Measuring Software Engineering

Table of Contents:

1.) Introduction

Software engineering is a popular career path in today's society, yet not many understand the roots of this field. So before getting into the main topics of this report, I believe a quick introduction to the field would be helpful. Software engineering is a new engineering discipline which focuses on all aspects relating to software development. This includes concepts, principles, theories and tools used to develop professional software. It's described as new as software engineering was only recognized as a discipline rather than an art or craft in the late 1960s. Software engineering is actually a separate field from computer science, which some people don't realize due to how interlinked software development has become with other fields.

However, with its rapid growth, and increased employment in the sector, many major companies have begun to evaluate certain factors which can have an impact of the development of software. That is what will be discussed in this report.

2.1) Measuring Software Engineering

One common task employers like to carry out is this idea of measuring software engineering, But this is unlike finding out how long someone has spent reading a book, or playing a particular game. In these scenarios, simply getting the time spent would suffice for a result. With software engineering however, this isn't the case, and so things become a little more complicated. How does one go about measuring a more abstract concept like the amount of software engineering carried out by an individual?

A common method regularly considered is simply taking a look at the programs developed by a person, and taking note of its size, in terms of space and lines written. However, I believe this process is not an ideal methodology and is actually quite a flawed way to look at software engineering. If one just takes quantity of code into account when trying to measure software engineering, a programmer can very easily just end up writing a bloated program. These programs can contain a lot of "dead code", which is essentially just sections of code that can be reached and executed, but the result of that section of code is never used in the rest of a program. Either it wasn't relevant to the rest of the codebase, or it adds nothing to the program. This just ends up wasting CPU performance and actually hinders further development, as introducing more code with no functionality increases the difficulty of debugging when an error occurs. This is due to both a larger program to search, and the fact that dead

code doesn’t contribute to the rest of the program, leading to an elevated challenge in tracing the error. This is also why I think measuring software engineering via bug fixes is also a futile effort, since someone can just willingly write faulty code, and just pace through it and fix the errors with ease.

An often used alternative is evaluating the result of tests, if present in the code. These have the benefit of actually running sections of the code written to see if it works as intended. This, therefore, already adds more value to a result gathered from this over a result gathered from checking the amount of lines of code written. However, this doesn’t guarantee that bugs do not exist in the program. Therefore, I’m of the opinion that tests on their own would not be enough as a metric.

This leads us to a very popular option regularly seen today, which is browsing repositories made by a particular individual on certain version control software such as GitHub. This is a method many employers use for many various reasons. However, one of the main reasons is that one can look at a user’s commit history. This, while allowing someone to see how often one works on a certain piece of code, also has the benefit of enabling them to witness what particular section of code was worked on, what changes were introduced to the program and gives an overall progress of the code. This gives a very accurate view of code development over the lifetime of a program and can give insight to a user’s workflow when developing a program.

This gives rise to an even better measure of software engineering when paired with code coverage. Code coverage basically runs code and returns what percentage of the code was actually executed. In my opinion, this would be one of the better forms of measuring software engineering as not only does using these tools help with the development of software, but with the monitoring capabilities of these tools, one can effectively remove both dead code and bugs, as well as ensure no unreachable code exists in the program, which leads to actual meaningful improvements in one’s code.

2.2) Relevant Platforms

Over the course of one’s programming journey, multiple platforms will likely be visited, whether it’s looking for inspiration for a project, searching for potential bug fixes, or simply hoping to obtain information previously unknown to them. Many of these sites will also benefit someone looking for software engineering related topics, whether they be online repositories or forum spaces.

```
function eventInfo(username){
    const reqEvents = new XMLHttpRequest();
    var pushEvents = 0;
    var eventType = 'PushEvent';
    const urlEvents = `https://api.github.com/users/${username}/events?per_page=100`;
    reqEvents.open('GET', urlEvents, true);

    reqEvents.onload = function(){
        const eventData = JSON.parse(this.response);
        console.log(eventData);

        for (let i in eventData){
            if (eventData[i].type == eventType){
                pushEvents++;
            }
        }
        console.log('Pushes: ', pushEvents);
    }

    reqEvents.send();
}
eventInfo('torvalds');
```

Fig 1: An example of a GitHub API request. This request filters the amount push events in the last 100 events of a user.

