Thomas Donovan – 14316040

Measuring Engineering - a report

**Question:**

To deliver a report that considers the ways in which the software engineering process can be measured and assessed 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.

**Introduction:**

Software engineering has long been a difficult profession to measure, due largely to the complexity of the work done and the high level of specialisation which is often needed, if you don’t fully understand the project or the roles that team members fill it stands to reason that you would find it difficult to say how valuable they were or how “good” a software engineer. This can seem somewhat ironic due to the huge amount of data that is obtainable from software engineers and also because many methods of monitoring employees’ performance in other professions are done using software, for example toggl or other timesheet or employee management software. Different methods have been used to attempt to measure a software engineers, each with their own issues and benefits, and ranging from the very simple to very complex. I believe that in modern software engineering organisations must find the right balance when measuring and must decide what data they really need and what downsides they are willing to accept in order to retrieve that data. Care must also be taken in how the data is gathered and stored, both from a practical standpoint and an ethical one.

**Why measure?**

Though software engineering can be difficult to measure it is important that it is measured. With so much of modern life now dependent on software it is imperative that we maintain an acceptable level of quality in the systems being created. By measuring software engineers and the development processes they use we can make sure that the systems we get are of an acceptable quality.

To look at it from a more business-oriented point of view, software engineers are expensive, and employers want to make sure that they have capable, hardworking employees. For other industries it can be easier to see how effective their employees are, but those measurements cannot be avoided just because they are difficult.

Another reason for measurement and data gathering that is extremely important in the planning and development stages is to avoid having to do unnecessary testing and patches. In the past testing was an expensive and arduous process and editing code was much more onerous than it is nowadays, as a result the software development process was more inclined to try and deal with issues early in the process to avoid these. Nowadays testing is cheaper and patching can be done extremely easy and cheaply in comparison, in some cases only taking days, however in the days of handmade core memory changing code was slightly more difficult than pushing some new changes to GitHub.

**Historical software measurement:**

In the past many different methods were used to attempt to measure the productivity of software engineers and the quality of software projects. Many software teams were faced with answering to managers who could impede progress even further by now entirely understanding what was important for the project, for instance, adding more members to a team when a project is behind schedule, somewhat counterintuitively, often delays them further as detailed in “The Mythical Man-Month”. This is a case where requirements and performance metrics actually have a negative impact on software quality.

Issues can often be created when measuring performance and setting goals for development teams. For instance, measuring lines of code created less readable and needlessly long codebases, which influenced maintenance and performance of the code itself. Measuring the commits of programmers could result in pointless commits or needless changes, which can increase the chances of bugs and actually make people less productive. Then issues can arise from other metrics such as speed, projects which are rushed are much more likely to have errors or bugs in them, as it only stands to reason. Other methods of speeding up software development can also have detrimental effects, such as adding more members to a team, which can actually delay the project due to the lead time required to understand the project.

The final issue that has historically plagued software projects are barriers to communications between, clients, engineers and project managers, due to the highly technical aspects of the work non-technical based clients cannot be expected to fully understand the software and communicate what they require in a technical sense and likewise the development team will likely not have a full understanding of how the business works and what functionality is needed from the project. This is where project managers and consultants need to bridge the gap between the two, as there is no point developing software if it is not what is required.

**Modern measurement:**

Modern software development processes have changed a lot since the early days of software engineering. With the rise in the amount of software made as well as it is becoming easier to make changes to code the importance of getting things right initially has been replaced with the need for speedy releases to stay ahead of the competition. The fierce competition currently seen in software seems not to reward perfect software and offers more of an advantage to first movers in a market. The accessibility of programming, that is to say how easy it has become to take up programming and create software for yourself has also given rise to more competition and lowered barriers to entry. High level languages such as python among others make code more understandable and have flattened the learning curve for interested individuals, evening out the playing field.

Agile focuses on software cycles and the iterative nature of modern development and face to face communication and feedback from clients, having the correct software evolve through many cycles and be constantly revised until it meets the expectations set for it.

Personal software processes are a more holistic type of process, taking many past methods of measurement and combining them to present a more complete view of the progress of the project and the performance of the team. It combines the size of projects, the quality of code produced and the effort taken, while also measuring how closely the project follows its planned schedule. This overarching method of software development provides a more detailed look at the process but requires more documentation and bureaucratic effort.

**Modern software development issues:**

Many, if not most of the issues with modern software development stem from the extremely competitive nature of modern software development, where barriers to entry are low and first mover advantages can make or break your product. However, with speed comes some downsides, and teams must decide what they are willing to compromise on to meet deadlines. Also, the ability to push now and fix later has meant that there is less importance on shipping a finished product as it is now possible to correct mistakes right up until, or even sometimes after, the product is launched. Day one patches have become commonplace for much software and many modern software development ideologies have change the traditional software development lifecycle and almost expect failure initially, with the hope of succeeding in subsequent iterations.

