[CS3012]

Measuring Engineering - Report

Megan Whelan Dalton

Student # 16324322

Brief:
"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."

Contents:

[Introduction]

The measuring of software engineering, how to approach measuring the process and which in itself is the most accurate way is often contested. It seems for every article claiming that “our metrics are the best” there is another two articles refuting them. The concept of “measuring engineering” is quite broad and everyone seems to have their own interpretation of it. Within it falls the subtopics of measuring speed, efficiency, productivity and quality, as well as the debate of measuring the process versus measuring the product. The vastness and debate of the subject is probably why there exist no single standard practice within the industry, each company having their own approach to measurement. In such a large industry, there is no “one size fits all” solution, instead it is best to understand the individual metrics and their purpose then decide which best suit your needs, your project, and your work environment.

However coming to grips with such vastness, and it almost become a battle of “measuring metrics” and “assessing assessment”. Luckily there is an abundance of computational software that enables the user to track an array of metrics, that help draw and deliver a clear analysis from all the nitty gritty details that go into engineering. These range from manual data entry and assessment approaches, to goal oriented target trackers, to almost invisible monitors that seamlessly tracking and analysing your progress while you work without notice. Being able to see and quantify progress, only promotes and enables more progress and growth, creating what should be a positive cycle. Giving metrics to the process and product also enables future improvement, seeing the level or “grade” of work you produce pushes you to aim for the next or maintain in the future. If that level falls, you can gain a greater understanding of how and where you went wrong, rather moving on with a similar ineffective method.

Unfortunately the impact of these metrics are not solely positive. There comes a moral dilemma when monitoring workers, with gathering and storing this information. This is particularly prevalent with the more “seamless” approaches to measurement, monitoring the engineer with little to no interference so they perform as if they weren't being monitored. Is this lack of awareness, also a lack of consent? Does there exist a pay-off between the “accuracy” of data and the awareness of the subject? Is this pay-off worth it, and ethically is it allowed? It is a similar dilemma that arises within psychological research studies, awareness can skew the data and impact result, whereas a minimum knowledge enables subjects to behave more naturally. Depending on the metric being measured the answer often differs, counting lines of code may not cause a moral imperative but counting the lines of code base off a person's mood or well being might send up red flags.

[Measurable Data]

There are a wide variety of metrics used behind measuring engineering, this is undoubtedly as result of the wide variety of practices within software engineering. No two projects are the same, no two workplaces are the same and no two workers are the same. Hence there is no standardisation to measure process or product, but there are plenty of metrics and measurable data to help achieve understanding of quality of work and worker, result and product. Understanding the metric, can help better assign it to the method. Process versus product are two very different things to measure, the process is ever evolving and fluid, where the product or end result tends to be more static in nature. Should the judgment be off the work put in or the end result? Which should define and determine an engineer. Which determines quality, and what determines efficiency?

More often than not, the product is used as the form of judgment. The product is a final result, and culmination of all work and effort, the finish line. It makes logical sense the software engineering would be just off the software engineered. But what are the metrics used to quantify the quality of product, and the caliber of engineer.

Lines Of Code is often brought up as a measurement of software, maybe as it is simple to track and easy to quantify. The significance of it within the debate might also be due to the fact that it was one of the first metrics to be used to measure programming productivity, and at one time quality. From an outside perspective, it may seem that that length of code could help determine the effort or the complexity of the code. It's the age old question of "Does Size Matter?". Longer code means more time spent and therefore more robust code, better error handling and more complex problem solving. Or does shorter, concise code correlate to time efficiency? No. It's not what you got, it's how you use it. No one requests software of X amount of lines of code, coding isn't a short story competition. In practice, you are given a problem to solve and if you can do it in 5 lines or 500, it tends to not matter; so long as it works. Lines of Code, is simply a measurement of length and unless used in combination with other metrics, is a pretty useless form of measurement.

Refining the Lines of Code metric can improve its usability by combining other metrics. One example of this was to calculate the defects per line of code. In an ideal design there wouldn't be a single fault, but a count of defects doesn't necessarily determine the overall quality. Defects or "bugs" are not all made equal, you could have a single defect that is fatal, where you could have 5 per line that don't impede the functionality of the software. The defect per line metric can help you pinpoint errors to revise and fix but isn't a definite guarantee of how operational a product truly is. An empirical study by N.E Fenton and N. Ohlsson found that size based

metrics were poor predictors of defects. It was observed in another study by E. Adams that most operational system failures were caused by a small portion of faults that are hidden by regular means of detection. This proving defects per LOC, a helpful but ultimately useless metric.

