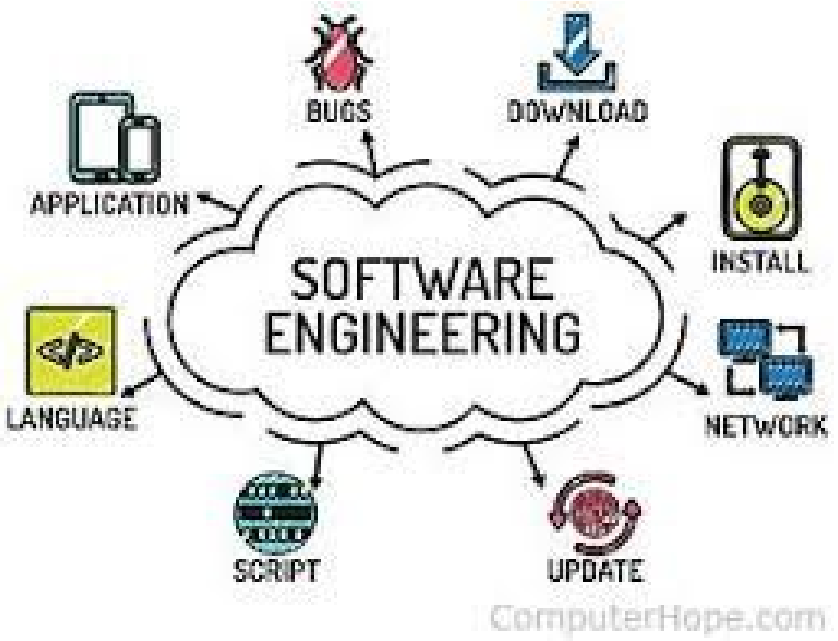
Measuring Software Engineering



What is Software Engineering, how do we measure it and what is the future of Software Engineering?

Conor O'Neill
18300645

What is Software Engineering?

Software is more than just program code. Program code is an executable code, which serves a computational purpose. Software is considered to be a collection of such code, associated libraries and documentations. Software, when required for a specific purpose, is called a Software Product. Engineering on the other hand, is the development of such products, using well designed scientific principles and methods. Fritz Bauer, a German computer scientist, defines software engineering as "the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines". Software engineering is not an easy or quick process. There are many stages to the development of a Software Product, and the progression through said stages is known as Software Evolution. Evolution begins in the requirement stage, where developers contact the client to gather requirements for the product. After, the developers create a prototype to show the client to get their feedback during the early stages. Throughout development, the client and developers keep close contact as the client may suggest changes in which several consecutive updates and maintenance keep changing too. Even after the desired product is in hand, the advancing technology and requirements force the software product to change accordingly. This extremely volatile process is called Agile Development, where constant change and communication between the customer and the development team takes place. There are many other development methods used by software developers, Agile being one of the most popular. Now that we know what Software Engineering is, we must look at how we can measure it. We

will look at measurable data, a variety of algorithmic approaches and consider the ethics concerns with this kind of Analysis.

How do we measure Software Engineering?

There are a number of ways in which software engineering can be measured. Focusing on the more general measurable data points, there are seemingly endless possibilities. The productivity of a Software Engineer can be monitored, as well as time spent coding, number of commits to a certain project, the depth of those commits and even how long an engineer spends at their desk at any given time. Measurable data can really be anything and we will find that every software engineer leaves behind a data footprint in the tools they use. The next question that arises is *how* can we measure these tendencies? What frameworks are available for the collection and processing of this data?

There are many different technologies in the world today that are used to collect data from employees. One of which is displayed in the company *Steelcase*. This advanced office chair measures posture, sitting habits, stress level, heart rate and even breathing as the employee works throughout the day. The results of the collected data are displayed on an iPad which coaches the user to improve their posture, gives them stretching and breathing exercises, and reminds them to take standing breaks throughout the day. This is an example of a framework that collects data and relays it to improve efficiency in the workplace. Similarly, *Timeular* have introduced an

8-sided tracking dice that allows the user to turn the dice to a side corresponding to what they're working on. If it is an email, the user will turn it to the email side and the dice will keep track of the time spent on that side. At the end of the day, the dice will collect all of the data and report it to a server where the user can visualize the results, and make changes to their routine to be more efficient. Similarly to the Steelcase chair, this method of data collection revolves around efficiency and self improvement. However, these are still very general forms of measuring data. Let's dig a little deeper into the intricacies of measuring Software Engineering.

A software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for various reasons; a few include measuring software performance, planning work items and measuring productivity. The company *Pluralsight* developed a Git management system called *Flow* (used to be known as Git Prime) tracks the productivity of a Software Engineer for a certain project. For example, by looking at how many commits they've published, how big those commits are, how deep across the code base those commits go, and so on. This software metric can be very easily implemented, as it simply counts the number of times the user does something, and relays that information to the server where the data is stored and visualized. This is a simple example of tracking and monitoring a Software Engineer and the work they do. Another service that is measured against the Github API is *Code Climate,* which has an automated code review tool that saves time and improves the maintainability of the engineer's codebases. Managers can use such services to identify, prioritize, track and communicate any issues to foster better team productivity. This enables effective

management and allows assessment and prioritization of problems within software development projects. The sooner managers can detect software problems, the easier and less-expensive the troubleshooting process. We know now that we can simplify the development process by using software metrics to track and analyze progress, but what algorithmic approaches can be used to track said progress and to what extent are these metrics useful?

