no matter how insignificant it is.

In conclusion, measuring the productivity of people and teams, using these metrics isn't the best way of evaluating a software engineer, as these measurements can be easily gamed. These measurements are pretty useless, as they only provide nice and concise statistics of the work that the developer has done and they can only be useful if used together with measurements of the software engineering process.

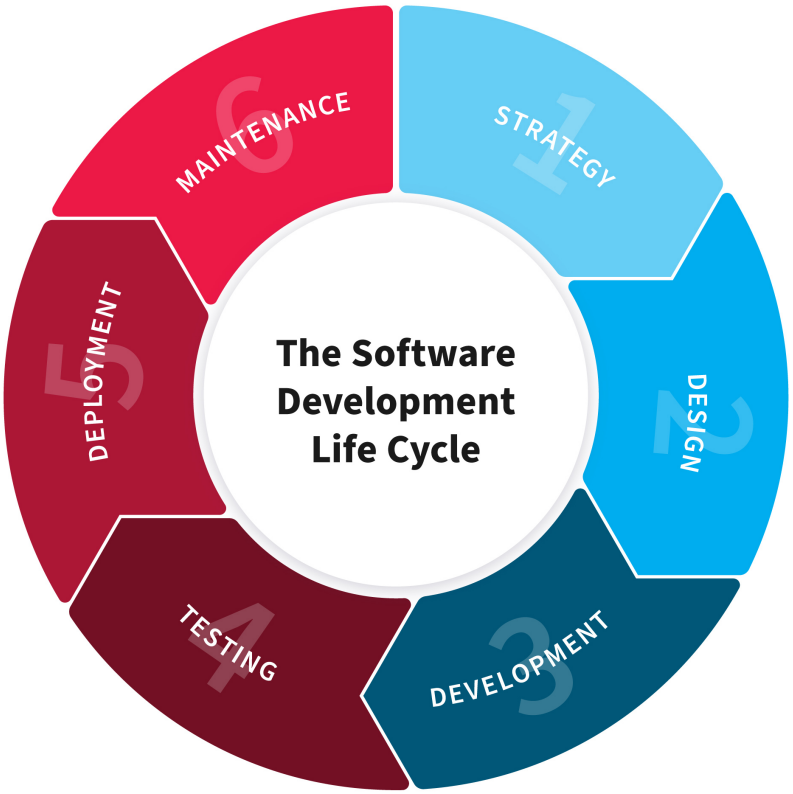
Measuring the process

It is far more useful to measure software engineering as a process, rather than an end-product. We can learn more about measuring the software engineering process, by looking into the software development life cycle.

The software development process includes multiple stages, which together create its life cycle.

The stages are:

- **Requirements analysis** - in this phase the requirements are gathered, in other words, the features of the system are determined.
- **Design** - the design of the software and system are created from feature specifications outlined in the requirements analysis phase.
- **Development** - this phase consists of the most important part - the coding.



- **Testing** - in this phase software engineers check for bugs and errors in the software, if the previous phases were done correctly, this phase shouldn't take too long.
- **Maintenance** - this phase encompasses the deployment of the finished software and maintenance of it, in case any more bugs appear in the future.

To measure the process of software engineering, the efficiency of these phases can be measured. There are multiple ways to do this, one such way would be to check the time it takes for a feature to go from requirement analysis to development and then to deployment. This is also called a cycle time, which measures the time it takes for the software development cycle to be completed. Another way to measure the process of software engineering would be to how often do we develop new features for the existing software and how often can we deploy this. Developing and deploying software are different things, as deploying software requires a lot of testing beforehand. It is also important to have multiple people working on the same software as if only one person will complete the whole feature set, if he is gone it will also mean losing important knowledge about a codebase. Measuring how often a developed software fails the testing is also important as it indicates that these software engineers, don't write quality code and it is a good indicator of the software engineers productivity, performance and prowess.

