Measuring Engineering
Michael McKay – 16324528

What is a software engineer? A software engineer is often confused with a general programmer, but the two are quite different. A programmer creates the code that makes a program run, whereas a software engineer is responsible for designing, developing and implementing the software solutions these programmers create.

I am writing an essay for the CS3012 module Software Engineering. The essay is on how the software engineering process can be measured and assessed in a number of different ways. There are of course many different ways in which software engineering can be analysed but a few of the key ways which I will be discussing in this essay, are in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethics concerns surrounding this kind of analytics.

I found this module interesting in a number of different ways and hope that in turn I can make this essay as easy to read and interesting as possible.

Measurable Data

This is the first topic I will be discussing. There are of course many ways in which data can be measured but I will just be discussing the most widely used methods. One of the most commonly used metrics in lines of code (LOC) or Source lines of code (SLOC).

SLOC is used to (no surprise) accurately count the number of lines of code in the text of the programs source code. This in turn helps to measure the size of the computer program. This was one of the

first metrics used to analyse code. This is often associated with how much effort one puts into a program, as in the more lines of code, the more effort put in. But of course, this is not always the case, some programs can be written in a number of different ways. For example, if I was to write a program to computer a person's BMI, I'm sure it could be done in ten lines of code or fifty. The program with the less lines of code is most likely more efficient, but if we were to judge the programmer off of the count of his SLOC, we would think of him as lazier or not as committed as the programmer who developed the same program but five times as long. SLOC is an interesting method of measuring data for sure, but can also be slightly useless.

Commits are used to make a set of changes permanent in a program, or to add the latest changes to part of the source code of a particular project. The most popular instance where commits would be used is through Git. GitHub is a web-based hosting service for version control using Git, and I will be discussing it here, even though there are other variations of GitHub. GitHub comes with different functions, like the ability to share repositories with other GitHub users, note changes made to files in the repository and to see the commit history. Commit history is quite big in today's world for developers and engineers. Even in this module, we are told to commit regularly in order to track our consistency. This is very useful to large companies with many developers and engineers. The commit history gives the person in charge of the project an idea of which of their developers are working consistently on this project. Of course, the commits need to actually be useful and not just someone changing the README slightly!

Another useful way of measuring data is **Code Coverage.** Code coverage is basically the percentage of code which is covered by automated unit tests. It basically determines which statements in a

body of code have been executed and which statements have not. This is a great way of ensuring that software has no errors or bugs. Unit tests are great but can also be a nuisance. For example, some companies can have programs with thousands of lines of code, it is very difficult to achieve 100% code coverage for such large programs. Trying to achieve this code coverage can be costly for companies and certainly take time, which would in turn affect the agenda of the company. The other issue with code coverage, is that they do not show how efficient a program is. The program may run well but is it of the best quality? Overall, code coverage is a solid way of measuring data.

Bugs and Errors is in my opinion quite a good way of measuring data. This is quite a good approach because it really gives an idea of the performance and efficiency of a program. It goes without saying that the less bugs you have the better. The problem with this though is that it is not the best way of measuring the programmer. One programmer could be working on a very simple task with next to no errors and his colleague on the more complex part of the project. The fact that the code that the programmer working on the easier part of the project has developed, has less errors in it doesn't mean he or she is a better programmer or uses their time better.

The **Performance** of a program is also very important. It is measurable and reflects the efficiency of the program and of the developer. It takes into account the execution time, the storage required and more. Some programs are more efficient than others, there's no denying this. For example, if I were to write a program that's purpose was to convert binary numbers into roman numerals, and my friend was also to write the same program, but his ran twice as fast and used far less memory, my friend's program would be the more efficient program. This can be useful

to show which one of two employees are performing better, if of course the employees are both working on the same task.

Overview of Computational Platforms Available & Algorithmic Approaches Available

I have decided to merge these two talking points together just to make the overall essay easier to read and to avoid repetition between these two points.

There are many different computational platforms and algorithms that are used today which assist in the measuring of software engineering. We all know that an algorithm is basically a set of rules to follow in calculations or problem-solving operations, but computational platforms sounds a little scarier than its definition. It is simply the environment in which we execute different software. It could be the operating system, the hardware or maybe even a web browser. Computational platforms analyse and cross-examine different metrics which these algorithms have obtained.

Our lecture discussed with us the process known as the Personal Software Process or PSP. The definition of PSP is a structured software development process that is intended to help software engineers better understand and improve their performance by tracking their predicted and actual development of code.

