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

A Report on Measuring Software Engineering

CSU33012

Ciara Lynch
19336403

3rd January 2021

Contents

Introduction

In today's society it is very apparent as to why software engineering is so vital. Across the world software is necessary for almost every industry, for multiple businesses all for a variety of reasons and uses. If something was to break or malfunction this would have huge implications for the specific sector. An effective, quick, and efficient solution to the error must occur as soon as possible. Therefore, software engineers within their respective fields must think through logical solutions on a frequent basis. This is why the process of measuring software engineer's activity and productivity has become so dominant in the tech industry.

Software engineering is defined as the process of analysing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development. In comparison to software development or regular programming the above description of software engineering is mainly associated with its application to large and complex software systems that are integral to different industries.

In this report I plan to go through a variety of sections that will cover different methods and ways to measure software engineering. I will first detail the history of software engineering and go through its background so to allow further understanding of the different methods of measurement. After discussing the metrics of measuring software engineering and the different platforms software developers use, I will elaborate on the ways all this data can be used through a variety of computational algorithms. The last section this report will cover is the ethical and moral discussions surrounding the use of this data and the legal perspective that must be considered with the data's application.

History of Software Engineering

In the 1960s was the beginning of software. From its initial footing, software quickly evolved into a world of how best to create software and maximise its quality. The quality of software is in reference to its main principles, its stability, maintainability, usability, speed, testability, readability, the size and cost it takes to run the application and the result for the customer's use.
Software Engineering itself has a broad umbrella to what it truly encompasses, the main challenges that must be tackled are to maintain the best-practices when coding that software engineering follows. The process of software development and how to ensure the most efficient delivery of the application also falls under the software engineering bubble.

The period of the so-called software crises occurred in the 1960s and 1980s, this spurred the push to create better practices and efficiency within in software engineering. It is common knowledge that this period of time was a real driving force behind software engineering.

The NATO Science Committee even sponsored two conferences on software engineering specifically in 1968 and 1969 which resulted in the field getting its initial boost. Previous to this there was many projects that ran over schedule and budget, these productivity issues also resulted in quality issues too.

The main example of this issue that occurred was the project OS/360 operating system. This project both ran over cost and budget. It continued to produce one of the most complex software systems of its time, but the few mistakes that was made in its management resulted in costing the development multiple millions of dollars. Looking back at this, it is clear that it was the lack of developing a coherent software architecture that resulted in these issues down the road. Unfortunately, there are software developments from the past such as the Therac-25 project that showcase how these defects can result in deaths.

One of the main figures in the field Michael A. Jackson has written many papers and books on software engineering. He has gone into extensive detail about the problems the field faces in regard to its lack of specialization. For years the debate of how programmers lacked discipline started to heighten. Many individuals when looking for solutions to different issues believed their solution was optimal and considered it the 'silver bullet'. There was many that believed that if formal engineering methodologies were applied to the process of software development it would create a sense of predictability in the industry similar to other branches of engineering.

A controversial publication was the No Silver Bullet article made in 1986 by Fred Brooks. He stated that within the industry no individual would be able to produce a technology, practice or methodology that would ever make a 10-fold improvement in productivity. However, as the industry continued to prosper every new technology has made vast improvements to productivity and quality overall. All these additions have gone on to build from each other to vastly improve each new application in an efficient manner.

Measuring Engineering Activity

In this section I will outline some of the metrics that are used in an attempt to measure software engineering. Since its beginnings as I previously mentioned, there have been many attempts to find the best way to measure and track software development, below I plan to elaborate on the metrics I find interesting.

Frequency of code commits

Frequency of code commits refers to the practice of a software developer uploading "committing" their project to a code hosting platform often where version control is managed by Git i.e Github. It is a very simple metric and depending on the platform easy to visualise the productivity of the developer as it can be seen how often the developer is working. However, this metric is not much more than a representing how often a developer is saving their work or uploading their project. Frequency of commits therefore in my opinion cannot be considered an accurate measure of efficiency or quality of an engineer due to this reason of a possible unfair interpretation.

Lines of code

Lines of code is a metric where the volume of code and how much is written is taken into account. In a similar way to the frequency of commits metric, it can be very easily misinterpreted and therefore not a popular option when trying to measure a software engineer's quality and efficiency in their development. If this metric is applied in a comparison of two software engineers what can occur is an unfair interpretation of their logic and efficiency.

When interpreting two engineer's code on the same project if one has more lines of code for a specific algorithm let's say than the other, they would be considered under this metric the stronger developer. However, it doesn't take into account if the other engineer was able to apply the logic of the same algorithm in a more concise manner which could also be easier to read and work with. Under the lines of code metric, quality of the code being developed is completely disregarded as well as efficiency, as it is a well-known fact in software development that quantity does not result in quality or efficiency.

