MEASURING SOFTWARE ENGINEERING

ABSTRACT

In this report I will consider the ways in which the software engineering process can be measured and assessed. I will do this under the following headings: measurable data, computational platforms available to perform this work, the algorithmic approaches available and the ethics concerns surrounding this type of analytics. This report will be compiled using academic material on the topic of software engineering as well as the content discussed in our lectures.

INTRODUCTION

In order to understand how to measure and assess software engineering we must understand what it means. The earliest origin of the term 'software engineering' was by Margaret Hamilton. Margaret was aware of the intensity and complexity of the work and wanted to ensure people got credit and recognition for their contribution to this field. Software engineering is a broad term and encompasses a wide variety of tasks, including the design, constructing, testing and maintaining software systems.

When attempting to judge the software engineering methods and techniques involved in a software process, we need to give consideration to the type of application that is being developed. Therefore, we will never be able to definitively define the best software engineering technique, the characteristics of the best code or the traits of the best software engineer. We must find ways of measuring the process that are applicable across the broad range of methods and techniques.

Although the progress of software engineering lags behind hardware development, we have still seen huge progression since the 1960s. It is with this rapid growth, as well as the increasing pressure for growth, that the need arose to measure and assess the process. Before we determine the best tools for measurement, let us first look at the process itself.

SOFTWARE ENGINEERING PROCESS

The activities involved in the process range from the development of the software from scratch or making changes to an existing system. A software process must include 4 activities

1. Software Specification
2. Software design and implementation
3. Software verification and validation
4. Software evolution

While this is a very broad description of the process, it gives us an idea of the steps involved for every project, regardless of any specific details. We will now briefly look at some software process models. Each model represents a process from a specific perspective, Once again, this is a broad and generic analysis. These models are adapted and extended in reality. The most common of these models are the Iterative Development and Agile model. The four phases of the Iterative Development model include

1. Inception – making a business case for the system
2. Elaboration – further analysis of the problem domain and start developing project plan
3. Construction – this phases is incrementally visited for each increment, filling in the architecture with code
4. Transition – delivering the system

The agile method is based on the incremental and iterative approach, where new and updated versions of the systems are released to customers every few weeks. This model focuses on close customer interaction, a system that is able to accommodate change and is simplistic. Two types of agile methods are Scrum and Extreme Programming.

MEASURABLE DATA

The questions 'how?' and 'what?' are two important words to start with when considering the measurement of the software engineering process. Software metrics are numbers, tools and analysis that are used to measure performance in software engineering (Fenton & Neil, 1999) These metrics are needed to improve the way we monitor, control and predict various attributes of the software engineering process. The use of these metrics dates back to the 1960s when the primary form of measurement was LOC (Lines of Code). The lack of effort given to developing measurement tools meant that the functionality and complexity aspect of the code was overlooked. Since the 1960s we have seen huge progress from this very basic approach. However, while the amount of academic work on the subject has blossomed, the application of this in industry is far behind. R.L Glass stated in an article for The Journal of Systems Software in 1994 that "What theory is doing with respect to measurement of software work and what practice is doing are on two different planes, planes that are shifting in different directions." (Glass, 1994)

While the question 'why measure software engineering?' might seem to have a more obvious answer – to keep track of the software's progress, streamline the process, find efficiencies and improvements – we must also recognise some contrasting opinions. Stephen A.Lowe discusses the relevancy of metrics in software engineering in his article. He suggests most of these metrics that have been development are irrelevant and we should measure the success of a project by the customer happiness it creates and business value it delivers. (Lowe, 2018) However, this is a far more subjective and difficult to measure result.

Finally, when using these metrics we must remember that when it comes to software development everything produced is almost entirely unique, a 'snowflake' as Lowe describes it. This means that metrics are only comparable within the project itself (i.e at different phases) but not comparable externally.

Stephen A.Lowe discusses nine metrics that are applicable for use in industry today in his article "9 metrics that can make a difference to today's software development teams".

