CS3012 – Software Engineering
A Report on Measuring Software Engineering

Donal Hogan – 15324573

Introduction

This report is on my findings and thoughts on how the software engineering process could be measured. Measuring the software engineering process means gathering data throughout the process on different aspects. An analysis of this data can be used to judge the process and understand how and where the process can be improved.

I will begin by going through how to measure the engineering process in general and some processes that people may use to do this. Then I will go through the kinds of data that can be collected and measured when trying to judge if the process went well. I will then mention the algorithmic approaches to analysing the data and understanding it. Finally, I will talk about the ethics of doing so.

By the end of this report I will have gone through the entire process of measuring software engineering. A better understanding of each method should come with it and the knowledge of how and when to implement them. If we should implement them also as I will mention later in this report, it isn't always the best decision to use these methods.

Measuring Software Engineering

It is a good idea to measure software engineering, as it can help you refine and improve your process when working to become a better programmer. It also allows you to tell how good you are in certain areas and helps you use your resources better during the engineering process.

They are all personal reasons to do this however. There are also reasons to use it when working as part of a team. You can use a process to analyse each individuals contributions to the team and tell where you should be putting praise and rewards. It can also help you tell

when someone isn't pulling their weight in the team so you can deal with that before it becomes too much of a problem.

It is hard to accurately measure a software engineering process. There are some techniques that you can use however to get a good idea of how well a process has/is worked/working. Most engineers come up with something called a PSP (Personal Software Process) to measure their own process and ability. I talk about this in more detail later.

The reason it's so hard to accurately measure the engineering process is because what you want to measure in order to define it as successful can change depending on the project. There are different types of data, which I talk about later, that can be collected and used to measure the process. The data can also be unreliable and many other circumstances like sick days, mis-understandings and just having a bad day can affect the data and make it harder to accurately measure the engineering process.

Having a measurement of the engineering process also helps for other practical matters while writing the code. It can help with quality assurance, cost estimation, optimisation, testing and debugging. These all can be used to make the final product better and improve your skills as a software engineer.

Goal driven measurement

Most places use this kind of measurement at least to some extent. This method involves setting goals to be met during the process and taking the data that can be gathered from the work done on those goals in order to measure how successful the process is. The goals can be of any size though it helps to keep them at around the same size in order to maximise the accuracy of the measurement. Larger goals could be broken down into smaller ones to make this possible.

The goals picked to be measured should align with the business goals/ aims of the project in order to get the relative data. A general business model for example would have cost(money), time, functionality and quality which can all be turned into measurable data and used to tell if the process is good or not.

Code Coverage

This uses the amount of lines run to test the efficiency of your code. The more lines of code that are run the more efficient it is. It works well with any test cases you have to run the program with as you can tell which lines were run and which are unnecessary to your

solution. It makes it easier to find any issues such as inefficiency or bugs in your program also.

A drawback to using this method would be that it doesn't necessarily show how efficiently the code was run exactly. This is because the program could be written to be particularly long or slow but still have all of the lines of code be run.

Cohesion of code

This refers to how the program works or possibly overlaps with linked programs such as libraries. If the link is relative to what the programs does then it makes sense for it to be there. However, if not or if the program does something that is already done in another linked program then the code is much less efficient. This is a hard method to get a measurement for however. You would only be able to do it once you've fully looked through the rest of the linked programs and your own, fully understanding all of them. If you can do that though then this becomes a very good measurement for your software engineering process and it becomes easy to see possible fixes for it.

PSP – Personal Software Process

This is when you try to generate and analyse data yourself. Such as using information from projects that you worked on to try and measure the process. The specific data that you measure or the method you use isn't necessary for this method and it can be mixed with other methods, where you would use them to just measure your own data.

There are 2 possible issues with this process:

1. Worse results than expected.
2. Manipulation of results.

The first case would occur when after analysing your results in some way, they come out differently than you would have expected. Mainly that they said you were not as good as you thought. You may still be in a fair to good standing but even showing that you're not in the top tier of programmers can be disheartening to some people. This may mean that they may dis-regard the results and/or process and start again using different methods hoping for another outcome, defeating the purpose of an objective measurement of the engineering process.

The second case could even be a continuation of the first in some case or its own issue. Manipulation of the results when measuring a process can occur even without them being

biased consciously. They may subconsciously think that they would come off in a better light or the results would be closer to what they expected if certain things were measured more or brought more into focus for the spec and the results become changed without them fully realising. It can also happen consciously, such as mentioned in the first issue, when they may be unhappy with the results they're getting and change them purposefully to get what they want. This is probably the most common issue when it comes to the PSP method of measuring an engineering process.

Measurable data

There are multiple sources of data that you could measure in the engineering process. They may not always be precise or may be mis-leading at times due to circumstances that may have caused them to change unexpectedly or they may just not be necessary for the project that is being done. You should only record the data that you need for the measuring process you're using to avoid wasting resources. The kinds of data that you record, as I mentioned in the previous section, should align with the goals/aims of the project you're working on. You should also take care to be ethical with recording the data, which I will talk about more later in this report.

