Seán Healy-16322679

Measuring Software Engineering

Abstract

The aim of this report is to consider the ways in which the software engineering process can be measured and assessed for quality. This will be discussed under a number of headings including Measurable Data, Computational Platforms, Algorithmic Approaches and Ethics after a brief introduction.

Introduction

In order to consider the ways in which the work of software engineers is assessed and measured, it's important to first understand what software engineering is. The Institute of Electrical and Electronic Engineers describes it as:

"*the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software*"

Software engineering is different to programming. Simply put, programming involves instructing a computer to do something with inputted data and provide an output. Software engineering, on the other hand involves being able to formulate and understand the requirements of a problem, plan out the solution using a variety of methods and execute it effectively while managing any issues that may arise.

The term 'software engineering' was first coined in 1968 at conferences organized by NATO. The discussion regarded the 'software crisis' of the 1960's which was the name given to the problems faced when developing large, complex systems at the time. The adoption of an engineering approach to software development was proposed with the belief that it would decrease the costs involved in developing software and lead to more reliable software.

Software engineering has had a huge impact on the world and how we interact with technology and even each other every day. Being such an integral part of our lives, it's essential to develop an understanding of how software engineering can be measured and assessed. The data and attributes for measuring software engineering will be discussed along with the available platforms and algorithms for such measurement. The ethics of this measurement will also be discussed in this report, but first we will look at why software engineering should be measured.

Measurable Data

Measurement is the process of quantifying or characterising an object or event so as to describe it in such a way that it can be clearly compared to other objects or events of the same type.

"*You cannot control what you cannot measure*" – Tom DeMarco

Data is an extremely important aspect of quality monitoring, improvement and feedback in all industries including software development. To effectively measure the work of software engineers, it's critical that the right type of data is collected. This data should be based on previously planned and agreed metrics which can be used as part of the measurement process and used to drive improvements in quality. Careful selection of data is of paramount importance given the amount of data which can be created day to day by software engineers, much of which may not be of help in assessing performance. Once data is collected it should be classified and stored in such a manner that it can be easily called upon and used as part of the measurement process.

There are many useful attributes by which the work of software engineers can be assessed through the collection and analysis of data, however it's also useful to use simple feedback and communication methods to obtain a better understanding of not just how an engineer is performing, but why they're performing at such a level. The following are straightforward methods as explained in 'The Hartford Business Owner's Playbook' which can provide valuable information for management on the performance of their employees:

- **Self-Evaluation:**
  Self-evaluation is an effective technique that allows management to find out how their engineers perceive the level of their own performance. Often times if an employee is underperforming, they will have an idea as to why that is or what specific area they're having trouble with. Simply getting feedback from a software engineer can show management what each person needs to focus on or needs assistance with, which in turn allows for positive improvement and better performance in the future.

- **Management by Objectives (MBO):**
  This a method which is also known as "management by results". It involves the planning of goals and objectives by an employee and their manager who jointly decide on what's expected and how performance will be evaluated. Management by Objectives provides employees with an understanding of their goals and allows them to participate in the process, fostering communication and teamwork.

- **360-degree feedback**
  360-degree feedback takes into account the feedback of an employee's performance from their circle of work colleagues within the company. It can include co-workers, managers and anyone else they work with. When analysing the feedback and opinions given regarding an engineer, a note is taken of positive and negative feedback and any similarities or trends that exist. It's also possible to identify areas that might be in need of further measurement and support.

These are some of the more general methods to get information on an employee's performance and while they are used in many industries, they are also extremely helpful in evaluating software engineers. However, given the complexity of work done by engineers, these methods alone are insufficient and a more quantitative approach needs to take place to gain a better understanding of the engineer's performance.

As explained by Ebert, et al. (2005) the areas to be measured in software engineering can be divided into three sections:

- Product
- Process
- Resources

Suitable software metrics can be applied to empirical criteria to obtain numerical estimates of how well each criterion has been met. A number of such criteria and measurement metrics are seen below:

Empirical Criterion	**Appropriate Metric**
Product-based evaluation	
Reliability	Mean time to failure.
Usability	Completeness of the documented product components.
Efficiency	Performance of response time of the considered systems.
Price-performance	Price level of the software compared to the current market level.
Process-based evaluation	
Project management	Capability maturity model evaluation.
At the date fixed	Average deviation from milestone dates.
Complaint management	Frequency of complaints per week.
Consulting	Mean time to repair.
Resource-based evaluation	

