MEASURING ENGINEERING

How Software Engineering is Measured and Assessed

Abstract

This report shall discuss the measurement and analysis techniques using in the Software Engineering industry, including discussion on measurable data, the computational platforms, the algorithmic approaches and the ethnics.

Bríd O'Donnell

Odonneb4@tcd.ie

MEASURING ENGINEERING

INTRODUCTION

The following report will explore the following areas and how they impact the analysis of software engineering:

1. Measurable Data
2. Computational Platforms
3. Algorithmic Approaches
4. Ethics

According to Ian Somerville, the definition of software engineering is "an engineering discipline that is concerned with all aspects of software production". Software production consists of these four main activities:

1. Deciding on the software specifications
2. Designing and implementing the software
3. Testing, verifying, validating the software
4. Maintaining software

Thus, when creating a thorough system for measuring software engineering, accounting for these four main activities is crucial.

Furthermore, the system must reflect the implementation of these activities, as there are several approaches different organisations take when developing software. These include waterfall, agile and iterative. Agile methods, which mainly split into scrum and extreme programming, is the most popular approaches currently. As a result, I will focus on measuring methods based on agile methods.

The term Software Engineering was only coined in the 1960s; however, since then, the industry has experienced exponential growth. Like all industries, trends to introduce quality control and improvements, to increase efficiency, have become common. However, despite academic research and publications relating to the measurement of software engineering, many companies in the industry still struggle to implement an effective measurement system, so that they can identify ways to manage costs and investments and increase productivity. Companies need to distribute their resources wisely, and this need is only growing as more and more companies, both old and new, are embracing software engineering as a critical component of their businesses.

MEASURABLE DATA

For any industry, data is at the core for any efficient, systematic improvement of quality and processes. A large amount of data is easily collected through the software

engineering process; however, this data may not be relevant or comparable to produce large-scale quality improvement.

TYPES OF METRICS

There are three types of metrics that software engineering can be measured by; Project Data, Process Data and Product data. These metrics can be broken down into entities and attributes. The attributes are further divided into internal attributes and external attributes. Internal attributes are “measured directly by examining the product on its own irrespectively of its behaviour" while external attributes relate to the product links to its environment. Both types of attributes have advantages and disadvantages; internal attributes are easier to collect. However, they are challenging to interpret, while external attributes are more subjective and require manual inputs. To create a thorough measurement system, you must combine both internal and external attributes. (Zenos & Stavrinoudis)

The most accessible data to collect and measure is project data. The metrics for measuring a project only concern the resources of the project and can be measured in the same way as projects are measured in other industries. The three main entities which are analysed are personnel, tools and environment. Although this metric is not unique to software engineering, from an organisational perspective, it is just as essential to have. (Zenos & Stavrinoudis)

The next type of metric that is relatively easy to measure is process metrics the process of creating a software product. For this metric, you are measuring each main activity of software engineering; specification, implementation, validation and maintenance, searching for the best practices or the problems that arise during the programming. The internal attributes always include time and effort, along with other measurements, for example, the percentage of test coverage during the testing staging. (Zenos & Stavrinoudis)

Finally, the most complex metric to define and measure are product metrics, also known as software metrics. These metrics relate directly to the product, its quality, deliverables and manuals. (Zenos & Stavrinoudis)

TYPES OF DATA

Here are some of the most straightforward data to collect:

1. Lines of code
2. Number of commits
3. Code Churn
4. Cyclomatic complexity
5. Test Coverage

However, all these metrics are flawed and should not alone be used to measure a software engineer.

The first data metric, lines of code, creates a big problem as it is generally best practice to create a program with as few lines as possible. Therefore implementing a metric which would incentivise engineers to write more lines won't necessarily lead to better, more efficient programs.