It is very important to focus on the metrics that help deliver useful software to customers. Often, sets of metrics are communicated to software teams as goals. Focusing on these targets help developers improve practicality of the software and user experience. It is important to know which metrics to focus on, as it is equally important to know which metrics are counterproductive. One way to distinguish between the two is to stop using metrics that don't lead to change. Different metrics provide development teams with different values depending on the goals of said teams; Thus, there is no standard or definition of metrics that give value to any given team. As I mentioned earlier, Agile development being one of the most popular methodologies, has many quantifiable metrics.

One example being Lead Time, which quantifies how long it takes for ideas to be developed and delivered as software. Lowering lead time is a way to improve how responsive software developers are to customers. Cycle time is another imperative metric, as it measures the time it takes to change the software system and implement that change in production. Team velocity measures the amount of work completed within a given time. It is important to recognize that velocity is "team-unique" and

thinking of this measure as a parameter in an estimation model is a mistake. The team must establish its own velocity for the work at hand. Mean time between failures (MTBF) and mean time to recover/repair (MTTR). Both metrics measure how the software performs in the production environment. Since software failures are almost unavoidable, these software metrics attempt to quantify how well the software recovers and preserves data. Similarly, Application Crash Rate (ACR) is calculated by dividing how many times an application fails (F), and how many times it is used (U). ACR = F/U. The lower the ACR, the better the code. These are only a few of the numerous metrics used by professional software developers to track and optimize code bases within deliverable software.

What is the future of measuring software engineering?

As we have discussed, there are various forms of collecting such data metrics, but what does the future offer for such data collection? Are there better, more efficient ways to measure the productivity of a software development team? Let's take a look into the future at what kind of technologies are emerging.

There is a big question surrounding the topic of using AI to measure developers' performance and make HR-related decisions. AI is being deployed as a way to intelligently forecast risk and measure the development performance of software organizations. This new method of using AI can give software companies a new level of understanding of their software delivery pipeline's performance. But the

question remains, is this ethical and to what extent should AI be implemented into these development teams?

Imagine a platform, using artificial intelligence, that would help organizations assign development teams to specific projects by using extremely accurate and thorough data that would indicate how well they would complete that task. That would be of huge value to any company or business. Obviously this comes with many ethical concerns and problems that would need to be dealt with delicately. Just as I mentioned earlier, by choosing metrics that are not easily manipulated and that are relevant to the development processes, one can remove much of the ambiguity that would cause concern about using machine-based measurement. Another concern that arises from the use of AI-driven insights stems from the legal challenges that a company may face. The concept of "the computer said to do it, so i did it" might not stand in court in front of a judge. This goes back to human understanding, and that the judge most likely does not know how AI works. But to overcome this, the best use case for this sort of intelligence is not to make employment decisions, but rather to make resource decisions. By leveraging AI to understand which teams and individuals are the strongest at a given task or type of project, you can better understand not only how to assign projects, but where your organization has weaknesses and where your team could benefit from more training as well.

Now that we know different ways in which software engineering can be measured and the different methodologies used to collect data, we are left with one final question: Is all of this ethical?

Is this Ethical?

Due to their unique nature, AI based systems influence a wide range of stakeholders with or without their consent, and thus the development of these systems necessitates a higher degree of ethical consideration than is currently carried out in most cases. But where is the line that constitutes a proper degree of ethical consideration? How much data should companies allow for machine-based instruments to access? How much access is too much or too little? These are all questions being asked today as this is an ongoing debate within the world of Software Engineering.

Personally, I think it depends on the nature of the task at hand. Some Software developments may need specific access to certain data to produce accurate results. The more data the system has access to, the more accurate those results will be. In the case of Agile development, when Software Development teams are being measured for productivity and efficiency, personal data files of employees aren't necessarily required. But when a machine-based software system needs access to personal data of employees, the issue of data privacy emerges. Data privacy is a very prominent issue in the world of technology, and with the emergence of new technologies at a seemingly exponential rate, it becomes even more important to keep customer's data secure. Thousands of companies today have access to millions of individuals personal data, and any sort of data leak, leaves millions of people's personal information at risk

and prone to corruption. This is why ethics in software engineering are imperative to consider.

So what? We looked at what Software Engineering is, how it can be measured and the different approaches of actually collecting and analyzing the data to improve the efficiency of a software development team. Additionally, we asked the question surrounding the ethics of all of this. I do believe that the movement to AI-based measuring is beneficial in many ways. It can create extremely accurate reflections of data to improve the selection of software developers for a specific task, it can also help managers of a development team locate the weak points of a company, and outline the areas that need to be improved upon. Although there is an ethical concern with machine-based measurements, I think with the correct data protection services in place, and using AI as a resource as opposed to relying on it to make employment decisions, one can add great value and efficiency to a company's software developments.

References

https://www.tutorialspoint.com/software_engineering/software_engineering_overview.htm

https://www.youtube.com/watch?v=Dp5_1QPLps0

https://stackify.com/track-software-metrics/

https://codeclimate.com/quality/

https://insights.sei.cmu.edu/sei_blog/2014/09/agile-metrics-seven-categories.html

https://techbeacon.com/devops/should-you-use-ai-make-decisions-about-your-software-team

https://ethics.acm.org/code-of-ethics/software-engineering-code/