The PSP helps software engineers to:

1. Improve their estimating and planning skills i.e. make them more organised in their work
2. Make commitments they are able to keep and achieve in good time

3. Manage the quality of their projects which could in turn benefit the entire group project if they were involved in a group project
4. Reduce the number or defects and errors in the work

PSP claims to give software engineers the process skills necessary to work on a team software process (TSP) team. TSP provides a defined operational process framework, this framework is designed to helps big teams of managers, engineers and others organise projects and produce software projects.

TSP is quite great because it is useful for small projects that of several thousand lines of code and for very large projects, which may contain over half a million lines of code. It is intended to improve the levels of productivity and quality of a team's software development project.

Before engineers participate in TSP, they need to have already learnt about PSP. This is important in order for the TSP to work effectively. The first step in the TSP cycle is called the launch, the goal of the launch is to get the team building process up and running. During this launch, all the members of the team which includes managers and anyone working for said managers, do a few things. They establish goals, define team roles, assess risks, allocate tasks, estimate effort and to produce a general team plan or agenda. After this launch phase, there is the execution phase. This phase is basically where the effort of the team is tracked, and consistent checks are made to see if the team is on schedule. The process is ended with a Post Mortem or autopsy of sorts. This is where the overall performance is assessed, in order to improve the performance for the next time.

Ethics

There are a lot of different opinions on whether or not software engineering should be measured in general. Many believe it is a good idea and that it leads to better efficiency and others completely disagree, they believe time can be better spent. I will discuss the different opinions and ethics below.

Many people believe that measuring software engineering/development is very useful. They believe it leads to more efficient programs, good for comparison of programmers and leads to projects meeting deadlines. I would guess that the people who are all-for measuring software engineering do barely any programming themselves and are managers/CTOs/CEOs. They can't possibly understand the pressure their employees are under.

Many programmers have come forward saying that they feel a sense of pressure due to the constant measuring and comparison of their work to others. I can't even imagine the pressure these programmers are under, I even feel pressure in regard to this module and others. A sense of nervousness when I go to show demonstrators problems I am having in various assignments. I know many other students who also feel pressure and have for the last two and a half years. Of course, a lot of that pressure is down to human nature and wanting to impress but having ourselves consistently measured in a number of ways and compared to our friends and other students certainly does not help. That's why I can't imagine the pressure of an actual work environment, a few modules in a computer science course in Trinity can definitely not compare.

I believe many programmers do not like the idea of their work being measured because it could potentially display to "the world" that they are only average at programming. To some this would be

considered a mortal sin, imagine not being the best at programming! I know I am not the best programmer, so do many of my friends but I personally don't really mind too much, but I could understand how for some where programming is effectively their life, they would not want to be considered as average.

Many believe that if anything is to be measured it should be the programmers happiness/job satisfaction over their productivity. This is because programmers and developers generally get more work done and better work done when they are happy with what they are doing. This is rather obvious if you ask me. I loved studying Spanish for example back in secondary school, and hated economics. Subconsciously, we all generally would work more towards something that interests us that something that doesn't. That is why I got a B in Spanish and a D in economics. This is just a small example of how even in the workplace, a programmer is much more likely to be efficient if they don't have to consistently worry about whether or not they have committed enough recently to GitHub or that their code coverage is flawless. Of course, this would be rather difficult to measure. How can one measure a person's happiness, there's not exactly a scale we all can relate to?

If I were to choose, I would prefer the measuring of software engineering didn't exist. It is of course useful in some instances but the well-being of the programmer whether he or she is in college or works for Google, is more important in my opinion.

Conclusion

Measuring software engineering is a sensitive subject. I believe it is a good idea in some instances, it can lead to productivity and better efficiency. I also believe from different online sources and a tiny bit of personal experience that it can be time wasting and can lead to pressure and stress in developers. The way that the data is

measured is an important factor in software engineering and in the end is the difference between whether or not this measurement is useful or useless.

I used many different sources when gathering information for this essay. I used YouTube mainly for trying to gain a better understanding of PSP for example and the different ways of measuring data. On the next page I will list out the different sources I used.

Sources:

- TSP : https://en.wikipedia.org/wiki/Team_software_process
- PSP: https://en.wikipedia.org/wiki/Personal_software_process
- General Measurable Data site https://medium.com/measurabledata/introduction-to-measurable-data-token-4-5-how-it-works-ab4c825fed1
- PSP: https://www.youtube.com/watch?v=h5CRzj3z75k
- Code Coverage: https://en.wikipedia.org/wiki/Code_coverage
- Different Metrics: https://diceus.com/top-7-software-quality-metrics-matter/
- https://www.scss.tcd.ie/Stephen.Barrett/teaching/CS3012/