The next metric, number of commits, is equally opaque in measuring software engineering as lines of code. A commit adds the latest changes to the source code to the repository, making these changes part of the head revision of the repository. Frequency of commits doesn't correlate with the skill and ability of a software engineer. It does prove that the engineer is active. However, best practice for a software engineer is to commit consistently, so again this is a flawed metric.

However, code churn, the metric measuring change volume between two versions of a system, defined as the sum of added, modified and deleted lines, can be useful for identifying problems in a project. Typically, a high level of code churn indicates a less stable project. Regardless, code churn without context does not reflect the performance or the productivity of a software engineer. For example, a high rate of code churn in a project could be attributed to a poor software engineer or indecisive product owners who frequently change the requirements for the project's features.

Cyclomatic complexity measures the complexity of code by seeing how many different paths one can take. This measurement is extremely useful in certain circumstances. If you have two programs during the same task, the one with the smaller cyclomatic complexity measurement should be taken as the better code. It is claimed that complex code is at a higher risk of getting bugs and errors. Therefore fewer complex solutions are desired. However, software engineers can't always reduce the complexity of their programs due to the size of the program, and that is not the software engineer's fault.

Test Coverage is an excellent metric for measuring software engineer and performance. Coverage is the percentage of the source code of a program which is executed when a test suite runs. The ideal is 100% and any team that operates a Test-Driven approach, which is common among agile users, would base a large part of their progress on test coverage. However, like most metrics, it doesn't guarantee certainty; 100% test coverage program could still feature errors and bugs. Equally, code with no tests can have no errors or bugs. Thus you can't rely on test coverage solely.

CONCLUSION

Obstacles arise while trying to measure qualitative data for such critical attributes as quality, usability maintainability, as those attributes can be subjective and difficult to standardise. However, approaches, such as identifying correlations between these attributes and other types of data that can be collected and quantify can overcome these problems. One such correlation is the number of errors and the quality of implementing the code.

Another issue when measuring software engineering data is the human aspect, which is very important for organisations uses the agile method for software development as agile

prioritises the human element in the process. Thus, a significant effort should be brought when designing a measuring system that it does not disrupt the engineer. The system should be unobtrusive to the engineer and encourage productivity, not diminish it.

COMPUTATIONAL PLATFORMS

The data discussed above, once collected, must then be analysed to be valuable to organisations. In recent years, rapid analysis has become increasingly important, as it allows for faster decision making and responses to problems that can sustain and increase productivity and keep organisations competitive. Thus, many computational platforms have been created and released. However, due to structural differences in organisations, such as team layouts, types of projects and the individual software engineer, the platforms and metrics used differ vastly from organisation to organisation.

Key platforms that broadly cover the different aspects and track-offs of monitoring include:

1. Personal Software Process
2. Leap Toolkit
3. Hackstat
4. Code Climate

PERSONAL SOFTWARE PROCESS (PSP)

PSP .was created by Watts Humphrey and is a structured software development process that is designed to help software engineers improve their performance and track their coding. It was built in response to the high number of defects that were commonly occurring in software projects. It also increases the overall quality of the software. Unfortunately, PSP requires manual input from engineers, obstructing them from doing productive coding on a project. Furthermore, it relates to human judgement, which can be subjective and prone to error.

LEAP TOOLKIT

Leap Toolkit was created in response to the failures of PSP, specifically the manual input feature. Leap automates and normalises data analysis which reduces the time software engineers dedicated to tracking their work efficiently. Leap is an acronym of its design principles; lightweight, empirical, anti-measurement dysfunction and portable, which compliments the principles of the agile method. Software engineers still need to manually input data, but Leap does not require the same level of effort as PSP. An exciting feature was the creation of a repository that engineers could take with them from project to project, fulfilling its portable principle.

HACKSTAT

Hackystat utilises sensors in development tools which collect data and sent them to a server for analysis. Importantly, hackystat collects data unobtrusively, with no manual

input from engineers. It also collected data in real-time and works for tracking groups. One obstacle to the industry adopting hackystat is the constant surveillance of software engineer's work and who had access to that surveillance. This issue will be further discussed later in the ethics section of this report.