Testing code coverage

Another more popular metric used in measuring software engineering is testing the code coverage of a project or application. Testing code refers to the process of creating test procedures to exam the code of a project for any bugs, errors, or uncaught logic faults. This method of ensuring the quality of code is to a high standard allows a developer to find any bugs easily but also provides the opportunity to double check that the code is set to handle the project specifications set for it.

This metric of measurement I believe is one of the best to measure the quality and efficiency of a software engineers' development. Through the use of stress testing, it can be found how efficient the code is when given different and larger than necessary parameters, also through testing it is easy to see the quality of the code and if it can handle the parameters expected of the project and more.

Cycle Time

A metric that is quite popular within different companies to measure a broader team perspective of software development is the use of cycle time. Cycle time is a metric that measures the amount of time spent on a project from its beginning to the deployment. This metric is based on the logic behind the method of agile development, which I will discuss in greater detail further on in this report. The stages of cycle time are, coding, pickup, review and deploy.

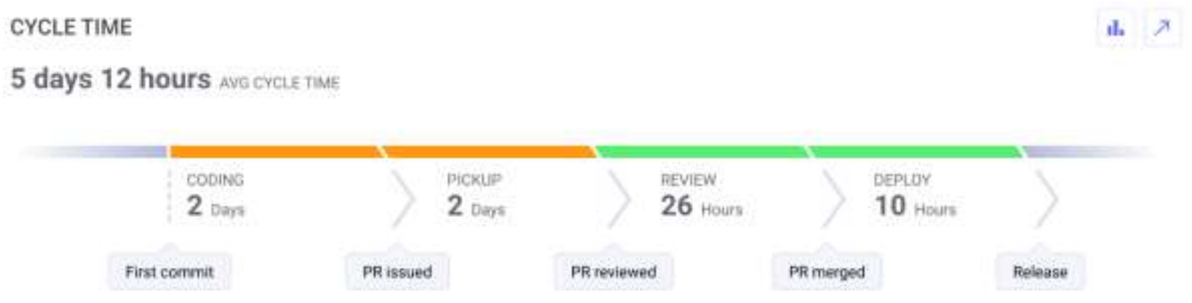


Figure 1 – visualisation of the cycle time metric

Code Complexity

Code complexity is a metric that more accurately can measure the quality of a software engineer's code and follows on from the lines of code and testing code coverage metrics. When a segment of code within a project is overly complex or complicated to read for a developer this can lead to a multitude of problems. It increases the likelihood of errors, bugs, inaccurate testing and many more faults, this all results in further unplanned costly solutions for the company. In my opinion code complexity is one of the better measurement metrics discussed in this report as it allows for a clear visualisation of the quality of a software engineers' code. When code complexity is taken into account within a project it will eliminate unnecessary and complicated code allowing for better tests, reduced risks in the project, lower costs when the risk of potential defects is reduced and greater predictability of deployment.

As a final note in this section, I would like to make my opinion clear on the use of these different metrics. I strongly believe that when a metric is applied to measure the ability of a software engineer, that they shouldn't be assessed by the volume of lines of code they've written or adjusted but by the quality of the code they produced and maintained.

Software Development Methods

Within software engineering many methods have been created and debated on to try and improve efficiency and performance when developing different projects and applications. Below I will discuss some of the more prominent methods and models used in industry by many teams.

Agile Methods

The agile model is a combination of iterative and incremental process models that have a focus on process adaptability and overall customer satisfaction through the rapid delivery of a working software product. How agile method is applied to the production of an application or project is that the project is broken into small incremental builds taken on by

different cross functional teams of software engineers working simultaneously on areas such as design, coding, unit testing and deployment.
The value in the agile method of development is that it allows the software engineers to deliver the project to a higher standard more efficiently and with a greater quality through an iterative approach.

