Measuring Software Engineering

Donal Hogan - 15324573

# Introduction

This report is on my findings and thoughts on how the software engineering process could be measured. I will go through different process that could be used, their pros and cons, the kind of data that could be measured, how to approach measuring the algorithmically and the ethics of doing this.

# Measuring Software Engineering

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. You could use a mix of these issues and be more likely to get a better idea however they all come with their own drawbacks. I’ve detailed a few processes below.

## 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.

Here’s a list of somethings 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.

## Aims met within a period

## Cycle time

## Issues resolved

# Algorithmic approaches

# Ethics

# Conclusion