CODE CLIMATE

Code Climate is a software company which develops specialised software for analysis of the software engineering process and improvement of code quality. Despite being an expensive option, it is popular among engineers. It combines test coverage and code quality to give a comprehensive overview of a software project for a team of software engineers. Some of its most impressive features are activity feed, team sharing, GitHub integration and a security dashboard. The differences between Code Climate and Hackystat is that Code Climate focuses mainly on data that is considered product metric like test coverage and not data on the engineer's behaviour, eases developer's privacy concerns.

CONCLUSION

Along with the platforms explored above, there are many more services and platforms available, including DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity and Sonar, each with specialisations and track offs. Certain platforms favour metrics on engineers' behaviour practices, at the cost of high overhead in time and effort to collect the data. Other platforms require less manual input or less frequent surveillance, but their analysis is less comprehensive. With such a full spectrum of track-offs and benefits, the choice of platform for an organisation should reflect the organisation's needs and principles.

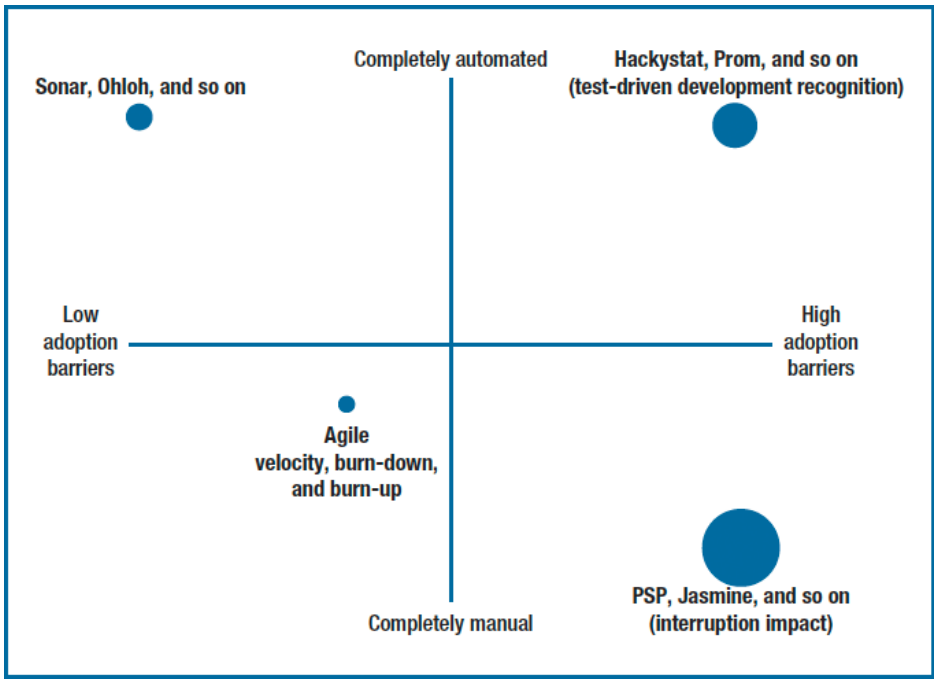


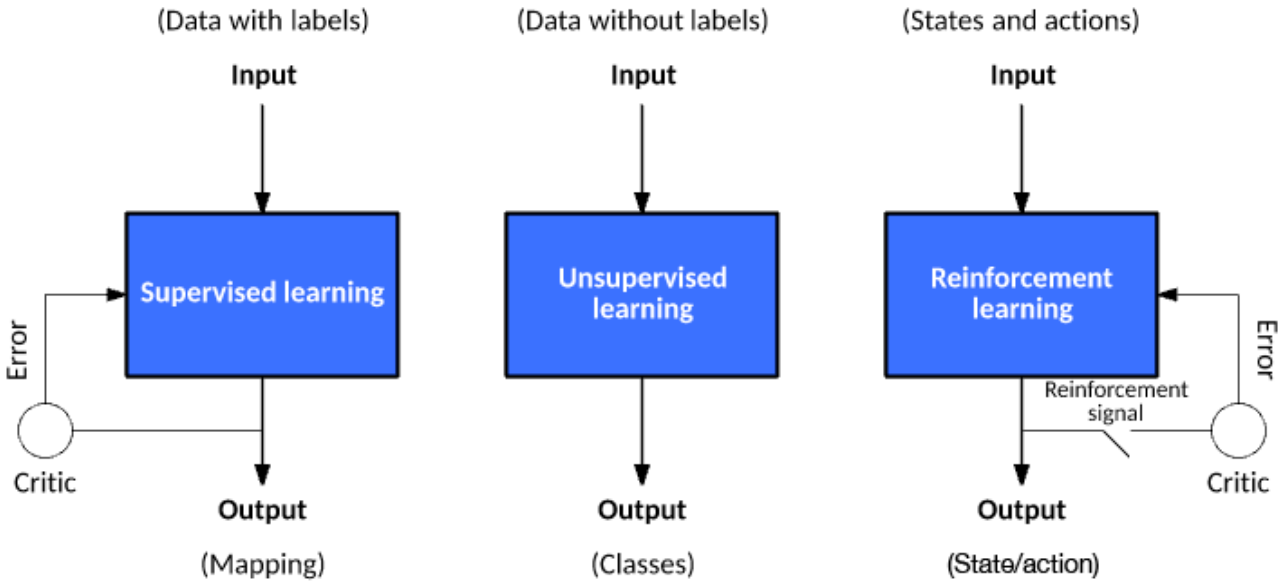
Figure 1: classification of software analytics platforms from "Searching under the Streetlight for Useful Software Analytics."

ALGORITHMIC APPROACHES

A significant trend in the analysis of software engineering is the automation of the work thanks to machine learning. Machine learning is derived from the concept of pattern recognition and fundamentally grounded in the theory that computers can learn without being programmed. Machine Learning and artificial intelligence are already popular, utilised for many frequent tasks; for example, machine learning is used to recommend videos to YouTube users. As such, machine learning has enabled engineers to automate their analysis and allow for rapid decision making.

Machine learning can be divided into three categories.