Here's a list of some things that could be measured in the process:

Time

This one is self-explanatory. You can easily measure the amount of time it takes for a task to be completed and compare it against other times. This may vary however due to the size of the task that's being worked on as smaller tasks would take less time, so consideration must be taken when comparing the times so only tasks of equal or comparable size are contrasted and considered when deciding if it's good or not.

You also need to be aware of the experience and expertise of the programmers that are working on the project. This can affect the time considerably as someone with prior knowledge in the area or with the tools used would have a much easier time with it than a beginner.

Lines of code

This can be a divisive way of measuring the process as depending on what you want to measure with this data it can change whether a larger amount would be better or a smaller amount.

A larger amount may be better if you're trying to measure productivity of someone/a team or when you want to see how much progress has been made on a task. This could be mis-leading however as more lines of code can mean inefficiency and lazy workers may try to get away with writing longer, worse code to seem more productive, while better programmers could be overlooked as they wrote their code more efficiently. It's also not a good indicator of how much progress has been made on a task as just because lines have been written doesn't mean that they're correct, it could all be out of scope or do something wrong and it would all need to be re-written. In that case little progress has been made despite the amount of lines seemingly indicating that it's much further along.

A smaller amount of lines may be better if you're trying to measure the efficiency of a programmer/piece of code. The smaller the code while still performing the task can often be considered better. The drawback to this however means that the code may become less understandable and hard to maintain in the future as clarifying parts would have to be left out to fit it in less lines of code. Another issue with this would be if the code was less efficient than longer pieces as it may have more calls to external procedures or use more memory to make up for the lines removed.

Code coverage

This as I mentioned earlier is the amount of lines in the program that actually are run. It is not only a method of measuring the engineering process but also a metric itself and can be used in other measurement techniques. You need to have test cases, especially with edge cases, in order to gather this data, but it can be very rewarding. Not only does it measure the efficiency of the program but can also help you spot errors within and help you fix them. This makes it not only a metric and a measurement, but also part of the process itself.

The drawback to using this method is the need to come up with test cases that cover every possible situation. It can be hard, especially for larger programs, to come up with every edge case and ensure it works as intended.

Aims met within a period

This is a relatively good way of measurement in my opinion, however like the others it does have its own drawbacks. This is when you set a certain period and check how many goals, be they features, programs or something else, are completed within that period.

This can be a good way of measuring the efficiency of how fast work is being done. It is used very commonly, even outside of software engineering and is generally considered a good standard. It makes it easier to spot someone who may not be putting in enough effort and can be used to tell if someone is doing particularly well.

The drawbacks to this is how it can become inaccurate if the aims are different sizes or require different amounts of effort. Larger aims that require more effort can't/shouldn't be contrasted against much smaller aims that need less effort to complete. That would throw off the results as you would expect more of the smaller aims to be completed within the period and less of the larger aims. This means that someone who put in more work could be considered as having done very little because they had larger aims. Conversely someone who put in just a little effort may be thought of as a much better worker since they got more of the smaller aims.

Cycle time

This is how long it takes you to make a change to your software system from the initial issue and deliver that change into production. The length of time this is expected to take can change from team to team, or even from project to project. In some cases, it can be measured in months or days while in others it can be done in minutes or even in seconds. This data can only be taken when the program is continuously deployed throughout its lifecycle, so it's only useful in certain circumstances. However, in the right cases it can be a good indicator of your process as it easily shows how goals have been met, so it works well with a goal driven measurement of the process.

A drawback to this could be that someone could realise that the more commits/changes they make to the program, even small changes, would get them more recognition. As such lazy workers could use this method to their advantage to look more productive than they really are.

Issues resolved

You can collect this data whenever a patch is needed for the project. This is another type of data that can be collected once your program/software has been released/being used, though it can be gathered during production. If gathered during production however, then you only get feedback from the tests you wrote yourself, so you may be missing some key information or edge case. In that way the data you get for this is generally better when you get it after it's been released/ is being used.

It's less about the number of patches that are made, mistakes are bound to be made during the development, and more about the general trend of the issues and resolutions. If a resolution is made and issued, it means that the process was bad if another resolution is needed for the same issue later.

Algorithmic approaches

This section is about different methods that someone could use to analyse the metrics that you have collected. It is important to choose the right approach so as you can view the results that you are looking for. Your approach should be efficient and output the results clearly so anyone looking at them – not just someone involved in the process, would be able to understand what is being measured and how effective the process used was.

Most approaches to measuring the process take the simple approach of just counting the metric, e.g. lines of code, then comparing and contrasting them against one another to find which process was best. This can be improved on by getting the averages of the metrics and creating graphs from the data, to better visualise the results.

You can also create graphs/models by comparing two or more different metrics and looking for correlations between them. This can often be seen in comparing time and aims achieved to try and find which processes perform better. It can also be used to tell which developer is making the most headway on a project.