In a similar vein of defect counting, estimating failure rates and software reliability modelling try an algorithmic approach to detecting the “chance of failure” and that of future failure. Reliability models checks the probability that the program performs a given function in a given time with certain conditions. It achieves this by executing the program until a failure occurs, removing the error and resuming the cycle. The time between these failures is use to predict that of future ones. This would seem like a fair metric to base the effectiveness of software off, except different models can produce different results and it is based of some assumptions that do not always hold true case to case. For example it assume that all faults are independent, that like in the defect counting approach they are not latent, and that by correcting current ones you are not running the risk of introducing any more. This creates similar issues as previous where not all faults are equal but are being treated as such.

As these more static metrics proved less and less useful, there was a shift to more abstract quantifiers, like “impact of code” and task based metrics. These Software Requirement measurements can be seen as more basic, and do not take an algorithmic approach limiting their predictive power. Function Point Analysis is used to predict both the size/cost of a project as well as the overall productivity. Though it is not considered a metric, more of an estimation. It is a calculation of all inputs and outputs, user interactions, interfaces and files used. Instead of focusing on how long the program is, there is a focus on how well it fits the solution not excessively lengthy and redundant code. It can be used like Lines of Code in conjunction with another measurement to convey more information. For example, by comparing the Function Point per hour, you can track productivity rates. It is said to provide and objective measurement for software that is applicable even in the early stages of development. However it was found in a study, that Function Points and LOC have a high correlation making it unsure if one was truly more effective than the other.

An issue with the previous metrics is the lack of understanding of the code complexity. This is where Cyclomatic Complexity is used. Using a control flow graph and the source code of a program, the number of independent paths of code is calculating. The lower this result, the lower the complexity is. A program with a lower complexity is easy to understand and more stable when it comes to modification and alteration. As result of the high independence of code it is safer to edit code without having to worry about affecting other segments that relied on it. Like Function Point Analysis, it helps illustrate the scope of a project. This method also showed that actual code complexity was independent of that of size, the complexity a result of

structure and increasing or decreasing the line count had little effect on the result. This metric helps not only determine the potential risk of a program but also the code coverage to enable the production of more stable and safe software.

Despite there being countless metrics to measure the product and the software in recent times, there has been a shift towards examining the process. Why? This might tie into the changing work environments, there has a trend towards team based projects. Especially in software engineering where most professional projects are worked on in teams in a collaborative effort, lone wolf indie developers though not completely extinct is definitely a thing of the past. So why can you not judge the value of the team off the quality of the product? In an ideal world you could assume that each part of the machine is working equally as hard and equally as good, but anyone who has ever been in a group project can testify that this is not always the case. Since the end product is such a intricate collaboration, it can be difficult to separate areas of the code and assign them to a single person, instead it makes more sense to judge the process of work.

Common measures of programmer "productivity" include LOC, test cases written and executed, documentation written and instruction produced per month. However, as explained before metrics like these don't always ensure quality. Lines of Code can also vary dramatically depending on language, lower level languages can take more lines to express what a few lines of a higher level language can. The same can be said for pages of documentation, a per page measurement encourages long winded explanations rather than clear and concise descriptions.

The notion of measuring the "productivity" of the engineer is a controversial within the industry, many agreeing that it is not truly possible to accurately convey "productivity". Despite this debate, many have tried to develop their own method of measuring it as such. Most of these theories focus around finding the most "productive" use of time. Time is usually the first metric that is tracked within the process. How much time was spent on the project? How many lines of code were written in X amount of time? How many bugs were logged and how many bugs were fixed in a given amount of time? Time is essential to the process affecting and being influenced by budget, as result of this time is also limited. So ensuring the most effective use of time is intrinsic to productivity.

Simply just tracking the programmers time spent on a project isn't enough to determine productivity, or effectiveness. But combining this metric with other previously discussed metrics is. The only issue with tracking time spent, is logging how that time is spent. It takes time to logged time, and time spent logging is time wasted. This is where Computational Platforms come in.

