CS3012 Measuring Engineering - A Report

Thomas Fowley

15315353

Measuring software engineering involves the collecting data during various stages of the software engineering process, measuring this data in some way and finding results which aim to ultimately aid the researcher (often a company or team) to improve their process and better understand what is needed to implement future projects.

This report serves to outline each aspect of this measurement process as well as offer insights and personal opinion on various different techniques commonly used today during each phase of the process. First, the subject of the data itself, what is and often isn't measured. Then we look at the measurement itself and how this data is used in a large scale followed by a brief overview of the personal software process used on a much smaller scale. This leads to a review of various algorithmic approaches used to measure the data as well as ethical considerations and finally, an assessment of this measurement and closing thoughts.

Measurable data

The first step in the process of measuring engineering involves the gathering of data, this could be in a large scale with accumulation of metrics from hundreds of programmers or a more focused effort on smaller teams or even individuals. There are many aspects of software development that can easily be measured while other aspects are not easily or cheaply quantified in simple terms.

The simplest and thus cheapest methods of data collection include:
Time

The overall time taken to complete tasks, this form of measurement has many limitations revolving around the fact that humans are not particularly good at estimating the time required for certain tasks, this can lead to underestimations of time required or team deliberately overestimating in order to have a more comfortable timeframe to work with thus efficiency is rarely achieved.

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law. — Douglas Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid

Lines of code

The amount of lines of code an engineer writes is also limited by far too many aspects to make it efficient. Most programmers would agree that more code does not necessarily equate to better code and offering programmers incentive to write code would, in my opinion, simply lead to more unnecessary code and long winded code structure.

Test coverage

How much of the code is used for various use cases and unit tests is perhaps a more accurate although still somewhat rudimentary way of assessing the quality of code, it is certainly preferable to lines of code or keystrokes registered but is limited by the accuracy of the testing and in no way gives any indication of the efficiency or scalability of programs.

Other

Other data which could potentially be gathered includes hours a programmer spends at work, how time is divided between meetings, calls or coding as well as reports from managers/clients after team meetings, these offer a more qualitative idea of what work is being done well and how the project is developing and implementing

team meetings and measuring programmer sentiment towards a certain project has become almost a necessity in the software development process today .

This leads us to some more tricky to quantify aspects of software engineering, there are a vast array of sought after qualities for an engineer to have, these could include leadership, helpfulness towards other team members and overall programming skill.

Leadership

Leadership is a difficult skill to accumulate data for due to its subjective nature, one project manager may bring the best and most efficient work out of one team but cause issues in another team as well as this it can be hard to differentiate between talented leadership and talented workers making it more logical to simply ask teams during meetings and project reviews who they thought displayed potential, however this then reverts back to subjectivity thus making any results very anecdotal.

Helpfulness

How helpful a certain programmer is to other team members/colleagues is near impossible to measure however is considered extremely valuable in programmers today. Measuring lines of code,commits or time taken however give no indication of how often an engineer may solve less experienced engineer’s problems or help others with several aspects of the development process. In fact, these rudimentary measurements are often negatively skewed for these types of developers as they will often help others at the expense of their own time thus making it seem as though they are doing less work.

Although many aspects of a team's work on a project can very easily be recorded, even in large scale, I still think that more work needs to be done with these results to actually determine if a certain programmer or team is “good” or “bad”. This

however does not necessarily mean there is no merit whatsoever in data accumulation and there are many practical applications for both individuals and larger groups.

Measuring software engineering

In this section we will explore several metrics used in agile and lean processes, such as leadtime,cycletime, open/close rates and “sprint”, and what they can offer, as well as briefly overviewing measurement of engineering with security as the main focus as opposed to efficiency or work quantity.

Leadtime:

The total time taken for a project to reach its conclusion, this involves the conceptual stage all the way through to production, this metric is useful for teams to compare and contrast various projects as well as offer more accurate estimates for customers.

Cycletime:

The time taken to implement changes to a project or system can be extremely useful for a team to estimate future changes and can be invaluable if teams need to work on projects with continuous delivery where changes may need to be made in extremely tight time frames.