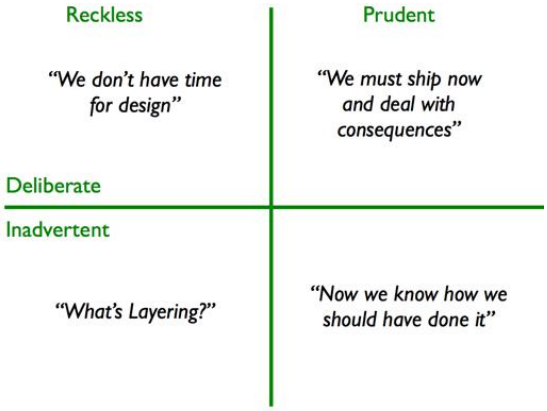
Teamwork	Provision of time as relation between the spent time and the necessary time for communication of every team member.
Professional Competence	Years of experience as level of developer skills.

The above metrics are useful when measuring the performance of teams, the success of projects and the quality of software, however individual developers can be assessed more closely using a number of specific metrics to measure their productivity. A number of commonly used metrics are discussed below:

- **Source Lines of Code:** This is one of the simplest metrics which can be used when assessing a software engineer's work. Counting the lines of original code produced by the engineer per time unit might seem logical, however this method equates more lines of code with a higher quality of work and greater productivity which is not necessarily the case. If a program can be effectively written in fewer lines of code then this is preferable, and the LOC method will be ineffective in assessing the work of the engineer.

- **Lead Time & Cycle Time:** Lead time is the time between the request by the customer for a solution and the final delivery of the finished product. Cycle time is the time between the order being taken for production or development and the project being completed. Both of these metrics can be useful in determining the efficiency of engineers in completing their work.

- **Technical Debt:** Technical debt reflects the implied cost of additional rework in software engineering caused by choosing an easy solution now instead of using a better approach that would take longer. Martin Fowler's technical debt quadrant in a useful visualisation which shows the ways in which technical debt can occur.



- **Number of Commits:** While regularly committing code shows that a software engineer is continuously working on a project, this method assumes that more commits on a project means more work being completed. It's possible that certain engineers simply wait longer and commit more work at once resulting in a lower commit count but the same

amount of work completed which is not taken into account by this method.

- **Code Churn:** Code churn is a measure of the amount of code change taking place within a software unit over time. It can be obtained without difficulty from the change history of a system, as is kept track of by version control systems. According to Nachiappan Nagappan and Thomas Ball (2005), using certain relative measures of code churn is "highly predictive of defect density" in software engineering. Examples of effective metrics are:

    - Churned Lines of Code / Total Lines of Code
    - Deleted Lines of Code / Total Lines of Code

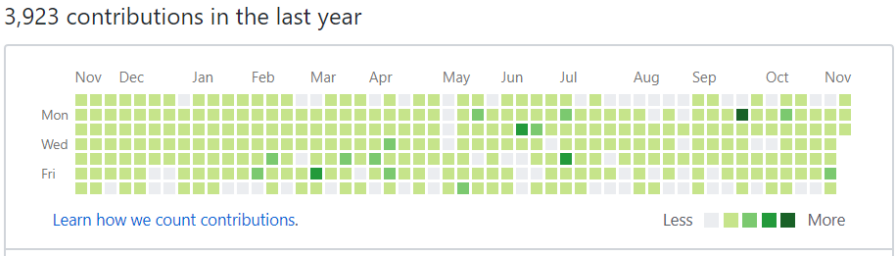
    It is expected that the larger the above proportions are, the larger the magnitude of the defect density will be.

Computational Platforms

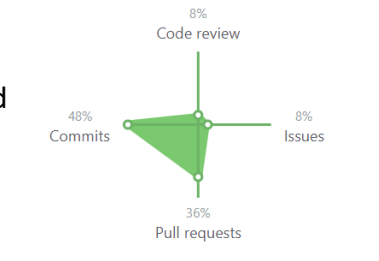
There are a number of platforms available to measure and assess software engineering, some of these platforms are free and others are provided for a fee and generally employed by companies. While data collection is an important part of the measuring process, the platforms with which that data can be analysed are essential in providing meaningful interpretation and insights from processing that data. A number of such computational platforms will be discussed below.

GitHub & Code Climate

GitHub is a web-based hosting service for version control using Git and is used by a large number of developers all around the world. (As of October 2018 it had 31 million users according to GitHub). GitHub measures key process-related metrics for software engineers and provides visualisations displaying the frequency of commits, and also a breakdown of the users' activity. These can be seen below:



The above image displays user 'phadej''s commits over the last 12 months with each square representing a day and lighter colour indicating fewer commits that day and darker colour indicating more. The image to the right displays the breakdown of this user's activity as a percentage. While this data might be useful for some developers, most will need more complex insights.