1. Leadtime – how long it takes to go from forming an idea to completing the final software.
2. Cycle time – the length of time it takes to change a software system and deliver that change.
3. Team velocity – how many units of software the team typically completes in an iteration.
4. Open/Close rates – how many production issues are reported and closed within a specific time period
5. Mean Time between Failures
6. Mean Time to recover/repair

7. Application crash rate – how many times the software fails divided by the amount of times it was used
8. Endpoint incidents – the amount of endpoints that have been impacted by a virus infection over a given period of time
9. MTTR – mean time to repair – the time between finding a security breach and delivery a working solution

Kan lays out in his paper *'Metrics and Models in Software Quality Engineering'* that there are three important measures required in order to effectively analyse the software engineering process: product metrics, process metrics and maintenance quality metrics. (Kan, 2003)

MEASUREMENT OF THE PRODUCT QUALITY

The metrics discussed in Kan's paper to assess product quality include

- Mean time to failure (MTTF) – measures the time between failures
- Defect density – measures the defects relative to the software size
- Customer problems
- Customer satisfaction

MTFF and defect density – These two approaches are the two key metrics for intrinsic product quality. There are issues with both however. With the MTTF, gathering data about time between failures is very expensive, requiring recording the occurrence time of each software failure to a high degree of accuracy. The defect rate metric on the other hand requires knowing the software size, often expressed in thousand lines of code (KLOC) or in number of function points. The Lines of Code (LOC) metric sounds more simple than the reality. The issues are caused by the difference between physical lines and instruction statements, comments and differences among languages.
Customer problems and customer satisfaction - The customer problems metric is usually expressed in terms of problems per user month (PUM). The numerator counts the true defects and non-defect-orientated problems for a time period and the denominator is the total number of license-months of the software during the period. This metric can be useful as it includes usability problems and lack of clarity for users. However, we do see an issue arising when the business is doing well and the denominator is increasing rapidly. This can skew the metric and underestimate the need to

reduce customer complaints. Customer satisfaction metrics are more simplistic and often measured by customer surveys.

MEASUREMENT OF PROCESS

This aspect of the measurement process varies greatly among different companies from approach to how thorough their approach is. The in-process metrics include the following

- Defect density during formal machine testing
- Defect arrival pattern during formal machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

Defect density during formal machine testing – This metric is a good indicator of quality and useful for release-to-release comparisons. If the defect rate is the same or lower than that of the previous release or similar products and the testing for the current release did not deteriorate than the quality perspective is positive.
Defect arrival pattern during formal machine testing – Analysing the pattern of defect arrivals or times between failures can give us information that we might not necessarily have picked up on from the total defect density. The overall aim of this metric is to get the defect arrivals to stabilise at a very low level or have the times between failures far apart.
Phase-based defect removal pattern – this pattern is similar to the defect density metric but it includes the defects at all phases of the development cycle such as the design reviews.
Defect Removal Effectiveness (DRE) – this formula is defined as defects removed during a development phase over defects latent in the product. The denominator has to be estimated by defects during the phase + defects found later. It can be used at all stages of the development process.

MEASUREMENT OF MAINTENANCE QUALITY

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix backlog and backlog management index (BMI) – the fix backlog measurement is related to the rate of defect arrivals and how quickly solutions

for these become available. The BMI is the number of problems closed during the month over the total number of problem arrivals during the month.
Fix response time – The ability to meet deadlines when fixing errors is a crucial part of measuring the software maintenance process. In some critical situations a business may not be able to operate until the software product is fixed in which case software engineers may have to work around the clock to provide a solution. The usual calculation for the metric is the mean time of all problems from when they are opened to when they are closed. In the case where there are outliers the median is used.
Percent delinquent fixes – For each software fix, if the turnaround time greatly exceeds the required response time then it is classified as delinquent. The formula used excludes open problems and is therefore not a metric for real-time delinquent management.
Fix quality – This metric keeps track of all fixes that turned out to be defective themselves. Defective fixes can be detrimental to customer satisfaction. The metric is simply the percentage of all fixes in a time interval that are defective. While all data relating to this issue is extremely important to collect, there is an argument against using a percentage with this metric. It can be skewed if the denominator is large due to the software having a lot of defects and therefore requiring a lot of fixes.