Sprints:

“Sprint” is the name given to a process whereby a “scrum master” sets a certain time frame allocated to a certain task and developers then work on this time, often having daily standup meetings to assure the sprint is going smoothly. The number of

sprints as well as the success rate of the sprint time frame is important to measure as it allows for accurate but also dynamic estimations of time required as well as work put in by a team.

open/close rates:

This again is a more dynamic measurement which shows how well a team reports and solves issues within a certain time period e.g a sprint. This doesn't necessarily give any information about the quality of the code or even the team, however it is useful to see how quickly issues come up and are dealt with during a project.

Security:

Security can be measured in a similar manner to open/close rates in the sense that the number of security issues on a certain machine, i.e Endpoint Incidents, are counted over a given period of time. This data is used in tandem with the MTTR or Mean Time To Repair which calculates how long these security issues take to be resolved. Using this data can offer a reasonably accurate estimation of how secure a system is or at the very least, how well the team can repair issues as they arise.

Although metrics are important to have and as is with most data, the more gathered the better, I still personally believe that numbers cannot tell the full story and the best way to measure engineering is to talk to engineers. Of course this concept itself has limitations such as human error, honesty and time when dealing with teams of tens or even hundreds of engineers, therefore a reasonable middleground should probably be found for the most accurate and fair representation of a team of project.

Personal software process

The previous section offers an overview into measurement for teams, groups or business however there is a slightly different process an individual engineer might choose to employ to measure of asses their strengths and weaknesses, this is known as the Personal Software Process (PSP).

The Personal Software Process was created by Watts Humphrey to help software engineers improve their planning and time management skills, improve quality and efficiency of work and reduce the number of issues in projects being discovered to late in the development process to be cheaply or easily resolved.

The PSP is structured into three levels:

PSP0 is the initial level which includes three main phases: planning, development and review. The planning phase may involve a conceptual overview of the project as well a rough sketches or modelling. This is followed by the development stage which involves design, coding, compiling and rigorous testing. Finally, during the review, an engineer will note issues or shortcomings as well as what worked and why to help improve both their process and overall quality of work.

PSP1 then involved taking the data gathered during the previous level and using this data to find the total time spent on various aspects of the project and cross reference this with time estimates to improve the engineer's estimation and time management skills.

PSP2 adds what is essentially a more indepth review phase in which the engineer reviews both their design of the project as well as their code using checklists and calculating how many issues arose and how well they were resolved,

This process seems well thought out and designed, and I believe that if properly implemented, could help any software engineer improve their process. The steps seem intuitive and easy to execute in most projects and I many f the practices such as a coding standard or the development stage of PSP0 are implemented universally in industry today.

Algorithmic Approaches

In previous sections we have explored the use of data accumulation such as lines of code, test coverage etc. These metrics are reasonably straightforward to acquire however if a business is interested in a more in-depth analysis of a project, a more complex algorithmic cost model would need to be used.

Algorithmic cost modelling is using estimates as aspects of a mathematical function to determine the cost and schedule of a project. The example we will be exploring for the purposes of this report is the COCOMO model but many models are similar in the sense that a project manager or team will give estimates of variable such as human effort and code size and a seperate team will give estimates of social factors, demand etc.

The COCOMO Model

This model is an empirical model based on project experience which has various levels, (basic, intermediate and detailed), as well as three separate modes, (Organic Embedded and Semi-detached), allowing it to be used for various different types of engineering projects.

Basic:

This level is used for quick, cheap and early estimates of a project overall cost and time required. Accuracy is sacrificed in order to make estimates quickly and cheaply.

Intermediate:

A more accurate estimation that the basic level, this level considers several aspects of the software engineering process, such as the factors and data previously mentioned in other sections. This level takes longer to implement but offers a more accurate overview at potential cost and timeframe.

Detailed:

The last option is the most detailed, hence the name, and offers the most accurate estimates because it accounts for a wide range of factors during each phase of the engineering process making it harder to miscalculate by much when this level of detail is implemented in COCOMO.

COCOMO Modes:

The three COCOMO model modes are:

Organic:

This mode is suitable for small scale and relatively stable projects where not many issues are expected.