Gathering Existing Data

There are multiple ways to gather data on developers to use it for measuring software engineers productivity. Some of them are taking a look at the code they write, taking readings on their well being from different sensors(might use heart rate bracelets to check on peoples stress levels) and applying AI algorithms to developer behaviours. There exists an abundance of data lying around about the software engineers, which can be easily gathered by using different means.

Code Churn

One way to gather data is by using a method called code churn. Code churn shows the amount of code that was modified in a certain period of time. The way to measure Code Churns is to use Lines of Code. When a churn spike happens, it can indicate that there might be an issue that has risen and it hasn't been dealt with or took a long time to get it fixed, as the developer made little progress on a feature even though they put a lot of effort and time into it. This can indicate to higher-ups, like senior engineers to help the software engineer to get through the part they are struggling with. Code churn can also identify if there are problems with the team, as it might take more time for people to understand other peoples codes before doing their part, this would show that there might be communication problems within the team.

Version control systems

Another way to gather information about software engineers would be, using version control systems to gather publicly available information. Such API's like GitHub API and BitBucket API, allow us to gather publicly available information on any user as long as we have their username and also gather information about public repositories. A lot of programming languages have their versions of these API's making it easy to gather and process information in many different languages. Using all these API's allows gathering various pieces of information, like user's repositories, location, commits, pull requests, forks done in repositories, etc. Processing all this data can give insight into, what the developer is interested in, how often he commits, is he popular and use this data to measure if he is a good software engineer.

Behavioural Information

AI's and sensors might be employed to gather information on the behaviour and wellbeing of the software engineer. Sensors like heartbeat monitors in wristbands and wristwatches can help us gather information on the stress levels of the employees, determining which part of their work stresses them out the most. By gathering this information we can, try to reduce stress levels of the employees making them work more efficiently and leisurely. AI's can be used to monitor the usual behaviours of the employees, such as how often they take breaks, when is their peak of concentration and what type of code they are most comfortable with. All these can be used to increase the productivity of software engineers and redirect them to work in the fields they are most interested in and most comfortable in.

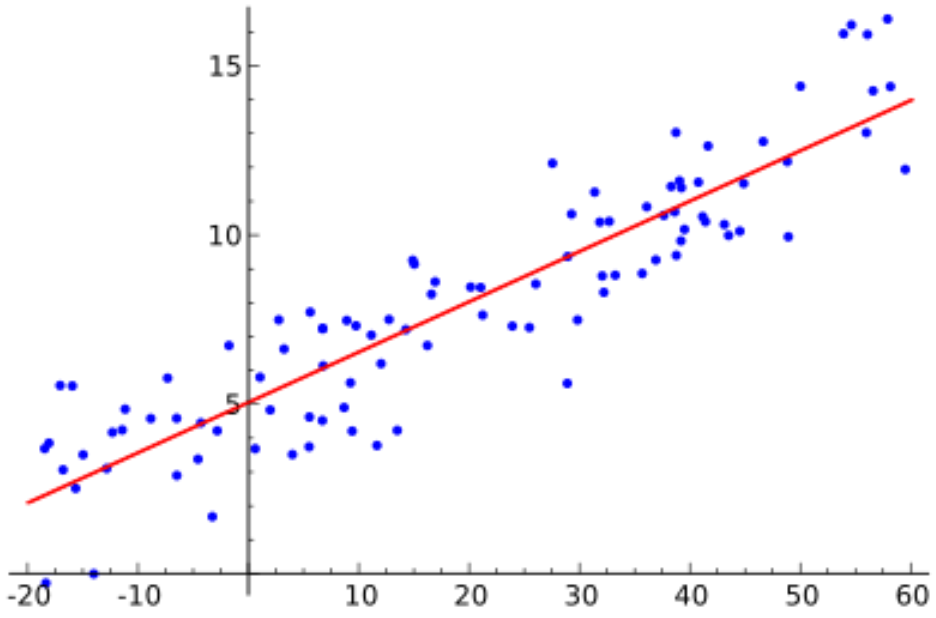
Methods and Algorithms

We talked about gathering data and measuring the software engineering that allows us to do necessary calculations to get measure software engineering correctly. So now we will talk about methods and algorithms which allow us to do so.