COMPUTATIONAL PLATFORMS

While it is great to have formulas to apply to data, it is hugely inefficient to manually collect and analyse data. Therefore, we need platforms to do this work for us. Philip M Johnson discusses some of the ways of gathering data from software engineers. He discusses the issue of observational bias whereby those collecting the data only collect what is easily available.

PERSONAL SOFTWARE PROCESS (PSP)

PSP enables software engineers to assess their progress and improve performance by tracking their predicted and actual development code. It was based off a previous process, the Capability Maturity model (CMM). PSP was created by Watts Humphrey who wanted software engineers to acquire a disciplined and effective approach to writing programs. (Humphrey, 1995) PSP is divided into certain stages. The baselines process PSPO can be split into 3 phases – planning, development and a post-mortem. Once sufficient data has been recorded and analysed, the programmer can analyse their own performance and manually input data. This led to accuracy issues due to the over-reliance on manual entries. Key metrics monitored by PSP include:

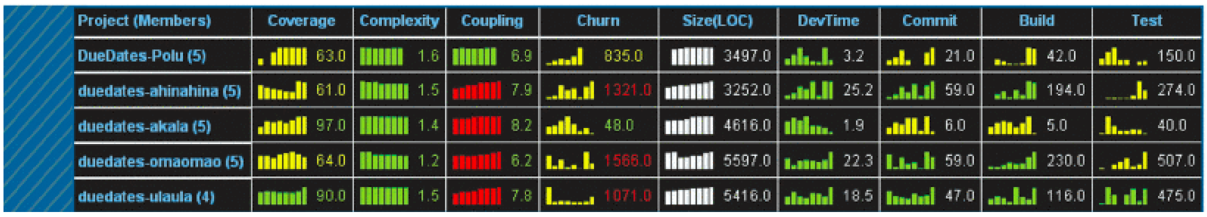
productivity, reuse percentage, defect information and earned value metrics. (Humphrey, 1995)

LEAP TOOLKIT

The LEAP (Lightweight, Empirical, Anti-measurement dysfunction and Portable software process measurement) toolkit was an advancement on the PSP. This new platform still required a certain level of manual input and therefore still open to human error. The Leap toolkit provided a further level of analysis that wasn't available with PSP such as regression analysis. It also automated and normalised data analysis. It is portable as it creates a repository of personal process data that developers can keep with them as they advance through different projects. It had its own issues however such as placing a burden on the data analyst to design and implement a new Leap toolkit for each new task.

HACKYSTAT

Since the requirement of significant manual data entry was still a major issue, the University of Hawaii developed a data collection tool called Hackystat. In Johnston's paper "Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined" the new and improved data analytics platform is discussed. Hackystat requires the development of client-side sensors that attach to development tools and automatically collects metrics such as effort, size and defect. Hackystat focuses on unobtrusive data collection. A common issue faced by developers is the interruption from having to record what they have already produced. The metrics are collected by sensors which then send this data to the server at regular intervals. From this data many metrics can be calculated such as the amount of developer effort spent on a given module, distribution of unit tests across a module, their invocation rate, lies of code written in a given module per day etc. (Johnson, 2003)



Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

Figure 1. A Software ICU (intensive care unit) display based on Hackystat. The Software ICU assesses a project's health both alone and in relation to other projects.

VISUALISATION OF DATA

GITHUB

Github is a web-based-version-control and collaboration platform for software developers. It is an open-source platform where managers can analyse developer data and measure the software engineering progress for free. Github provide the users with the ability to visualise data. Every Github member has a contribution heat map on their homepage which displays the total number of commits from the member over a specified timeframe.

