Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

I have read and I understand the plagiarism provisions in the General Regulations of the University

Student Name	Tianze Zhang
Student ID Number	19334401
Course Title	Computer Science
Module Title	CSU33012: Software Engineering
Lecturer(s)	Stephen Barrett
Title	Measuring Software Engineering Report
Date Submitted	03/01/2022
Word Count	3000

Contents

1 Introduction 2
2 Software Metrics 2
2.1 Agile Process Metrics 3
2.1.1 Lead Time 3
2.1.2 Cycle Time 3
2.1.3 Team Velocity 4
2.2 Developer Productivity Metrics 4
2.2.1 Active Days 4
2.2.2 Code Churn 5
2.3 Test Metrics 5
2.3.1 Code Coverage 5
2.3.2 Defect Density 6
3 Measuring Tools and Platforms 6
3.1 GitHub 6
3.2 Jira 6
3.3 Technologies Supports 7
4 Algorithmic methods 7
4.1 Simple Counting 7
4.2 Expert Systems 8
4.3 Machine Learning 8
5 Ethical Considerations 9
5.1 Employee privacy 9
5.2 Human/ psychological factors 10
6 Conclusion 10
7 Bibliography 11

1 Introduction

In the current information age, careers related to software engineering have become very much in demand, which led to an increasing number of people becoming software engineers. Thus, it would be interesting for both the individual and companies to know the key performance indicators or attributes which a successful software engineer possesses. Companies could use those metrics for hiring, tracking the developers progress and effectiveness, and ensuring the delivery of the highest quality of code.

In this report, firstly, I discuss how to measure software engineering and what activities can we measure, by using different software metrics. Secondly, different tools and platforms that can be used to do the measuring are introduced as well as the underlying technologies that supports them. Thirdly, different algorithmic methods that performs analysis on the gathered data will be compared. Finally, possible ethical concerns for measuring software engineering efficacy are raised.

2 Software Metrics

In this section, I want to discuss how to measure engineering activities based on software metrics. Software metrics are a standard measure of range of activities concerned in software engineering (Fenton, 1999). In simple words, software metrics are data taken through the development process that measures the performance of software engineers. Software metrics helps measures a wide range of activities in software engineering including collaborations, process, code quality, productivity, and testing (Sealights, n.d.). Software metrics are beneficial to both the software engineers and the businesses. For example, Project manager can measure the cost and effectiveness of each process to obtain better insight into the project and make improvements. The developers can also improve their productivity by measuring their frequency of commits, quality of code and code coverage (infopulse, 2018). However, software metrics can be subjective, useless and have limitations. For example, a recent case study of more than 600 software professionals revealed that only 27 percent viewed metrics as “very” or “extremely” important to their software project decision-making process (Johnson, 2005). In the next few sections, different software metrics that are used as means to measure software engineering activities are introduced and discussed.

2.1 Agile Process Metrics

Agile is a development approach used in software engineering through discovering requirements and improving solutions through collaborating effort of self-organising and cross-functional teams with their customers (Wikipedia, Agile software development, 2021). In my opinion, measuring the process of developing is more useful than only measuring the quality of the end-product as by measuring the developing process, a better insight into productivity in different stages of software development lifecycle will be provided. Therefore, provides better indication into the performance of software engineers and software engineering teams. There are many metrics used in measuring agile process including lead time, cycle time and team velocity.

2.1.1 Lead Time

Lead time is the period between task creation and the actual delivery (Bluemel, 2020). Lead time is an important metric as it provides the exact time calculation for every process. Therefore, by looking at the lead time data, developers can see which stages leads to high lead time and try reducing on it in the future. It is sometimes more useful to view lead time and cycle time together, which might lead to more valuable conclusion.

2.1.2 Cycle Time

Unlike lead time, cycle time is the period between the moment where the work is started and the actual delivery (image 1). Cycle time data is helpful when combining it with lead time. For example, one can see whether if it is a high cycle time that causes a high lead time, which mean developers are not effective enough on developing. Alternatively, if a high lead time with low cycle time is observed, it might mean that developers are not effective enough analysing the task etc (Linnanvuo, 2015).