GitHub is an online software I'm sure many of us are familiar with. This website allows users to create online repositories, where one can upload code on a main branch. Many new branches can be made, where you write new code to, and then you have the option to merge this branch with your main branch to basically update your code without having to worry about breaking previously working code, since you'll always being writing new code into a new branch prior to committing changes to the main branch. However, GitHub also allows for the gathering of this information. With knowledge of the GitHub API, one can collect the amount of repositories a user has, commit history, the amount of other users that follows this user and the amount of users this person follows, among other interesting pieces of trivia. This data can then be stored locally in a database, as a JSON object, etc. and used in calculations. With proper usage, this can allow someone to gain unique information such as the average amount of commits per certain time period, the number of followers earned in a specific length of time. As long as you can make sense of the data being returned, you can form any custom queries you desire.

Another webpage many would know is Stack Overflow. Stack Overflow is a collaborative question and answer website for programmers. If an individual is unsure about a certain piece of code, unsure how to implement specific

functionality, or practicing the writing of code, this is a webpage many would consider consulting first. With the Stack Exchange API, it is then possible to gather information relevant to this site such as filtering comments by ids etc. and by extension, you can make personalized queries such as the most active posts, a user's most recent contributions to the site whether it's a comment or a question.

The entry level for making these requests are low as an additional bonus. Queries are HTTP and URL based, with the response being in JSON object. This makes extracting the exact data you want simple, as long as you know the data attribute's name in the JSON object.

2.3) Possible Computations

Having the data of various software engineers is a nice first step, but without performing any computations, the collected data ends up being of less use to people. Different calculations performed over multiple datasets adds value that simply displaying unaltered collected data lacks. Whether someone wants to compare different software developers, measure one's efficiency or simply gather average amount of code written in a given time period, performing computations on the collected dataset is often what transforms a bundle of data to actual meaningful results that can be interpreted by even the general public. With that said, how does one choose how to do their computations?

A basic method for doing computation over datasets is simply counting. I can already think of several issues where counting alone will not suffice and one of those area where issues arise is scale. While counting manually can be comfortably carried out for small sample size, with increased scale the shortcomings of this method becomes more and more apparent. One of its shortcoming is that the time required to perform this increases, and over a certain size it becomes highly infeasible and/or largely impractical to continue with this. Another issue is potential for error. As scale increases, so does the room for potential mistakes in this method. Due to the fact that as sample size increases, so does the accuracy of the dataset, people tend to work with larger sample sizes rather than smaller sizes. As the two errors I've mentioned become more and more of a problem in larger datasets, this therefore renders them highly impractical for this use.

This leads us to another way of handling this data, which is via algorithms. Measuring the complexity of a piece of code has become a major study and different notations such as Big O, Big Theta and Big Omega have become

common phrases when discussing algorithms. However, the actual use of algorithms helps with the computation of useful data when related to software engineering. Stacks and queues are two such example. Queues, as a starter, tend to have helper functions so you don't even have to redefine the same function multiple time. Queue.size() can help easily get the size of the dataset being worked on, which already makes counting even less desirable. With stacks, Stack.size() serves a similar purpose. However, the top() function can return the value stored at the top of the stack. By popping the top value and calling this function in a loop, you can access all values present in the dataset.

Another popular method that has recently emerged in the field is the use of various machine learning techniques. Allowing a computer program to naturally see problems as humans do gives rise to many advantages. Various different machine learning approaches can be implemented as a result of increased research in this sub-field. Some examples include, but are not limited to:

a.) **Supervised Learning**

This is a tool that is founded on the concept of example-based learning. This approach to data analysis involves filling a complex algorithm with mined data. This method of machine learning allows algorithms to filter large quantities of data, and can even allow for interconnection among datasets.

b.) Unsupervised Learning

This is where the algorithm is left alone to detect any structure in its input. This has an advantage in detecting hidden data patterns or avoiding any bias that may be present in the dataset.

c.) Deep Learning

Deep learning streamlines tasks so that algorithms can perform image and voice recognition, among other things such as making insightful decisions.

d.) Reinforcement Learning

This involves a program interacting with a self-motivated setting to complete a task.

Knowing the correct tool to use can result in the gathering of relevant data that can be free of human bias, in a process that can be automated and therefore, very error-proof for a large sample size range.