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning



SUPERVISED LEARNING

For supervised learning, there is an input and output variable where an algorithm is created to map the input to the output. The theory here is that after trial and error, the algorithm should be able to compute the corresponding correct production for new inputs. This type of algorithm is "supervised" because the algorithm undergoes training and only go into production when they reach a certain level of accuracy.

Examples of supervised learning include decision tree analysis, k nearest neighbours, regression or logistic regression.

UNSUPERVISED LEARNING

Unsupervised learning has only input data and no output data. The goal of unsupervised learning is to identify the distribution of the data and model around that. The issue with unsupervised learning is that there are no correct results, and we must rely on the algorithm.

Examples of this learning are K-Means Clustering, the Apriori Algorithm and Principle Component Analysis.

REINFORCEMENT

Reinforcement relies on trial and error approach and uses feedback to improve its action. This is similar to how humans learn; however, it is difficult to achieve as it requires a large amount of memory, especially for more complex problems.

Examples of reinforcement algorithms are Markov decision, dynamic programming and Monte Carlo method.

CONCLUSION

The rise of machine learning has dramatically helped with monitoring the software engineering process. As it continues to be utilised, the growing scale should create even more accurate analysis, especially with the supervised learning technique. However, a significant concern machine learning presents is the lack of human perspective. Machine learning is often considered a black box, and current machine learning algorithms barely think the same way humans do. Thus while they may report accurately and appropriately 99% of the time, it may not react correctly to a variable that the algorithm has never seen before. Granting too much sovereign to these algorithms could cause some unfortunate circumstances that had a human element been involved could have been adverted.