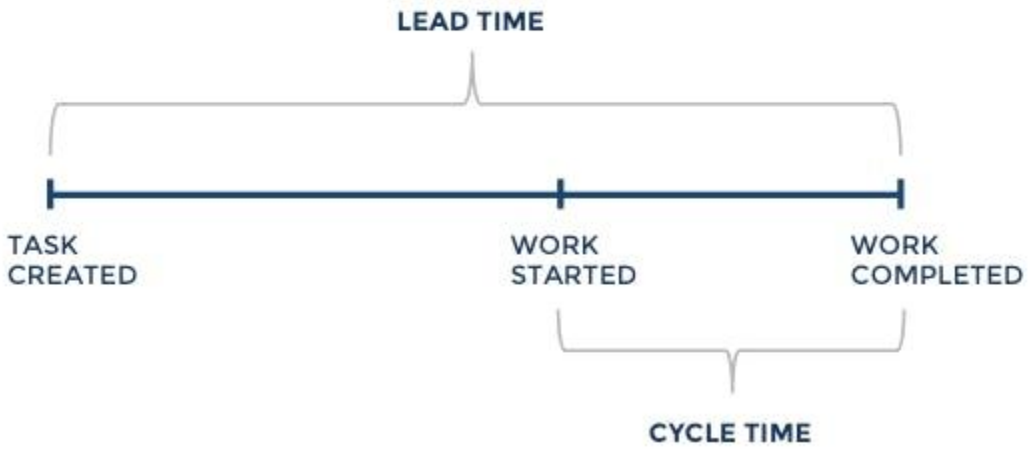

Image 1: The above image shows the relationship between lead time and cycle time:
Lead time includes cycle time which is the time between task created and work completed,
while cycle time is the time between work started and work completed.

2.1.3 Team Velocity

Velocity is a measurement of unit of works completed in each time frame, including engineer hours, user stories, or story points. It helps to understand the effectiveness of the team in providing to the customers in unit time. It also helps predict how many sprint/times are required to complete the project (Sealights, n.d.).

2.2 Developer Productivity Metrics

Developer Productivity metrics helps to measure how efficient a developer is to get a project done. There are metrics such as total commits and lines of code, which are considered less reliable and not used in modern development environment. For example, there might be a developer that writes and commits huge amount lines of code frequently, that are either not working or full of bugs, and thus shouldn’t be accounted as an efficient developer. Therefore, two metrics that are more reliable are introduced in the next few sections.

2.2.1 Active Days

Unlike measuring the hours spend developing, in my opinion, it is more reliable to measuring the total active days. For example, some developers can complete the work faster and better than others, but after the work is completed, he doesn’t collaborate with the team anymore, such as not attending the scrum meetings. Additionally, it may lead to lack of ownership in the developers work, such as fixing a bug on a code previously committed by them. Measuring the active days therefore can give more reliable measurement of developers.

2.2.2 Code Churn

Code churn is a metric that measures how often a piece of code gets modified. If a piece of code is getting changed too frequently, it means the quality of this piece of code is poor: either it is full of bugs or the developer had incomplete understanding of the requirements from client, which indicates the developer who delivered this piece of code is an inefficient developer. Code churn is helpful for the developers to know if they are rushing the development process, developers might improve by spending more time on requirements analysis, testing and debugging. Oftentimes, a low rate of code churn is expected (below 20%) (Schults, 2021).

2.3 Test Metrics

Software testing metrics are measures used to the progress, quality, productivity, and health of the software testing process (Hamilton, 2021). Most software engineers use TDD (test-driven-development) which means writing the tests before implementing the actual code. Therefore, in my opinion, providing more efficient and effective software testing process are essential and directly links to the quality of software delivered. Below are two test metrics that are explored and discussed.

2.3.1 Code Coverage

Code coverage is a metric that measures the amount of code that the test suite that you have written covers. Software engineers can use this metric to tell the quality of their test suite. By improving the code coverage, the probability of bugs in the code decreases. Test suite that has a code coverage of 70% or above often indicates a healthy test suite, which means the test suite written by the software engineer tested 80% of lines of code produced (PITTET, n.d.).

2.3.2 Defect Density

Defect Density metric measures the number of defects detected in software or component during a development period. It is a helpful measurement to decide whether the software is ready to release or not. If the software does not satisfy the requirements to release, the defect density metrics provides an estimate of the amount of fixing and testing needs to be done. By analysing common defects density, developers can also predict remaining defects. Overall, Defect density metrics is an efficient testing technique, which saves software engineers a huge amount of time (Nanda, 2021).

3 Measuring Tools and Platforms

There are various tools and platform out there that are able to calculate and gather these datasets used to measure software engineering activities. They are supported by different technologies to make collecting large amount of data possible and also enables complex and computationally expensive calculations to be performed.

3.1 GitHub

GitHub is a code hosting platform for version control and collaboration. In fact, most companies and software engineers store their code base on GitHub. As of November 2021, GitHub reports having over 73 million developers and more than 200 million repositories (Wikipedia, GitHub, 2021). GitHub itself provides different software engineering measurements to both users and repositories. It provides measurements on productivity and bug tracking. For example, GitHub stores every users' number, history and frequency of commits. GitHub also provides the lines of code and contents that have been changed between commits. GitHub has light weighted bug tracking system called GitHub issues that are available in all repositories used to report bugs and request features. All of those data are accessible through GitHub's rest API (Application Programming Interface) (Docs, n.d.).