The REST API is also a feature of Github which shows the number of commits made to the project, the number of developers working on it and how often each developer contributes. The data obtained from this can be accessed using other languages such as R to create visualisation data for managers to interpret.

CODE CLIMATE

"Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous and ubiquitous" (Codeclimate.com, 2018). Code Climate is just one of a number of systems that have built on the foundation of Hackystat to provide an automated data collection service. Code climate allows for code to be reviewed to ensure it is easily understood, it doesn't repeat itself, it can be modified and maintained with ease and is reusable. It offers many important produce measurement metrics and enables companies to ensure their code is of high quality and standard. (Code Climate 2018) The product performs automated analysis of software code and can be hosted on-site in a company data centre or accessed as a cloud-based service. Code Climate can also be used in conjunction with Github, where it will show a user data like their code coverage, technical debt and a progress report. (Carpenter, 2018)

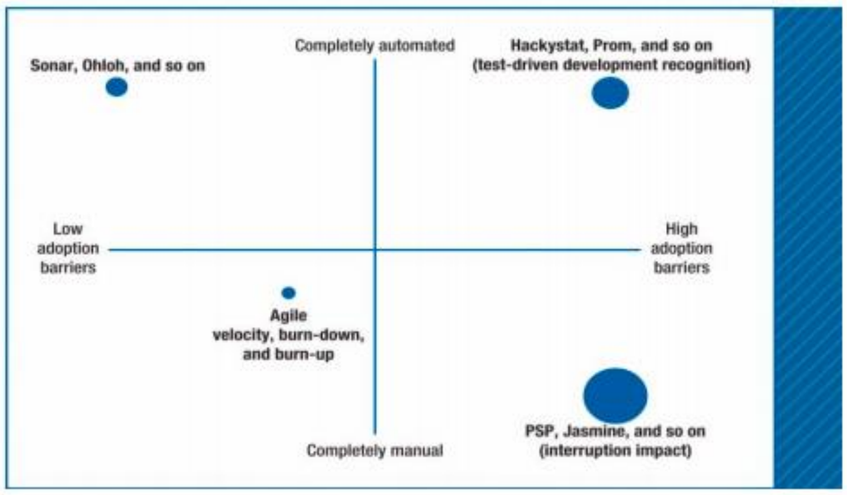

Sonar, Ohloh, and so on
Completely automated
Hackystat, Prom, and so on (test-driven development recognition)
Low adoption barriers
High adoption barriers
Agile velocity, burn-down, and burn-up
Completely manual
PSP, Jasmine, and so on (interruption impact)

Figure 2.
A classification for software analytics approaches, including automation, adoption barriers, and the breadth of possible analytics the approach supports (indicated by the circles' size). In the parentheses are analytics that would be difficult to implement with techniques or technologies in the other quadrants.

ALGORITHMIC APPROACHES

So far in this paper, I have discussed what data is available to us, how we can interpret this at a basic level and how we collect the data. I am now going to discuss the analysis of this data at a higher level and the algorithmic approaches developed to aid in this analysis.

PROCESS QUALITY INDEX

The Process Quality Index (PQI) has five components

1. Design quality as a ratio of design time to coding time
2. Design review quality as the ratio of design review time to design time
3. Code review quality as the ratio of code review time to coding time
4. Code quality as the ratio of compile defects to a size measure
5. Program quality is the ratio of unit test defects to a size measure

These five elements are then multiplied together. The output is a figure between 0.0 and 1. To interpret this briefly, products that produce values below 0.5 are considered to be bad quality. This is a very simplistic algorithmic approach and there are far more complex ones available.