Simple Calculations

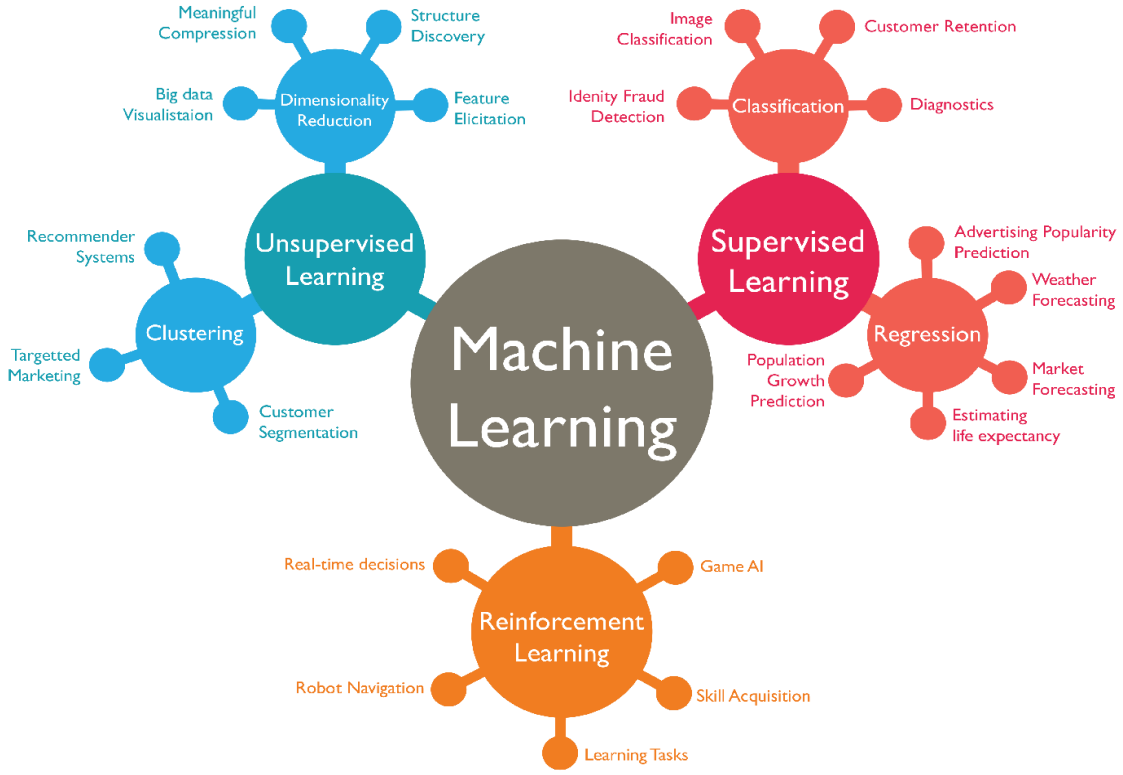
Simple calculations involve all the counting methods that don't require too complex calculations, which would require strong computing power or machine learning methods to compute.

One such method would be Simple Counting, as its name implies, it is just counting. This method is often used to count the number of line changes, commits over some time, repositories, etc. All these examples use Simple Counting to gather a lot of useful information which can be used when measuring software engineering. While they are straightforward, they are the basic building blocks for more complex methods and algorithms such as machine learning. An algorithm that would fall under simple calculations would be something like linear regression. Linear regression algorithms can make linear regression models, that can accurately predict and deduct useful information. One such example would be a model of commits done by a software engineer and the time which the engineer took to complete the project.