3.2 Jira

Originally, Jira was designed as a bug and issue tracker. Nowadays, it became a powerful tool used in many companies, as a work management tool for all kind of use cases in agile software development. For example, tasks can be easily set up, tracked and report in Jira clearly and accessible to all software engineers. Jira also provides visualization of data that are useful for software engineers, such as virtual scrum, Kanban boards, dashboard reporting of issue, sprint, and release progress. Jira also has an open rest API that allows for integration with other tools such as GitHub and Bitbucket. Different branches of GitHub's repository for the project can be set up based on different Jira tasks. These features provide good measurements on agile development process. Therefore, it helps the software engineers decrease their lead time and increase their team velocity (Software).

3.3 Technologies Supports

As these platform and tools are expanding massively worldwide, more and more computational power is required to store those massive amounts of data and be able to pass through the API. However, cloud hosting platforms such as AWS (Amazon web services), GCP (Google cloud platform) and Microsoft Azure that provides services such as virtual machines and Kubernetes for these platforms to host on (Wikipedia, Cloud computing, 2021) which supports the gathering of large volumes of data. In fact, both GitHub and Jira rely on AWS, and private repositories in GitHub are hosted on AWS (AWS, n.d.). These cloud platforms also provide different methods to analysis the data, including machine learning which will be discussed in the next section.

4 Algorithmic methods

This section focusses about some data analysis techniques that could be done over software engineering data, in order to profile the performance of software engineers. Methods such as simple counting, Expert systems and machine learning will be discussed.

4.1 Simple Counting

This is the simplest method to perform on software engineering data. This method involves counting the total amount of data iteratively and raise up with some conclusion. Although this method is straightforward, however, it still can be used widely in simple datasets that are countable. For example, simple counting can be used in developer performance metrics: total commits, frequency of code changed, and active days can all be gathered and analysed by simple counting. Simple counting also can be used in testing metrics: the lines of code coverage and number of defects are also gathered using simple counting. The main advantages of simple counting are that it is easy to understand and inexpensive. On the other hand, its disadvantage is also clear that counting is hard to perform in an extensively large dataset. Conclusion that are drawn only be counting the dataset can also be unreliable and hard to interpret than other methods such as expert systems and machine learning.

4.2 Expert Systems

An expert Systems is a computer system simulating the decision-making ability of human experts. Expert systems were designed to solve complex problems by reasoning through its knowledge base, which are mostly if-else rules, along with its inference engine to deduce new facts from a given data set (Wikipedia, Expert system, 2021).

By given values of certain metrics, the system could then provide interpretation of any abnormal patterns by reasoning through it's knowledge base. There are several expert systems built in order to measure software engineering performance including development of four separate, prototype expert systems from IEEE (IEEE, 1989).

A main advantage of expert system is it could give more reliable and useful interprets from a complex dataset which cannot be achieved by counting. However, the disadvantage is that as its own knowledge base increases significantly, it will raise many computational problems. Therefore, experts' systems still do not solve the issue with massive amount of dataset.

4.3 Machine Learning

Machine learning is the study of computer algorithms which can improve automatically through experience and the use of data. Machine learning not only can calculate and analyse data, but also predict a software engineering data. For example, machine learning can be useful to predict other defects from a given defect metrics, using supervised machine learning methods such as linear regression and logistic regression.

Machine learnings major advantage over other techniques is that it performs nicely, no matter the size of the dataset. For extensive large data set, dimensions reduction techniques such as

PCA (principal component analysis) can be performed on the dataset, which only extracts out the important components.

A major disadvantage of machine learning is that it might suffer strongly from different data bias. However, in my opinion, machine learning is still the most effective way to analysis software engineering data, as it provides most analysis algorithms and also performs well with large datasets.

5 Ethical Considerations

Although measuring software engineering activities can be beneficial for both the companies and the developers, there are ethical concerns that must be considered. Some of the research on measuring software engineering activities will be listed, and I will discuss my opinion on those ethical considerations.

5.1 Employee privacy

The first ethical concern is employee’s privacy. In the EU, GDPR regulations set clear guidelines on the transparency of data collection and storing methods by companies, and specifically, the extent of how and when workplace monitoring is allowed. Under GDPR, companies are allowed to gather and process essential employee data, however, they are under no circumstances justified in using exhaustive or automated monitoring methods to excessively monitor employees, such as looking through their browser history and workplace communications (Irwin, 2021). I agree with the fundamental ideas and concepts of the EU GDPR regulations. I believe that individuals should have control over their own data (is it fair that companies profit from data that are generated by employees/someone else), and that employees should possess a certain amount of freedom in performing their day to day jobs. If intrusive metrics are collected, it creates an unhealthy workplace culture of monitoring and fear which can be detrimental to an employee’s wellbeing and sense of security. It may lead to decreased employee creativity and initiatives which could be beneficial to the company. Intrusive metrics such as tapping into an employee’s browser history and doing analysis on it may have the intentions to increase productivity but end up having a negative effect as employees might rebel against or work around the rules, thus decreasing their productivity.