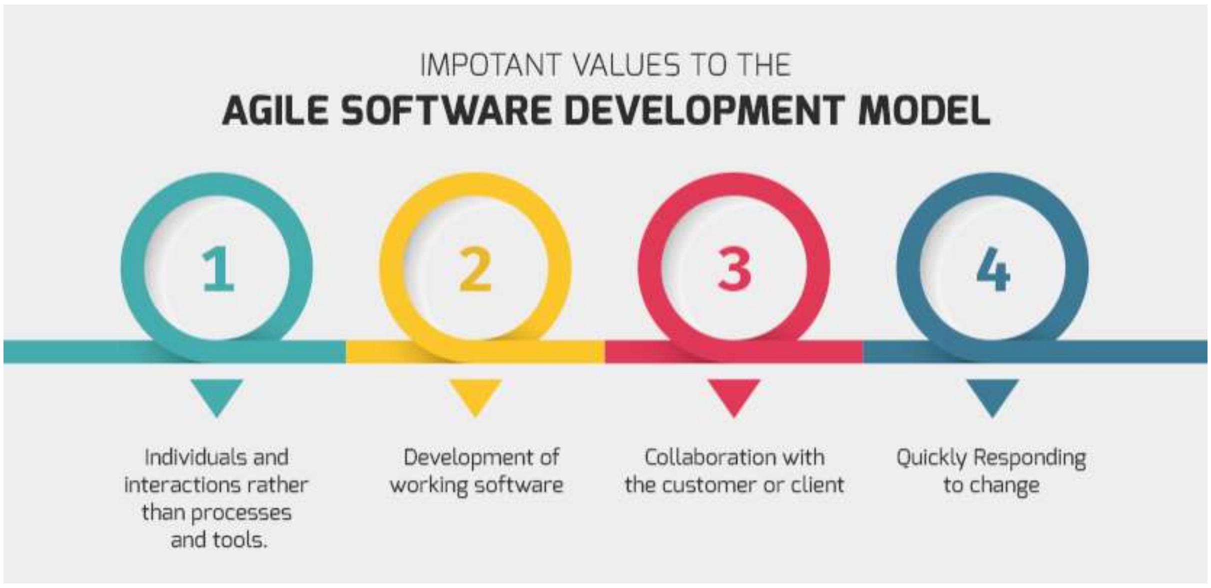
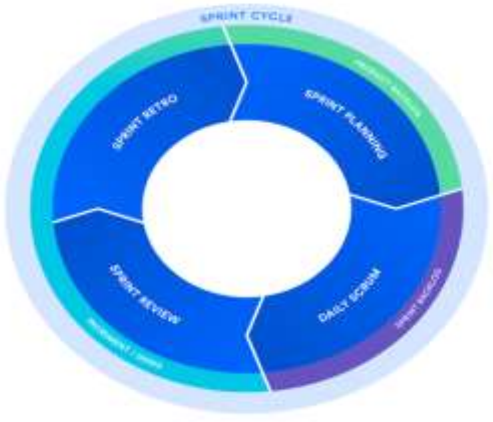

IMPOTANT VALUES TO THE
AGILE SOFTWARE DEVELOPMENT MODEL
1
2
3
4
Individuals and interactions rather than processes and tools.
Development of working software
Collaboration with the customer or client
Quickly Responding to change

Figure 2 – An image stating the important values in the agile model

Scrum
A subset of the agile development model is Scrum. It is the most-widely used method of the agile model within industry as it an efficient, process framework for software development. How it differs from other agile processes is that scrum has specific concepts and practices which are classified as roles, artifacts, and time boxes.
The scrum method is considered a process framework which means that it follows a specific set of practices for it to be consistent. The Scrum process uses the development cycles referred to as Sprints. Each of these processes are kept as small as possible to ensure optimal productivity for each section, therefore Scrum is considered a lightweight method.

Figure 3 – Image showing stages of Sprint cycle



Spiral Methods

The spiral method of software development was first brought to light by Barry Boehm in his 1986 paper on the topic. It is a combination of the iterative and waterfall style development methods, which I will discuss further below. It aims to take the advantages of both of these models which are their top-down and bottom-up approach to production. There are four phases, identification, design, development and evaluation and risk analysis. When this model is applied to a software development project, these processes are passed through repeatedly in iterations called spials. This method is very advantageous in managing the risk of a project as a result however the software engineers must be well-versed in risk evaluation.

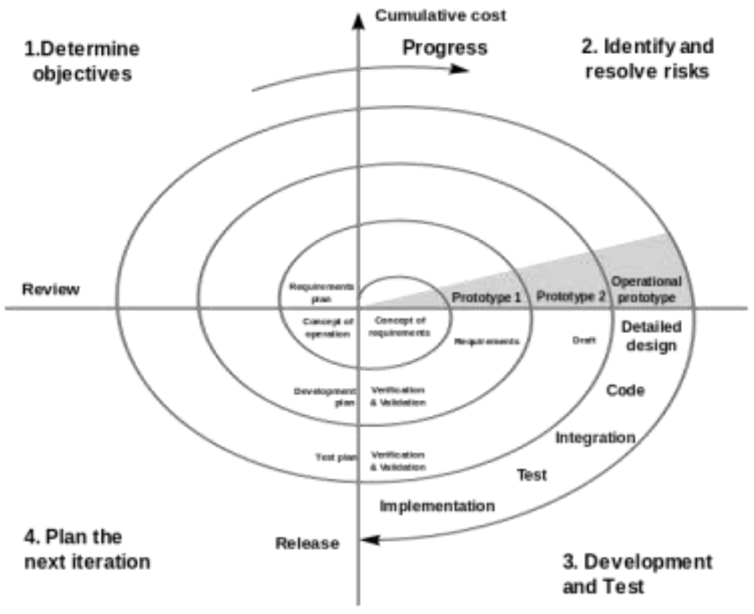


Figure 4 – Visualisation of the spiral model processes.