Machine learning

Machine learning refers to the wide range of techniques and algorithms that can adapt and improve their results automatically through data, i.e. “experience”. Machine learning methods are statistical and try to imitate the way the human brain processes information and “learns”. Machine learning aims to find general patterns in datasets that can be used to predict results in unseen datasets, this way a completely new set of data can be analysed. This is often used, where it is difficult to create a conventional algorithm to perform the tas needed.



An example of an algorithm that is used in machine learning would be logistic regression, it is used to find the likelihood of success or failure of a given event. The simplest example of this would be predicting if a developer will have a readme file or not. This is done by the algorithm using machine learning methods to learn from a dataset about the behaviour of a software engineer. This is one of the simplest algorithms used in machine learning and it is often used for binary and linear classification problems. Machine learning is specially used to find patterns in the data and uncover patterns, that we can take action for. It is very effective on huge datasets, as it is hard for a human to go through that much information and find correlations between them. But it is still run by algorithms made by humans so the algorithms created should be flexible and adaptable to different datasets, that can occur

while trying to measure software engineers as every person is different and it's hard to correlate every software engineer or a team.

Ethical issues

Ethical concerns arise every time it involves collecting data and trying to predict human behaviours using AI. The ethical issues that arise are especially more complex when software engineering is measured, as software engineers are the workers of the big corporations, so they have no say in their data and information being collected and used for research. New regulations and new laws are being put in place to protect our privacy. But with technology advancing so fast it's hard to account for the rate at which data analysing and collecting is becoming advanced.

One of the biggest ethical issues is the issue of consent. As often software engineers and other workers of big companies, have their information taken without any form of consent. Luckily in European Union, there is the GDPR(General Data Protection Regulations) legislation, which limits the personal information being extracted and analysed. But this doesn't apply in a corporate setting for software engineers, as the code they write is the asset of the company they work at. This is somewhat understandable, but it would be beneficial if corporations would show how they use and analyse the data they have collected and how they measure software engineers productivity. It is true that if a software engineer learns about the criteria he is being assessed with he can try to game it. But I am sure this can be easily avoided if complex calculations including a lot of criteria are performed.

Conclusion

All in all, in this report I tried to examine and show how software engineering can be measured, what data already exists that can be gathered, what methods and algorithms exist to do so and what are the ethical concerns that come with trying to measure software engineering. In the end, we can see that there is no special technology that can magically measure software engineers performance, only by using a combination of technologies can this be done. Also, everyone's privacy should be respected and the data collected should not harm the software engineers in any way, even if the technology advances at a fast pace.

References

- https://codescene.io/docs/guides/technical/code-churn.html
- https://nix-united.com/blog/software-development-life-cycle-nix-approach-to-sdlc/
- https://www.getclockwise.com/blog/measure-productivity-development
- https://www.softermii.com/blog/top-9-software-development-metrics-for-measuring-productivity-and-products-quality
- https://en.wikipedia.org/wiki/Linear_regression
- https://en.wikipedia.org/wiki/Machine_learning
- https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-1-d0368d769a7b
- https://en.wikipedia.org/wiki/General_Data_Protection_Regulation
- https://www.youtube.com/watch?v=Dp5_1QPLps0
- https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf
- https://phoenixnap.com/blog/software-development-life-cycle#:~:text=Software%20Development%20Life%20Cycle%20is,%2C%20Test%2C%20Deploy%2C%20Maintain.&text=SDLC%20is%20a%20way%20to%20measure%20and%20improve%20the%20development%20process.
- https://www.javatpoint.com/software-engineering-software-metrics#:~:text=A%20software%20metric%20is%20a,productivity%2C%20and%20many%20other%20uses.
- https://stackify.com/track-software-metrics/
- https://www.geeksforgeeks.org/software-measurement-and-metrics/
- https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/
- https://gdpr-info.eu/
- https://en.wikipedia.org/wiki/Logistic_regression
- https://www.pluralsight.com/blog/tutorials/code-churn#:~:text=Code%20Churn%20is%20when%20a,(typically%20within%20three%20weeks).