This type of development makes testing more important and frequent as each iteration must be tested individually. The importance of appropriate testing is crucial, as the quality of the tests can often be as important as the quality of the code itself. If the tests are not properly thought out and implemented then they become a pointless waste of resources, however proper testing can save massive amounts of trouble in the long run.

**Issues with measurement:**

As George Box famously said “All models are wrong” and “the scientist cannot obtain a "correct" one by excessive elaboration” it is a popular theory to gather all the data one can in an attempt to make a perfect model that can give you all the answers, however no matter what you gather it is impossible to make a model that can predict the future, the context of each project must be taken into account and facilitations made for it, if a team is not performing as well as it should on a project there could be any number of reasons why that is, from personal or client issues to the technical complexity of the task itself, and these must all be observed and taken into account when measuring the performance of a software engineering project.

One of the issues with measuring employees is that the more data you gather and the more obtrusive it becomes, the more disruptive it is for the development team, and the knowledge that they are being monitored can even affect their behaviour, so you must make a decision on how much disruption is too much, you must also factor in the mood of your team, as the happiness of tea members can greatly affect their work. Happy employees are likely performing better however if they are upset by the obtrusion of your data gathering they may become less effective due to it, making the data less than useless. However, on the other hand, as explained in “Searching under the Streetlight for Useful Software Analytics”, the easier data is to gather, the less useful it often is. A balance must be struck where useful data is gathered but not at the expense of productivity.

Also, what can be an issue is knowing which behaviours are beneficial and which are not, for instance, taking frequent breaks may sound negative, however if it helps focus then it is surely positive, or someone not writing any code for a long period of time could be explained away by them taking time to understand what they are writing before they start. When measuring employees, it is very important to also take the circumstances and personalities of each individual into account, otherwise your results could be skewed and the data may not reflect the actual working relationship of the team.

**Ethics:**

While in many cases it can be tempting to gather as much data as possible, it is important to take into consideration the ethical implications of what data you gather, how you gather it and how it is stored and managed. The principle of corporate social responsibility should also be observed to ensure that harmful software is not inadvertently created and that the purpose it is being used for remains moral.

While gathering data and building profiles of teams and individual members it is important to question each data point to figure out if it could potentially be harmful. For instance, race, gender, politics and sexual orientation are all variables that some may have a prejudice towards, therefore gathering this data should either be avoided or if they are stored, must be stored in such a way that the data is sufficiently protected and cannot affect those individuals.

The topic of data protection is very topical at the moment with the new general data protection regulation coming into effect next may, which details the way in which data is to be stored and managed in the European Union. It states that every company handling large amounts of data must have a designated data protection controller to ensure that the data is being handled in accordance with the GDPR, it also changes the laws to make it clearer to users when and what data is being gathered and makes it so that there can no longer be “opt-out” clauses in data gathering, and now you must get the user to consciously “opt-in” to their information being collected. There are also now new rules about how data must be deleted and when as well as how people can access their own data. Whether these regulations will be properly adhered to by large corporations remains to be seen, and the spotlight is very much on Ireland since we have so many large software companies such as Facebook, LinkedIn and Google just to name a few. Previously our laws were seen as rather lax by the rest of Europe and we were known previously for giving companies that didn’t comply a lenient penalty, however with the new regulations coming in there are more concrete penalties in place for data preaches or misuse of data. However, these laws affect all European data, and there have been many cases of other countries being non-compliant, most famously the U.S. where the safe harbour agreement that was set up between the two was widely panned as being ineffective, and shown to be so by the Edward Snowden leaks and Max Schrems widely publicised case against Facebook. The new privacy shield, while slightly stronger, appears to also be considered largely ineffective.

**What constitutes a “better” software engineer?**

While addressing the question of measuring software engineering much of the difficulty comes from the question: what constitutes “better” in software engineering? With so many different specialities, job descriptions and different frameworks used it is nearly impossible to come up with an empirical metric for what is best in a given scenario, it depends on too many factors, such as experience of the team, what frameworks are easier for certain tasks, details of the project specification, timeframe, need for future proofing, the list can go on and on. The need to take things on a case by case basis and the speed with which the industry evolves means there will likely never be a single correct way of doing something, and for many people that is what makes software engineering so appealing, the ability to problem solve yourself and come up with your own method to complete a task. With this in mind obviously some things are just easier that others, that’s why you rarely find people writing assembly code anymore.