5.2 Human/ psychological factors

Numbers are a precise way to quantify patterns and trends, however it presents an abstracted view which can overlook other important factors. For example, only looking at engineering activity metrics without considering the effects of human factors is poor decision making and can be counter-effective. In fact, in the past, software measurements were implemented in many companies, however, due to the overjudging and overusing of those measurements, studies have shown that they are counterproductive and many people resist against measurements, and claimed that measurements aren’t helpful at all (MCDONALD, 2018).

In my opinion, the metrics should be assigned less weight in decision making and only serve as a motivation to the developers, as a general guide to help developers improve. Companies shouldn’t use them to control employees. In addition, they shouldn’t judge an employee’s skill and expertise based on a singular metric, which contains limited information and doesn’t show the holistic view of the employee. For example, over relying on those measurements might result in negative employee behaviour such as spending all their efforts to fake a good result on the measurement, rather than focusing on the client requirements, quality of the code and learning and development. Therefore, it produces a negative effect.

6 Conclusion

In conclusion, measuring software engineering is complex, and there are many aspects to consider when measuring software engineering activities. The complexity and considerations were discussed carefully in the report: Firstly, various software metrics measuring different activities was discussed along their limitations. Secondly, different tools and platforms to consider using which helps to perform this measure was discussed. Thirdly, some algorithms methods that performs analysis of software engineering data were compared. Lastly, possible ethical concerns was raised a discussed. I believe measuring software engineering are beneficial and developers can make improvements from them if used correctly, and it is definitely worth trying out.

7 Bibliography

AWS. (n.d.). *AWS*. Retrieved from AWS: https://aws.amazon.com/codecommit/

Bluemel, A. D. (2020, May 28). *Lead time: What is it and why should Agile teams (or any teams) care?* Retrieved from Shortcut: https://shortcut.com/blog/lead-time-what-is-it-and-why-should-you-care

Docs, G. (n.d.). *REST API*. Retrieved from Github Docs: https://docs.github.com/en/rest

Fenton, N. E. (1999). *"Software metrics: successes, failures and new directions."*. Journal of Systems and Software 47.2 .

Hamilton, T. (2021, December 18). *Software Testing Metrics: What is, Types & Example*. Retrieved from Guru99: https://www.guru99.com/software-testing-metrics-complete-tutorial.html

IEEE. (1989). *An evaluation of expert systems for software engineering management.* IEEE.

infopulse. (2018, October 19). *TOP 10 SOFTWARE DEVELOPMENT METRICS TO MEASURE PRODUCTIVITY*. Retrieved from infopulse: https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/

Information, C. (2020, September 29). *Data protection in the workplace*. Retrieved from Citizens Information: https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/data_protection_in_the_workplace.html

Irwin, L. (2021, May 20). *Can your organisation monitor employees' personal communications?* Retrieved from governance: https://www.itgovernance.eu/blog/en/the-gdpr-can-your-organisation-monitor-employees-personal-communications

Johnson, P. M. (2005). Improving software development management through software project telemetry. In P. M. Johnson, *IEEE software 22.4* (pp. 76-85).

Linnanvuo, S. (2015, October 28). *Agile and Lean Metrics: Cycle Time*. Retrieved from Screenful: https://screenful.com/blog/software-development-metrics-cycle-time

MCDONALD, M. (2018, January 3). *Do Your Measures Make Employees Mad? Or Motivate Them?* Retrieved from Workplace: https://www.gallup.com/workplace/231659/performance-measures-motivate-madden-employees.aspx

Nanda, V. (2021, July 13). *What is Defect Density? Formula to calculate with Example*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/what-is-defect-density-formula-to-calculate-with-example

PITTET, S. (n.d.). *An introduction to code coverage*. Retrieved from ATLASSIAN CI/CD: https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

Schults, C. (2021, February 10). *Code Churn: Everything You Need To Know*. Retrieved from LINEARB: https://linearb.io/blog/what-is-code-churn/

Sealights. (n.d.). *Top 5 Software Metrics to Manage Development Projects Effectively*. Retrieved from Sealights: https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/

Software, J. (n.d.). *What is jira?* Retrieved from CPRIME: https://www.cprime.com/wp-content/uploads/2018/03/Jira-FAQ-cPrime.pdf

Wikipedia. (2021, December 28). *Agile software development*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Agile_software_development

Wikipedia. (2021, December 31). *Cloud computing*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Cloud_computing

Wikipedia. (2021, November 6). *Expert system*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Expert_system

Wikipedia. (2021, December 31). *GitHub*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/GitHub