[Computational Platforms]

The beginning of Computational Platforms for measuring engineering began with Personal Software Process. The Personal Software Process or PSP is a metric gathering and analysing data from the engineering process. By logging and managing data of the engineer it enables better analysis of the work enabling them to improve and maintain good practices and products. In turn this promotes productivity. PSP is a low cost method that is both flexible and customisable, a simple spreadsheet can be tailored to suit the needs of a specific project. Unfortunately this approach is all manual, data must be logged yourself and calculations must be done by the developer. This can lead to issues with data quality, as human error is common. Inevitably this makes the analytics fragile, and the essential process conclusions drawn incorrect.

Hackystat was a third generation PSP data collection tool, that aimed to overcome the issues with manual logging and analysis of information. It is a lightweight tool, completely based on open-source elements. Developed by the University of Hawaii, they focused on developing ways to collect data with little overhead. This was achieved with both client and server data collection through sensors attached to development tools. This data collection occurred automatically on a minute by minute basis removing the intrusive nature of previous models. The process was no longer interrupted to track it. It encompasses all metrics mentioned previously, such as complexity and LOC, seamlessly giving a broad overview of the project, and detailed insight into progress. Individual in nature it allowed for a high level in privacy and analysis of a single developer, however it lacks in advantages for groups or for a enterprise as a whole. This tool focuses in on developer improvement and progression as an individual entity.

PROM or PRO Metrics, is similar to Hackystat in the fact was is is an automated tool for collecting and analysing software metrics and PSP data. It does this from the early stages of development right up until a product is completed. Like PSP, it encourages developers to refine and improve their process with individual analysis, however adds on tools for business management. One aspect that users took issue with while using Hackystat was that you could visualise each developer's DevTime trend. In PROM, individual could view their own data but also grant access to another for collaboration and assistance purposes. There is no access to an individual's data from a manager's perspective, however there is a whole project summary available to them. This creates a balance that provides useful information to see project progress and trajectory without compromising the privacy of the developers. This view hyper focuses on the aspects those within managerial roles take

more interest in such as cost and scheduling, while removing details that are used for a developer's own personal progress.

These means of monitoring progress and productivity are further enhanced by the shift in business towards adopting game theory. Gamification is a powerful tool that has been proven to promote motivation within business, and in doing so directly increasing productivity. The more motivated an employee is to work, the better the quality of work. The gamification of work/project based goals ties into creating intrinsic motivation, where the developer wants to complete a task. This form of motivation makes a person more likely to persist and work through difficult obstacles or problems encountered. This is achieved through an engagement loop, where motivation, action, and feedback are cyclically intertwined. The computational platforms fit perfectly into the cycle, delivering feedback to the user which drives motivation and in turn feeds action.

[Ethics]

When tracking, logging, and analysing human behaviour ethics and morality will always be called into question. Software Engineering is a fast moving and competitive sphere, so it is easy for people to forget about ethical consequences when trying to keep up within the industry or even get ahead in it. There is more concern with improving and innovating by the best means possible rather than the most moral, leaving some ethical ambiguity.

The computational approaches to measuring and analysing software process metrics are loosely comparable to that of psychological studies. At their core they are watching and studying human behaviour to better understand it, using collected and observed data for analysis purposes to draw conclusion to “measure behaviour” and in the case of software metrics, improve it. Psychology in the realm of sciences is a relatively new science the first dedicated laboratory only being found in 1876, and most major developments only occurring in the past 150 years. Like technology it was a quickly developing and highly competitive field, as result of this lines between innovation and ethics were quickly blurred. The first ethical guidelines were introduced in 1953, and being revised several times since there after to further protect subjects and strengthen studies integrity.

As mentioned prior there are lots of benefits to measuring engineering, and many studies back this up as well as solidify their importance within the industry. However tools have evolved from a simple unit of measurement to that of a behavioural training toolkit. Now there is nothing inherently sinister about this, promoting good practices and processes within a work environment, or any environment can and should be allowed. It all comes to how they are used, yet without restrictive measures or ethical guidelines it is easier for people to slip into an abuse of this technology. So is possible to apply ethical guidelines from psychology to ethics in measuring engineering to see how ethical current practices are?