A more complex approach is to use a cost-efficiency model such as the COCOMO model, which compares the costs and efficiency of different processes at lower, intermediate and higher levels. Each level represents a jump in data such as a higher cost/efficiency to the last level. This model needs fine tuning as it varies greatly from engineer to engineer, but it can become an effective and useful tool to analyse the process.

A meta-heuristic optimization algorithm can also be used to measure a process. It deals with the resources needed to finish processes and checks the amount of aims met by those processes in order to find the most optimal process. This doesn't take efficiency into account unless you make efficiency one of your aims, in which case you need to define a minimum efficiency, so the algorithm can compare the processes against it.

Assessment of measurement

Once the measurement of the software engineering process has been completed or is close to completion you need to decide how to interpret it. There are two main things to keep in mind when you come to this stage of the analysis, which can decide if you accept or reject the results of the measurement:

1. Communicability
2. Repeatability

Once those two requirements are met you can then look at the results and compare them with what you expected. Were they better or worse than you hoped? Do they tell you what you wanted to know? Can you see areas that could be improved? Should we be looking at other pieces of data to get a better understanding of certain parts of the process? These are the kinds of questions you should be asking when looking at the results. If you can't answer them or didn't get the right results in scope, then you should think about taking another measurement with different information or under different circumstances.

Communicability

When displaying the results of a measurement it is necessary that it is clear and communicative, so any readers/viewers understand what it is that you are measuring and the results that you have obtained by analysing the data that you recorded. Having clear results and being able to display them in the right manner is key to being able to understand if the process was successful and should be used again or unsuccessful and needs to be refined before further use.

Repeatability

Repeatability is an important factor that in many cases goes overlooked. It is when the results can be reproduced when someone else is under the same circumstances. If the results cannot be reproduced then you should probably err on the side of caution and reject the measurement of the engineering process as a whole, though it may still work as a measurement of the work done by the software engineer. The results should be repeatable for the measurement to hold up to scrutiny as it is supposed to be a measurement of the process used when developing the program, not the effort of the engineer. So non-repeatability means that your data choices were likely out of scope and caused too much fluctuation in the results for the measurement to be accurate.

Ethics

We all agree that it is important and often valuable to measure the effectiveness of software engineering. However, when doing so you need to be careful about how you go about doing so. There can be ethical concerns around the process, especially when it comes to the kind of information you gather and store in order to measure the engineering process. Ethical concerns are when part of your measurement may be unmoral, particularly when you deal with another person's information. There is also the question of whether or not you should even be measuring the process in the first place as it's not always in the programmers' best interest to do so, though this happens only rarely.

The biggest ethical concern with the measurement of a software engineering process comes from the gathering and retention of the information needed in order to analyse said process. When gathering the data, you need to be careful and ensure that all of the programmers/ contributors to the project consent to have their information recorded and used in the analysis. Some types of information which someone may be uncomfortable with you recording could be their name or experience. This data needs to be in the public domain so you're sure that they consent and can't pull it from the measurement. They may be uncomfortable with you recording that information as they don't want to know the results of the analysis possibly, but either way it would be an ethical concern if you were to take information about someone, or their code, without their explicit consent before hand.

You also need to be ready to get rid of the information recorded or show the owner the information you have on them should they ask. This is due to the new GDPR act brought in across the EU, meaning everyone has a right to know what information you have about them recorded and also to be forgotten and deleted from your records. This could also be an issue for older records/analyses you have on engineering processes as if they are not organized correctly now then you are in violation of that regulation. Those records can then clearly be ethically unjust.

Another ethical concern would be when you use these methods to judge a programmer. The measuring of a software engineering process generally is meant to reflect on the process taken when developing the program. However, it is also often used to judge how effective/ skilled a programmer may be, which I mentioned above. This calls into question whether or not it is ethical to judge and reward/criticise someone for the processes they may have used. This is

especially so when they have to follow certain guidelines that may be put in place that would limit them from following their usual process.

The final ethical concern that I will talk about is how the information that was gathered is stored. If it isn't in a secure environment and encrypted, then there can be cause for ethical concern. The data stored can be sensitive in some cases and not anyone should be able to see it, as such the people whose information has been stored have a right to know that their data is safe and secure after being recorded.

Conclusion

In conclusion, the measurement of a software engineering process is more complex than it appears on the surface. There are many methods and metrics you can use to do so but the real trick is knowing when the best time is to use them. They each have their own advantages and drawbacks which make them more suitable for different circumstances.

The analysis of the metrics to get the results must also have thought put into it in order for the measurement to stay in scope. It's useless to gather all of the information if you don't know what to do with it or what you want from it after all. You can also analyse the differences between the processes in order to find the best based on circumstances that you set.

Measuring software engineering has many different approaches in all of its sections that have their own advantages and drawbacks and can synergise together well if used correctly. So, if you make the right choices for each section the measurement can be quite fruitful in helping you improve your development process.