ETHICS

The act of measuring software engineering entails a certain level of surveillance, which brings ethics and legality into the equation. Organisations reverse the right to monitor their work; in fact, this measurement is necessary to stay competitive.

However, several factors make measurement and analysis of software engineers a little more complicated than the monitoring systems for other professions. Firstly, a software engineer's job is more multifaceted than a regular office job, even though software engineers are treated similarly to standard office workers. A software engineer must have an extensive range of skills and measuring the combined work those by different skills is difficult. Furthermore, there are many standards and expectations that software engineers must fulfil, such as scalability, compatible, ease of use, tested, secured, debugged, on-time and inexpensive. Thus, it is understandable why software engineers had different monitoring and measurement systems to other workers.

SOFTWARE ENGINEER'S OBLIGATIONS

According to Merriam-Webster, ethics is "the discipline dealing with what is good and bad and with moral duty and obligation". With regards to measuring software engineering,

this subsection will discuss the ethical obligation a software engineer has towards society.

Software engineering is a crucial profession required for the continued functioning of our modern digitised society. The daily work that nameless software engineers do in any company can have massive impacts on people's lives and society as a whole. Software used to score student's tests or to measure someone's credit score can have enormous implications on someone's lives and if the software has a bug or cybersecurity gap or even a fundamental flaw to how the software interacts with humans. The power that engineers have mean that it is critical to hold them to account and make their work transparent.

The Association of Computing Machinery (ACM) introduced a code of professional practices, containing eight core principles on ethnical behaviour and decisions for software engineers. However, the code is not legally enforceable, and recent research from North Carolina State University shows that the code has little effect on engineer's work. Legal matters will be discussed in the next subsection. Still, governments and law-making bodies are too slow, due to their size and structure, to keep up with the fast phase development of software engineering and cover all the issues that software may cause. Therefore the only way software engineers can be held account currently is by the organisation. Thus, monitoring, measuring and assessing software engineers is vital for everyone's sake.

It is essential to acknowledge that software engineers are not actively designing unethical software. Software engineering is a difficult job before taking ethnic into account. There is no right answer when designing software, and often it is incredibly difficult to understand the impact software will have before it released to society. Regardless, monitoring software development can give insights into the best practices and prevent future issues.

LEGALITY

There is a legal aspect which impacts the monitoring of employees. The General Data Protection Regulation (GDPR) is the EU's landmark data regulations, released in May 2018. The critical impact GDPR has had is that organisations can only hold any person's data is the individual's consent. Furthermore, there are higher regulations and limits on the data they can hold and how it is used and stored, increasing transparency and individual protections. Fortunately, GDPR is also enforceable; a breach of the law will result in a 20 million euro fine or a fine of 4% of their revenue.

GDPR has inspired an intense desire for more data regulation, and similar aggressive protection bills are being formulated in other justifications, for example, the California Consumer Privacy Act.

Compliance to these new laws has caused high costs for any companies that collect data, which is most in modern times. However, these laws are consumer-centric and don't directly address workplace monitoring. Interestingly, both GDPR and CCPA protect

personal data, which has an interesting on workplace monitoring. Even though conducting personal affairs in the workplace is considered a cybersecurity risk, many employees do from time to time use their work devices for personal reasons. The data collected from those activities are considered private, which organisations must treat differently to data such as the number of commits you made. This creates doubts about the legality of something like the active badge location system.

Furthermore, since employees enter into legal contracts with employers, many of the rights that these laws outline are forfeited by employees to ensure employment. While software engineers are in high demand, not everyone is in the position to negotiate a beneficial contract that protects their data rights.

DATA ACCESS AND USAGE

One of the significant rights that have emerged from new data laws is the right to control who has access to your data. This new right is relevant to employees, as data on your personal info or even your work can fall into the wrong hands if organisations are not careful. Data that is open to a whole team, as well as managers, can cause tensions within organisations as they compete with each other as they compare themselves to each other.