2.4) Ethics

With the amount of data we can now have access to, the wealth of information we can now produce has also increased exponentially. However, this has led to an understandable increase in ethical concerns over the processing of data. While some may think processing code that someone has written shouldn't be an issue as it isn't personal data, I do believe that what those people fail to understand is that it isn't exactly the gathering of what has been written that's the issue. It's the ability to access how one writes their code. This is a point one has to take into consideration, since certain programmers write their code in slightly unique ways. One might write variables in lowerCamelCase, some may choose to declare static final variables in all uppercase, etc. Knowing how someone writes their code gives someone access to identify their other works on different projects. And to some, this isn't a pleasant thought that someone that just essentially reference every piece of code they have ever written.

Of course, there are other aspects that people don't enjoy when it comes to accessing information. An example can be seen in accessing the GitHub API. Certain requests can reveal the email address of users, such as members of an organization. Being able to access someone's email wouldn't be such a big issue if data breaches were not a threat. As seen in the live-streaming site Twitch in 2021, a massive data breach resulted in over 100GBs of private data being leaked publicly online on October 6th. Among the data leaked was user data, as well as other information such as payment information for the past three years, security tools and more. The main issue is that if someone had a Twitch account and also had a GitHub account in an organization, someone would have access to the necessary email and password to access their account. This is a major security risk that many people would never want to experience, especially if the individual is someone that uses the same password across multiple websites, accounts etc. While many users were smart and changed their password as soon as this leak occurred, we can't say for sure that everyone carried out this task. While I don't personally understand why someone wouldn't complete this task, I can relate to the feeling of having been compromised due to the lack of security of personal data, and the allowance of this data to be leaked online for the majority of the world to see.

However, I can also see the other side and their view that being able to access certain information can yield increased benefits without risk of data loss. Taking account of people's health, physical and mental, and graphing that against their productivity can give rise to impactful insight as to how to increase workflow without the obvious choice of adding more manpower. Certain studies have looked into how a workplace's mental state, and whether they are happy or not, can influence the productivity of said workspace. And the information collected

for this study doesn't pose any risk even if exposed. The main information gathered for the study included physical activity, duration of said activity, and volume of sales, none of which would result in a loss of privacy if leaked. And the data from this study had an effect on how workplaces managed productivity when the result showed that working spaces that were happier also had higher productivity values. This then has the effect of promoting friendlier communication within a working area, and also ensuring the mental health of workers present.

The only issue I can see with measuring a worker's productivity is if it's constantly being monitored and not anonymous. That can be a breach of data security, especially since it becomes a simple task for an employer to just fire an employee if their productivity rate falls below a certain value. But I can also see that having the inverse effect of plummeting workers' happiness to an all time low, due to increased stress of having to maintain an unspecific productivity, which can already be seen as an abstract term. While the average amount of employees may remain unchanged, the amount of long term employees could definitely see a decline if this certain style of management was implemented. Also, I can also just hope that employers don't abuse the system by accessing data that is unnecessary, such as medical conditions, martial status etc., as this gives an employer way too much power over their employees, and can lead to problems such as forcing workers to take days off if they had a history of a medical condition, or unwanted approaches in a professional, working environment.

3) Conclusion

All in all, this talk of measuring software engineering can be a very delicate subject matter, as many need to first realize that there is a difference between software engineering and other fields such as computer science. While I have brought up many points about the subject of objectively measuring software engineering, how one can go about this and different tools one can use to conduct their own research and construct their own dataset, it's highly important to note that this report only goes so far. By carrying further research on the general theme of software engineering and all that it entails, someone can come to their own conclusion about what software engineering truly is. While many points brought up by multiple people will be similar without doubt, to come to an exact agreed upon definition is improbable and we'll have to agree to disagree. However, perhaps it is through this difference that software engineers come together and push the field even further beyond. Differences can breed

innovation, and software engineering and the fields it has integrated itself into has shown to lead to some very innovative end points indeed.

References:

Machine Learning Approaches to Data Analysis | by FuseMap | FuseMap | Medium

Measuring Happiness Using Wearable Technology (hitachi.com)

(PDF) Network Effects on Worker Productivity (researchgate.net)

An Overview of the 2021 Twitch Live Streaming Data Breach (mitnicksecurity.com)