Embedded:

Appropriate for large projects with a tight time frame. These projects are usually innovative in nature and would often involve complex structures or algorithms.

Semi-Detached:

Semi-detached mode is a term used to catch any project which does not fall under the previous two categories. This could include medium sized projects or larger projects with less time constraints or anything in between.

Once the scale and level of the project has been assessed and the mode and level of the model are chosen, the main factors algorithmically calculated are effort and resources. Effort is usually expressed in person-months and resources can be calculated with effort to find an estimate of calendar months i.e four working weeks (five days per week). The basic formulae are as follows:

Effort:

MODE	EFFORT
Organic	$E = 2.4 * (S^{1.05})$
Semi-detached	$E = 3.0 * (S^{1.12})$
Embedded	$E = 3.6 * (S^{1.20})$

Where E is in person-months and S is in thousand lines of code.

Resources:

MODE	SCHEDULE
Organic	$TDEV = 2.5* (E^{0.38})$
Semi-detached	$TDEV = 2.5 * (E^{0.35})$
Embedded	$TDEV = 2.5 * (E^{0.32})$

Where E is calculated above and TDEV is in calendar months.

Images taken from MU lecture slides.

These formulae can again be improved by adding in extra variables and factors such as cost drivers, source code size, training time required etc but these images serve as a basic overview of the model.

I think these models could prove very effective especially when more variables are considered however I still don't think an algorithm can tell a full story. In person interviews and team meetings are still essential to accurately track progress.

Ethics

Before I conclude, the final aspect of the engineering process I will be including in this report is ethics. Ethical considerations are often overlooked by many engineers however some could argue that this is one of the most important aspects of the overall conceptual design of a project. I will refer to the concept of data sovereignty and privacy which should be considered for large scale commercial projects.

First we should address the accumulation of the data itself. In my opinion this is simply too invasive in most cases and would personally make me uncomfortable working in these heavily monitored environments. Recording keystrokes or monitoring an engineers computer screen is unnecessary in most cases. I personally don't agree with this becoming the norm and something to "get used to". A case could be made that one must simply separate work life and private life but in an industry that so heavily emphasises worker happiness by offering free gym memberships, meals, yoga classes and work outings, this monitoring seems almost ominous.

Note on data sovereignty and data protection:

Once we get past the data being gathered, there are still several ethical issues to be considered, one of these issues mentioned in lectures is data sovereignty which s heavily linked to privacy and data protection. Data sovereignty implies that data stored must be subject to the laws of the country in which the data was collected. Due to EU regulations, personal data stored in the European union must abide by the GDPR, making this not only an ethical consideration but a legal one.

Conclusion and Assessment of Measurement:

My assessment of measurement revolves around the fact that measuring engineering is inherently flawed due to data fidelity and human honestly issues. Data cannot give the whole story but engineers are prone to miscalculations and subjectivity. For the most accurate result I believe a balance of data, team meetings/reports and external expert opinion should be employed to have the most well-rounded and fair results.

With these results I think there are many aspects of the engineering process that should be investigated for potential improvements, such as tools/software which could be more efficiently implemented or if seminars or external training is needed for a team to better employ the tools and frameworks into their workflow. Another potential result of these analyses is to record which workers are not commiting quality code or working well in a team and may need retraining or letting go, however this specific case must be approached with caution as I believe these results should not be considered conclusive when it comes to how “good” or “bad” a specific engineer or team may be.

This leads us to some final points to note. Although this data can be useful it is important, in my opinion, for companies to remember that resolute conclusions should not be taken lightly and that even a strong correlation does not imply causation. As well as this, it is important to remember that every engineer and every team is different and what may work for some projects may not work for others and final the aspect of fairness should be considered during each phase and in every aspect of the measurement process to increase the comfort of the engineers as well as the accuracy of the measurements.

Resources:

- The Personal Software Process (PSP) Body of Knowledge, Version 2.0
  August 2009
  By Marsha Pomeroy-Huff, Robert Cannon, Timothy A. Chick, Julia L. Mullaney, William Nichols
- Lecture slides on algorithmic approaches from The University of Maryland
  http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/slides16.pdf