Code Climate provides engineering process insights and automated code review for GitHub and GitHub Enterprise to help users develop better software more quickly. Their services are employed by companies such as Salesforce, Intercom, Condé Nast and Heroku. Code Climate runs two main product lines, Velocity and Quality. Velocity provides trends and dashboards, team insights, pull request insights and risk alerts. Its purpose is to “remove bottlenecks and maximize the output of your engineering organization.” Its other service, Code Climate Quality provides automated code review comments on pull requests, test coverage, solutions to combat technical debt and the ability to correlate code quality information against areas of high churn and identify “hot-spots” to be looked at.
Below are a number of graphs provided by Code Climate which illustrate the insights that can be provided with this platform:

The above are examples of metrics that are extremely important when measuring the work of software engineers, displayed in a straightforward and easy to understand way and provide a useful platform for software engineers and companies to monitor and improve their productivity. A significant benefit of using this approach is that the collection of data is automated which saves the team from having to manually input or analyse any data themselves.

Jira Software

Jira Software is a software development tool created by Atlassian with the intent of helping agile teams to “plan, track, and release great software”. Jira Software provides scrum boards and kanban boards for the planning stage, which are fully

customizable and intend to decrease cycle time. Goals can be set and tasks laid out with each team member being included in the process. Roadmaps can be used to sketch out a timeline and compare and measure actual progress to what has been planned.

Jira Software includes a number of out of the box analytics to assist teams in measuring their progress against their goals. A variety of charts and figures on important metrics provide real-time, actionable information for each sprint.

One of the particularly beneficial aspects of Jira Software is how customizable it is. Custom filters can be created with Jira Query Language for, the platform can be integrated with developer tools and the platform supports over 3,000 applications which can be fully integrated to provide a variety of insights for teams.

Custom workflows can be created to “increase transparency, accountability and productivity”. Every team-member gets a custom digital board showing the status of their work but also that of the rest of the team. Each team-member is able to access the most up to date data at any point and view one another’s progress and the status of the project as a whole. Major milestones and deliverables can be planned around the product roadmap, feedback can be given to team-members and sprints can be planned effectively using the product roadmap.

Algorithmic Approaches

With technology becoming increasingly advanced, algorithms and machine learning are being used in ways they previously couldn’t. As explained by the SAS Institute, “the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development.”
The Institute for Scientific Interchange also explained that classic data mining techniques are often ineffective with today’s data and thus there is a focus on analysing large quantities of data in order to gain new and better insights and actionable knowledge which can be applied to solve problems.

A number of these algorithmic approaches and machine learning applications with respect to their use in measuring software engineering will be discussed.

Improvement in the accuracy of defect prediction and subsequent reduction in errors can lead to significant cost savings for developers, and statistical analysis and machine learning techniques could help to further develop the accuracy of such predictions. There are many software tools allowing developers to execute analysis on code using a variety of rules, however the unintended result of this is Metrics Galore as described by Bouwers, E. et al. (2012). This is whereby a software development team is trying to measure too many performance metrics simultaneously which can result in loss of sight of important goals of the project and focusing on the wrong metrics.

Regression, Classification and Machine Learning Models

Unsupervised machine learning may be able to help with the metrics issue according to Hinton, G., Sejnowski, T. (1999), however regression and classification models could help with predictive analysis by providing a model to the software engineers themselves. In a study conducted by Ross MacDonald (2018) appearing in 'Advances in Artificial Intelligence' at the 31st Canadian Conference on Artificial Intelligence, MacDonald applied regression and classification prediction models to a large set of software metrics data.

Hundreds of static metrics were collected, but the data was reduced to leave only the most important metrics. Several regression methods were performed such as Linear Regression, Neural Network Regression, and Support Vector Machine (SVM) Regression with the raw defect count targeted as the outcome. For classification, Decision Trees, Random Forest, Neural Network, and SVM were used with binary classification having a 'defect' or 'no defect' outcome and multi-class classification having outcomes of "No defects, Low defect rate, Moderate defect rate, High defect rate, and Extreme defect rate."

The results of the regression analysis were not as desired, with the best RMSE (Root Mean Squared Error) being quite large showing predictions that were not particularly accurate. However, binary classification showed very promising results, with accuracy of over 70% using Random Forest Classification. This is a powerful machine learning algorithm containing multiple decision trees. The random forest takes an average of all the individual decision tree estimates when making a prediction leading to a more robust outcome. (Koehrsen, W. (2017)).