COMPUTATIONAL INTELLIGENCE

Computational Intelligence (CI) is a tool used by the software companies to gain a more in-depth insight into the data provided by the software engineers (IEEE, 2018). CI is the ability of a machine to learn a specific task from a set of data and/or from experimental observations. It provides us with a new way to solve modern, complex problems where traditional mathematical techniques are not adequate anymore. This could be the case when the process is too complex, when it is of random nature or where there is certain amount of uncertainty during the process. There are five primary components to CI

1. Fuzzy logic – This is an approach to computing that alters the blunt 'true' or 'false' approach to Boolean logic. It is employed to handle the concept of partial truth and has a wide array of applications.
2. Neural Networks – NNs are not algorithms themselves, but rather they provide a framework for machine earning algorithms to work together and process data inputs. (IEEE, 2018). There are three components to the networks. The first processes the information, the next sends the signal and finally one to control the signals when sent.
3. Evolutionary Computation- this is a family of algorithms for global optimisation based on the theory of natural selection. Applications of this are found where traditional mathematical techniques are not adequate to solve a wide range of problems
4. Learning Theory – this helps us understand the emotional and cognitive effect on human reasoning. It attempts to make predictions off its analysis
5. Probabilistic Methods - tries to evaluate outcomes of systems defined by randomness, like the real world. It provides possible solutions to a reasoning problem, based on past experiences and knowledge

CI has clear benefits to it as a means of measuring and analysing the software engineering process. These models are able to handle more data than humans can comprehend. Their ability to manage incomplete data, human error and deal with vast amounts of information should lead to better optimisation and decision making going forward.

MULTIVARIATE DATA ANALYSIS

The data that we are dealing with is complex and can have many variables associated with a point therefore it can be described as multivariate and we can look at some the multivariate analysis techniques to gain a better insight into the data (Houlding, 2018).

When dealing with algorithmic analysis, there are two main subsections: supervised methods and unsupervised methods. Supervised learning occurs when there is an input and an output and algorithm is used to map the input to the output. Supervised methods are based off the idea that after ‘trial and error’ the output will be accurate enough to compute output for future input values. Supervised learning gets its name from the fact that we know the answer expected from the original input data. As a result we are able to monitor that the machine algorithm is predicting correctly and if not changes can be made to fix it. (Brownlee, 2016). Examples of supervised learning algorithms are Linear Discriminant Analysis and K-Nearest Neighbours.

Unsupervised methods, on the other hand, do not have a teaching process. We must discover the structure of the data, using only the data that we are given. Unsupervised learning methods include clustering where the input data is placed into distinct groups based on similarities and differences for example the K-Means clustering algorithm (Brownlee, 2016).

ETHICS

LEGALITY

In May 2018, the EU released the General Data Protection Regulation (GDPR) to replace the previous Data Protection Act (DPA). GDPR has greatly strengthened the protections surrounding data collection. Any European organisation that collects or uses personal data (data that relates to, or can identify a living person), are now subject to GDPR regulations. Any company found to be in breach of GDPR can be fined 4% of their revenue or €20 million. As a result of this regulation, any firm looking to measure their software engineering process must pay careful attention to the new legislation and ensure they are not in breach of these new rules. (Citizens Information, 2018)

This brings me on to the word ‘consent’. Speaking from personal experience, I think the new GDPR is a great idea for improving regulation around privacy. However, I find myself clicking the ‘I Agree’ button for every website I go on without reading the policy statements or terms and conditions. I strongly believe I am not the only one who does this, in fact I’d say that at least 70% of standard internet users do just the same. This leads me on to the question- does the consent form place all of the responsibility in my hands? Is this the only way forward, should I take my responsibility more seriously or should it still be made more clear to me what exactly my data is being used for?

PRIVACY

One of the biggest issues with the development of software engineering is the privacy concern. This topic is very controversial when defining what 'crosses the line'. What lengths will we go to in order to extract data from customers, employees or potential clients? The Facebook and Cambridge Analytica Scandal was a major political scandal in 2018 when it was discovered that Cambridge Analytics was collecting and using the personal data of millions of peoples' Facebook profiles with no consent. This ignited many discussions regarding the rules, regulations and responsibilities surrounding data collection. Facebook's actions, or lack of, plunged the company into the greatest crisis it had encountered in its 14 years. It showed the impact which negligence can have on a company. Despite the line being undefined and unclear, if and when the line is crossed the consequences are serious.