Due to the similarities of psychological research and developer measuring toolkits, it is very possible to see the application of ethical guidelines from the former to the latter. The first guideline is informed consent, such that participants must be have research being carried outlined to them as well as its procedures and benefits, and the length of the experiment/research. This is easily applicable to the computational approaches, and for the majority is already present, where developers consent and aware of the fact their work is being logged and analysed. The debriefing guidelines is also very much adhered to and built into the tracking software, the automatic analysis and result display emulates the debriefing period after a research has completed where the findings are explained.

There is already overlap and adherence between the two realms that can be seen but then we fall into a grey area. The guideline of protection declares that no distress caused or physical harm comes to a participant. This protection is from mental harm, embarrassment, and fright such that the risk of harm no greater than normal lifestyle. In the context of a work environment which can be classified as “normal lifestyle” is the stress and distress caused as result of poor metrics absolved from the protection guideline, or should this fall under it to prevent embarrassment or mental harm? Stress is inevitable within a workplace, but in some studies surrounding Hackystat users mentioned discomfort about some of the features, a discomfort that would not be present without the utilisation of the tool.

Another ambiguous area is that of confidentiality. It very much varies from platform to platform, some following the rule of keeping data anonymous unless otherwise consented and having no names in reports, as is the way of PROM. All claim data privacy, but this privacy is enforced at different levels. Is the confidentiality strictly to the individual, confidential within a team or department or on the grander company scale? Should management have access to the data? This would have to be regulated on a industry level to truly be implemented correctly.

Withdrawal is the final guideline within psychology ethics. It is defined as a participant being allowed to leave the research at any point, and allowed to withdraw their data. They must also be clearly informed of ability to withdraw. In a shorter term research this is easy to implement. However in a long term work environment this withdrawal aspect is hard to implement. If employees were able to opt out of their computational platform would they? Without full participation from a workforce how useful can the software be? It highly unlikely the businesses are solely focused personal growth and development but rather result and output, providing this software on a small scale for those individuals who are interested seems redundant. Is it even fair to withdraw your own personal data from say your manager’s project overview, which would skew and misinform the results directly impacting project resources?

Without actually enforcing ethical guidelines it is impossible to predict their effect on the industry. Would it impede the rate of development and production? Today there does exist some self regulation on both a company level but also within the tools themselves. Can we continue to maintain this level of self policing throughout future growth and expansion?

[Conclusion]

All in all, there are vast and varied resources to “measure engineering”, countless metrics each with pros and cons and an array of supporters and detractors. The only way to truly judge what measurements suit your personal needs best is to understand what result you value and see how the various metric fit into those values. On top of the metrics the tools to measure them are also in abundance, cherry picking and combing the aforementioned measurements to provide contexts and analysis but also enable self-improvement and development. Even still it is difficult to grasp if these means are truly giving an accurate measurement of engineering, or just a means to drive growth within the industry. These enable the refinement and idealisation of the software engineering process. This doesn't come without ethical implications however, wherever human behaviour is being tracked and quantified there arises issue.

[Bibliography]

1. Fenton NE and Neil M, Software Metrics: Roadmap, Proceedings of the Conference on The Future of Software Engineering, p.357-370, June 04-11, Limerick, Ireland, 2000.

2. Fenton NE and Ohlsson N, Quantitative Analysis of Faults and Failures in a Complex Software System, IEEE Transactions on Software Engineering, to appear, 2000.

3. Adams E, Optimizing preventive service of software products, IBM Research Journal, 28(1), 2-14, 1984.

4. McCabe TJ, A Complexity Measure, IEEE Transaction on Software Engineering (4): 308-320, December, 1976.

5. Humphrey Watts, The Personal Software Process (PSP), CMU/SEI-2000-TR-022, Software Engineering Institute, Carnegie Mellon University, November, 2000.

6. Johnson PM, Searching under the streetlight for useful software analytics, IEEE Software, July, 2013.

7. Michael Jun Kiat Ong, Gamification and its effect on employee engagement and performance in a perceptual diagnosis task, University of Canterbury , 2013.

8. Singer J and Vinson NG, Ethical Issues in Empirical Studies of Software Engineering, Institute for Information Technology National Research Council, Canada, 2002.