While all this is true for individuals there are certain traits that can be more or less beneficial to a project. Being the “best” programmer in a team does not always mean you are the most valuable member of that team, a wide range of skills are needed to complete a project successfully, such as the ability to communicate, both with the client, and project managers, and other team members, without proper communication you may end up with the best written piece of software ever created, that unfortunately doesn’t do the task required of it. Software architecture and planning is also a skill that is sorely needed in every project, the ability to visualise the system and plan the design before jumping into the coding of it can save huge amounts of time and effort later. There are also people especially suited to testing or quality assurance in the team whose skills may not come into play until later on in the development cycle, so may appear to be weaker member of the team early on in development

**What to measure:**

How accurate/efficient do we need to be for the task – what do we really need?

A key question when completing a software system is: how good is good enough? At what stage will the system function as it is supposed to and how much effort past that point is wasted. For instance a system for everyday use that can complete a task 10% faster is not necessarily worth spending an extra 30% of the time on. Also after a certain point the extra memory or processing power required can become a downside, possibly causing certain machines to find the software difficult to run. This could even become more important as Moore’s law appears to be coming to an end!

In teams, the more important things to measure may be the quality of the software at completing the desired task or agile cycle time or project lead time, and using this data over time it becomes easier to predict how long a project will take, it is important to be able to provide reasonably accurate estimates for clients as to how long a project may take, but as Hofstadter's Law states: “It always takes longer than you expect, even when you take into account Hofstadter's Law.” But by measuring past projects and taking steps to properly design the software initially and not make changes to the project specification it should become possible to semi accurately predict how long the project will take and how much it will cost. In the world of software engineering where over half of projects are failing, many due to going too far over budget or over deadlines accurate assessments of these two are absolutely critical.

In the monitoring of software projects, the documentation, though occasionally tedious, serves a critical role in ensuring their success. Project specs, use cases and progress reports serve as key methods of communication between team members and clients and also provide a resource to look back on during the course of a project to make sure that they stay on topic. Team member appraisals during the course of a project can also be very helpful, even though they can be unpopular, being able to see what your team members think of your work and where you can improve can be invaluable, as it is often difficult to evaluate yourself, and through these you can help teams communicate more openly.

**My suggestions:**

My opinions on measuring software engineering is much like agile with more software architecture. Keep Agile’s frequent communication with the client and project managers and emphasise more communication with other stakeholders in the endeavour, such as employees of the business it is for or customers, I believe that more effort needs to go in at the planning stages of projects and that project managers must spend the time to understand the needs of the client and also be able to understand the technical requirements of the system. With this extra communication and planning I hope that software can be less iterative and more of a step by step system. I would also suggest adding in prototyping and regular project updates for the client to facilitate this. the importance of dedicated testing and quality assurance personnel being a part of the team during the development process I also believe will be crucial, as all members will then fully understand the system and the process by which it came about. The multidisciplinary aspects of teams like this give them access to a much wider range of problem solving capabilities.

**Conclusion:**

The process of software engineering and the measurement of it are still being developed and changing as the different aspects of software development change. Though measurement can be difficult and there can be issues, both ethical and practical it must be done to ensure successful projects, and I believe the measurement process can be improved more to make it more accurate and useful. One thing is for certain, the future of software engineering will surely evolve greatly over the next few years and decades, just looking at the changes that have been made since software began suggest that the future could look very different to what we currently see.

**Bibliography:**

* Searching under the Streetlight for Useful Software Analytics - Philip M. Johnson
* How to improve performance of software systems: A methodology and a case study for tuning performance - Siddhartha R. DalalMichael S. HamadaTzyh‐Jong Wang ([link](https://link.springer.com/article/10.1023/A:1018910926921))
* The Carnegie Mellon Software Engineering Institute
* Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System - Philip M. Johnson
* The Goal Question Metric Approach - Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach ([link](https://s3.amazonaws.com/academia.edu.documents/40605563/gqm.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1511987559&Signature=Kbgq%2Fv7KivI4EjZH9%2BcYmMDhyqQ%3D&response-content-disposition=inline%3B%20filename%3DThe_Goal_Question_Metric_Approach.pdf))
* No Silver Bullet - Essence and Accident in Software Engineering – Frederick P. Brooks, Jr.
* You CAN (And Should) Measure Software Engineering Performance - Nader Akhnoukh ([link](http://engineering.kapost.com/2015/08/you-can-and-should-measure-software-engineering-performance/))
* It sounded good when we started: a project manager's guide to working with people on projects - Dwayne Phillips & Roy O'Bryan
* Science and Statistics - George E. P. Box
* IT And Architecture: Inside-Out Perspectives - Bruce Laidlaw, Michael Poulin