There are then the companies that don't necessarily make the headlines. The data collection that is treading dangerously close to this 'line' but has not quite crossed it and therefore remains under the radar. An example of where I think a company is going too far is Humanyze. Humanyze provide their clients with biometric badges that look like normal employee badges but with a catch. These special badges are equipped with radio frequency identification (RFID) and near field communication (NFC) sensors, bluetooth for proximity sensing, infrared to detect face to face interaction, an accelerometer and two microphones. The function of the Humanyze is to collect and analyse data to aid management in their strategic decision making. The badge tracks everything an employee does, where they go in the office, who they were interacting with, how long they are away from the desk and how many 'coffee breaks' they go for. While for some this could be interpreted as taking data collection too far there is also the argument that the company has the employees consent.

It is of inherent important that software engineers are aware of the wider responsibilities attached to their role. Professional engineers must conduct themselves in an ethically and morally responsible way. Principles like confidentiality, honesty and integrity are key to the successful performance of a software engineer (Sommerville, 2011).

CONCLUSION

This report has outlined the ways in which the software engineer process can be measured and assessed in terms of measurable data, the computational platforms available to do this work, the algorithmic approaches available and the ethics concerns surrounding this kind of analytics. While it is possible to measure software engineering to a certain extent, it is not an easy process. It can cause interruption and conflict while being expensive or time consuming. There is also no perfect approach, each having its own issues and degrees of inaccuracy. There are also the soft-skills required by a software engineer that are far more difficult to quantify. How can we measure their ability to communicate with their colleagues, their teamwork or management skills? We do not have any real metrics do analyse a developer's ability to be creative or innovative. While progress has been made towards measuring and assessing the software engineering process we are still a long way off seeing the whole picture through our objective, quantitative approach.

BIBLIOGRAPHY

Brownlee, J. (2016, March 16). *Supervised and Unsupervised Machine Learning Algorithms*. Retrieved from Machine Learning Mystery: https://machinelearningmastery.com/supervised-andunsupervised-machine-learning-algorithms/. [Accessed 23 October 2019]

Carpenter, W., 2018. *Code Climate: How it Works and Makes Money.* [Online] Available at: https://www.investopedia.com/articles/investing/022716/code-climate-howit-works-and-makes-money.asp#ixzz5XCtdZq26 [Accessed 21 October 2019]

Code Climate. (2018). *About Us: Code Climate*. [Online] Retrieved from Code Climate:
https://codeclimate.com/about/ [Accessed 25 October 2019]

Fenton, N. & Neil, M., 1999. *Software metrics: successes, failures and new directions*. The Journal of Systems and Software, Volume 47.

Glass, R., 1994. A tabulation of topics where software practice leads software theory. *Journal of Systems Software*, Volume 25.

Humphrey, W. S., 1995. *A Discipline for Software Engineering*. Addison-Wesley

IEEE, 2018. *What is Computational Intelligence?.* [Online] Available at: https://cis.ieee.org/about/what-is-ci [Accessed 27 October 2019]

Johnson, P., 2003. *Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined*. 25th Annual Conference on Software Engineering.

Johnson, P. M., 2013. *Searching under the streetlight for Useful Software Analytics*. IEEE Sofware, 30(4).

Lowe, S. A., 2018. *Why metrics don't matter in software development (unless you pair them with business goals)* [Online] Available at: https://techbeacon.com/why-metrics-dont-matter-software-developmentunless-you-pair-them-business-goals [Accessed 22 October 2019]

Sommerville, I., 2011. *Software Engineering*. 9th Edition ed. s.l.:Pearson Education.