A logical extension of this right to understand who their data is disclosed to is that employees should have a right to know how their work is assessed. The implication of this extension means companies should be transparent in their metrics, and the platforms they are using to measure software engineering quality. Furthermore, this can allow for feedback from engineers on the metrics, which should increase the accuracy of the measurements and analysis.

Another part of measuring and analysing software engineering is that it must be impartial and fair, which can motivate engineers to pursue improvement naturally. An employee must trust that their data wouldn’t be used in a discriminatory way, provided them from accessing opportunities and promotions.

IMPACT ON EMPLOYEES

Some would also argue that constant surveillance on your workforce will have a negative impact on your employees’ productivity. The continuous monitoring, assessing and essentially judging of a software engineer's work can put them under pressure and cause stress, resulting in an increase in the number of mistakes.

Aside from that aspect, constant surveillance can reduce innovation in the workplace. Since employees under surveillance and scrutiny are focusing on being as efficient as possible, there is no room for the employee to take a risk and to try something new. Innovation is defined as "anything that is new, useful, and surprising” by author and innovation expert, Drew Boyd. Innovation is equally as important as efficiency, if not arguably more so, for companies to stay competitive and grow, especially in the tech industry.

This leads me into the most critical aspect of measuring a software engineer. The process of measuring their work must not impede on the software engineer or their work. That means that firstly the software engineer must not be dedicated too much time that could have been spent on working on the actual software problem on recording data to measure their work. However, as mentioned above, the measuring system must not mentally impact the engineer and weaken their productivity. However, that aspect depends genuinely on the workplace culture of the organisation.

In my own opinion, I have many issues with surveillance, and I do believe the mass data collection that occurs in the workplace, normalise the data collection that happens in our lives from the likes of Google and Facebook. Therefore, I do believe that new laws should be created, especially within the workplace, that reduces the ability to collect data collection, for all professions to protect employees. Moreover, I believe those actions are more enforceable than actions taken directly on consumer data collection. The line between surveillance and data collection has been profoundly blurred, and I think that it will be one of the most defining issues of our time.

CONCLUSION

This report has discussed the importance of measuring and analysing the software engineering process. It has also outlined the ways in many ways the software engineering process can be measured. These metrics have their limits and the level of detail and context that the metrics contain, vary depending on the level of manual input or surveillance. These track-offs lead to a diverse range of platforms available to automate the analysis process. Analysis of software engineering has also developed in recent years thanks to machine learning, and continued development should allow for greater understanding of the individual software engineering process. Finally, the report examined the ethical implications of software engineering monitoring. These issues largely depend on the organisation monitoring the process, however more significant government intervention may be needed to contain the massive rise of data collection and surveillance, both in the workplace and as a consumer.

In conclusion, it is unsurprising that there is no standardised method for measuring software engineering. However, organisations should actively work on their measuring and assessing system to ensure it has a positive impact on its software engineers and suits the needs of the organisation and its engineers.

BIBLIOGRAPHY

Brownlee, J. (n.d.). *Supervised and Unsupervised Machine Learning Algorithms*. Retrieved from Machine Learning Mystery: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Grambow, G. (2013). Grambow, G., Oberhauser, R., & Reichert, M. (2013). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. *International Journal on Advances in Software*, 213-224.

Johnson, P. (2013, July-Aug). Searching under the Streetlight for Useful Software Analytics. *IEEE Software, 30*, 57-63.

Kan, S. H. (2003). *Metrics and Models in Software Quality Engineering.* Addison-Wesley Professional.

Sillitti, A. (2003). Collecting, Integrating and Analysing Software Metrics and Personal Software. *29th Euromicro Conference.* IEEE.

Xeno, M. N., & Stavrinoudis, D. (2008). Comparing internal and external software quality measurements. *Proceedings of the 8th Joint Conference on Knowledge-Based Software Engineering*, 115-124.