I believe that this study is a very good indicator of the promise of algorithms and machine learning for the future of measuring software engineering and predictive analysis on code. While the results are from a single study and can't be taken as

conclusive, I believe that in the coming years algorithmic and machine learning approaches will play an even greater role in measuring engineering.

Ethics

There are several ethical considerations when we discuss measuring software engineering. There are legal issues, for example GDPR must be adhered to when collecting and storing information and data on employees, but also moral considerations. Is it right for a company to scrutinize their employees to the extent that they could be tracking every letter they type? I will discuss the ethical considerations when measuring engineering from both a legal and a moral standpoint.

Legal Considerations

In my opinion, the legal factor which is the most influential on how companies collect and manage their data is undoubtedly GDPR, which was implemented across the European Union in May 2018. The consequences for breaching the General Data Protection Regulation are serious. Companies can be fined up to the higher of €20 million or 4% of global annual turnover. Therefore it's essential that companies measuring their employees' performance and collecting large amounts of data comply with this regulation. The company may keep track of employees' emails, phone calls, instant messaging activity and even keystrokes and internet search history. Some of this data could be sensitive, so it's up to the company to ensure that they have the security in place to be able to store this information safely. As we have seen with data breaches in the past, such as Equifax in 2017 where over 145 million of their customers' social security numbers were compromised, the effects of data breaches can be severe, and thus it's a serious ethical concern for companies regarding what data and how much of it to collect on their employees.

Moral Considerations

While it's essential that a company stays within the bounds of the law when collecting data and measuring its employees, moral considerations are just as important, especially considering there is no industry code of ethics for software engineering as a whole to guide companies. The moral question in my opinion is whether it's right or fair to measure the work of employees with such specificity on so many metrics that engineers are put under tremendous pressure not just to accomplish their tasks, but to meet each of the numerous metrics they know they're being measured on. Instead of focusing on the overall goal, this could lead to a box-ticking approach for some engineers who instead of trying to do use a creative approach, focus instead on simply not doing anything wrong so as to satisfy a large number of metrics. It could also frustrate employees who don't enjoy being monitored so closely and may find it invasive, thus affecting overall morale in the workplace.

Conclusion

The world of software engineering is constantly eveolving, and so it's natural to expect the measurement process to advance alongside it. I have discussed the measurement process under a number of headings including Measurable Data, Computational Platforms, Algorithmic Approaches and Ethics. From my research it seems clear to me that measuring software engineering can be incredibly useful in tracking and improving performance. I would expect the computational and algorithmic approaches used to become even more effective in the future, with fields such as machine learning potentially having significant influences on the measurement process. However it's important with such advances to keep ethical considerations in mind to ensure that while measuring software engineers, their data is carefully managed and protected, while also ensuring

Bibliography

Sommerville, I (2008) Software engineering history https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/index.html

DeMarco, T (1982) *Controlling Software Projects, Management Measurement & Estimation*

Pedhazur, Elazar J.; Schmelkin, Liora Pedhazur (1991). *Measurement, Design, and Analysis: An Integrated Approach*

The Hartford, *Defining, Measuring, and Improving Employee Performance, https://www.thehartford.com/business-playbook/in-depth/measuring-evaluating-employee-performance-data*

Ebert, C., Dumke, R., Bundschuh, M., Schmietendorf, A. (2005). *Best Practices in Software Measurement*

Techopedia. *Definition of the term "Technical Debt" (plus, some background information and an "explanation").*

Fowler, Martin. *Technical Debt Quadrant*. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Thompson, Ben. *Why Code Churn Matters.* https://blog.gitprime.com/why-code-churn-matters/

Nagappan, Nachiappan., Ball, Thomas. (2005). *Use of Relative Code Churn Measures to Predict System Defect Density.* https://www.microsoft.com/enus/research/wpcontent/uploads/2016/02/icse05churn.pdf

Code Climate, https://codeclimate.com/

Atlassian - Jira Software, https://www.atlassian.com/software/jira

SAS, https://www.sas.com/

Institute for Scientific Interchange, https://www.isi.it/en/research/algorithmic-data-analytics

MacDonald R. (2018) *Software Defect Prediction from Code Quality Measurements via Machine Learning*. In: Bagheri E., Cheung J. (eds) Advances in Artificial Intelligence. Canadian AI 2018.

Bouwers, E., Visser, J., van Deursen, A. (2012). *Getting what you measure*. Commun. ACM **55**(7), 54–59

Hinton, G., Sejnowski, T. (1999). *Unsupervised Learning: Foundations of Neural Computation.* A Bradford Book, McGraw Hill Book Company

Koehrsen, William (2017). *Random Forest Simple Explanation.* https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d