The Challenges of Measuring Productivity in Software Engineering

Leo McClean

Trinity College Dublin

# Introduction

The advent of software engineering came at the dawn of the 1960s, when most of the modern world realised its problems could be simplified by embedded systems that simplified large calculations and operations involving machinery. For example, with space travel, complex calculations needed to be performed constantly by an on-board computer for a shuttle to correctly exit and re-enter the atmosphere. This ‘software engineering’, where the term was born in Cape Canaveral, involved first designing the routines by hand, then moulding them into punch cards to be tested. When the program was proved to be working, it was hard wired into a shuttle, quite literally, using wires threaded through and around magnetic cores (through signalling a binary 1, and around signalling a binary 0) (McMillan, 2015). It wasn’t until years after the mostly successful Apollo launches that software evolved from binary, into the first high-level languages such as COBOL and BASIC. In the thirty-year period from the 60s until the 90s, we leapt from punch-cards to a variety of languages, some imperative or procedural – object-oriented or functional. And within this rapid development of new technologies to better facilitate the creation and application of software, we ultimately failed to properly design any method to successfully determine the required input of labour for designing a system, the output we can achieve with a certain number of programmers, or even an effective measurement of productivity in software engineering. And although we have had another 30-year period of innovation and advancement since 1990, we still struggle to adequately measure the process of software engineering. In this essay we will explore what data we can measure as software engineers, why it is difficult to transform this data into a measure of productivity, how companies today are improving their development cycles, and how, as we look to the future, software engineering may become a fully measurable process eventually.

# The Data of Software Engineering

With a piece of functional, concise, yet fully fledged software visualised as the end-product of the work of a team of software engineers, what metrics could be used to measure progress towards this goal? What metrics are useful? Although having a completely digitised development means the measurement and collection of data is easier than ever, separating out the relevant metrics from the bulk is challenging. With so much data, what really tells the story of productivity?

When building a physical item productivity or efficiency can be measured as a function of output over input. We can also track waste with the creation of physical products. With a production speed of 50 units per hour of labour for say, and an average wastage of one unit, any manager can take a proposition such as: double efficiency for double wastage; and quickly decide whether 100 units per hour is worth the 4 unit wastage. Here all the numbers are relevant to the work, and finding inefficiencies is quite simple. In the manufacturing industry the easiest way to measure inefficiencies is to look at the physical and process flows, that is the movement of components around an assembly line, and the steps required to process base materials into the finished product. Rearranging an assembly line can yield massive increases in efficiencies for nearly no cost to a business (the relevant number being distance travelled in metres).

With development of software, there is one central location where all work is carried out. Although an increase in say, monitors, can yield improvements in productivity similar to physical flow mapping, these efficiencies are negligible as every component tool, or system at an engineer’s disposal is at their fingertips, merely milliseconds away. What then is measurable in our development assembly lines? Number of lines of code, time spent at workspace, number of commits, deployments, rollbacks, or bugs, these are all measurable quantities. Indeed, parallels can be drawn here to the world of physical design, such as product design or architecture. But the key differences between software and ‘hardware’, or physical products, is the end goal of a physical product has a fixed form, a shape, a measurable and visual entity that cannot exceed the constraints of a certain space. Software has no such final form, merely a set of requirements to fulfil. It is to the discretion of the programmer to find pathways to fulfil these requirements, and this can create huge differences between the visions or work of different programmers working to similar goals. A relevant example would be like an essay assignment for students: despite a fixed topic, or length, the content of the work and how it was reached would be entirely different processes, for some students taking significantly longer for comparably worse work.

With software its beauty isn’t in the bulk of its source code, but how less code can accomplish the same, or better, results. The removal of a fixed space which the program must occupy (mostly) allows software engineers to explore a variety of options to achieve the desired functionality. Indeed, in situations where a maximum byte-limit is given for a piece of software, a program holding less space may be more time-efficient than a larger alternative. This makes ‘lines written’, as a quantity, difficult to use to measure any amount of productivity. The worst engineer could write a hundred thousand lines to achieve a barely functioning product, but the best could write an accomplished final product in significantly less.

Time spent at workspace is also difficult to use as a valid comparison for work done. In one of the initial largest switches of a company to a 4-day work week, there were gains of productivity of around 20%, despite the decrease of 7 hours per week of work per employee (Booth, 2019). Although this is a more general approach to ‘time spent at workspace’, it is a valid critique of how time invested is not always proportional to input regarding productivity. Especially in design-oriented areas such as software engineering, problem-solving can account for large parts of the work, relying not just on time spent but the creative and inspired parts of an employee. While certain individuals could be stumped by a problem for days or weeks, others may naturally work around the problem with no difficulty. It could be said that the employee ‘spinning their wheels ‘ is inefficient, but in a field requiring not just knowledge but also creativity to deliver is it fair to measure a metric so dependant upon creativity, which itself is immeasurable (except anecdotally)?

In an interesting shift from measurement of progress to measurement of detrimental factors, number of bugs is a relatively easy metric to base decisions on provided that all bugs can be observed. This is unique to the other two data points examined thus far, as bugs are not always innately visible to a programmer, tester, or quality assurance worker. They can rely on incredibly specific use cases to be brought forth, or unique hardware, or even certain future conditions that will not be met for a long period of time, such as number of users, date, or other incremental variables. Sometimes bugs can rely on failure of several individuals works to accommodate all use cases and situations, and in these cases it is the managers discretion to decide which individual failed to make the necessary unit tests or precautions to prevent such bugs. In most cases though, it is easy to pinpoint the specific work which caused a bug, and which software engineer wrote such code. Measurement of creation of bugs as a metric can be useful when observed aside other metrics, such as lines written, so that a more focused figure such as bugs per line coded can be reached. This will highlight underperforming software engineers who need to work on their contributions.

And so we arrive at system specific metrics, such as number of commits, number of deployments, and number of rollbacks. In most every software company there is a different pipeline, or process for the development of software. With in-house repositories, development, staging, and production servers, as well as a myriad of other tools to assist in the engineering process, we find ourselves a myriad of metrics we can observe and use to measure productivity.