Waterfall Model

The waterfall model is quite different to previously mentioned software development methods, it breaks down the project into a series of linear sequential phases where the progression of the project depends on the completion and deliverables of the previous section. Due to this approach in software development, it is quite less iterative and flexible than the other methods and therefore considered to be quite outdated. Teams that chose the waterfall model are usually developing projects that do not anticipate unforeseen changes mid-development as this model does not support going back to previously completed phases.

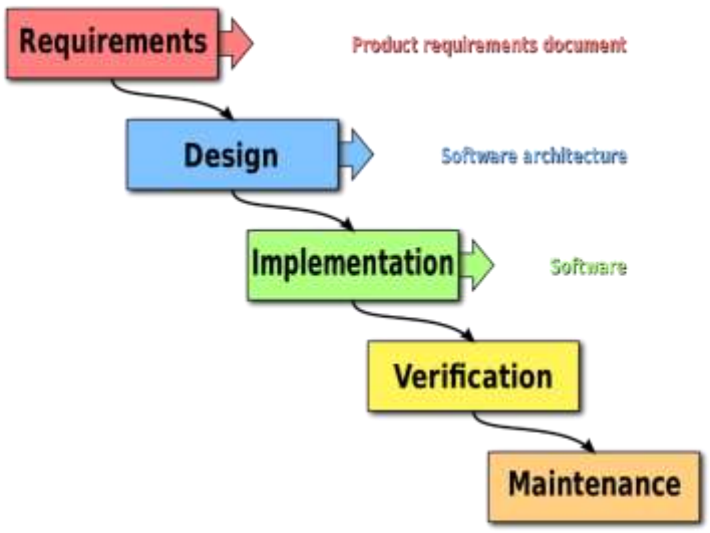


Figure 5 – Waterfall Model Phases

Software Development Platforms

In this section I will go through two of the most popular platforms that are used to collect and review the performance data sets of software engineers.

GitPrime/Pluralsight Flow

As I previously talked about many software engineers will subscribe to the use of git version control when they are amid development on various projects. GitPrime, now recognised as Pluralsight Flow is the overhead performance measurement platform that collects and analyses useful data gathered from these platforms i.e. GitHub and Bitbucket. GitPrime allows head engineers in companies or on project teams to analyse and review their team developers progress and productivity. GitPrime has various functionalities to allow companies to be able to analyse their software engineer's performance one of the more popular options being a percentile performance of contributions made to a project. On this platform, there is an emphasis put on metrics applied to lines of code such as the functionalities impact scores measuring the speed and difficulty of line changes.

HayStack

Another platform used within the industry to help analyse and measure software engineers' productivity and performance is HayStack. In comparison to GitPrime, HayStack offers a wider variety of metrics than just lines of code such as cycle time, which I previously touched on, deployment of the project/code frequency. A notable feature that haystack also offers is the potential for burnout metric which will show the project managers the position their software engineers are in and ensuring that they do not get burned out which could result in project errors down the line.

In my opinion HayStack provides better functionality and utilisation of the data collected from software engineers as it ensures a fair and broad interpretation of the productivity of the developers through use of greater and more accurate metrics

Computational Methods to Profile Software Engineers

When the different data is collected many companies will then go and delve into different methods of analysing this data on their software engineers without the use of the previously mentioned platforms. There are many approaches that can be taken to measure and assess the software engineering process; However, it has become clear in the last number of years that applying the data sets to machine learning algorithms is the most effective and accurate.

The algorithmic approach of Machine Learning

Machine learning (ML) is defined as the application of artificial intelligence that provides computational systems the ability to automatically learn and improve from experience without explicit instructions or coding, relying on patterns and inference instead through the study of algorithms, statistical models and provided data sets.
Algorithms used in machine learning build upon mathematical models based on sample data provided to it known as training data for the system. It learns from this training data and the patterns to make predictions and decisions without being explicitly programmed to.

In the case of this report, the data sets would be a variety of data points generated by each software engineer such as frequency of commits, lines of code, code complexity, code coverage and cycle time on a project. All of this data would be supplied to the machine learning algorithms, which I will discuss in detail below and then it would generate a profile output based on the patterns of the efficiency and quality for example on the software engineer. Below I elaborate on the types of ML algorithms and briefly explain the statistical logic behind them.

Supervised Learning

The goal in supervised learning is to map an input to and output based on given examples of input-output pairs. The task of learning this function is inferred from a function supplied by the training data that consists of the input-output examples. Supervised learning algorithms are mainly split between the Classification and Regression approaches.

The classification method is aimed at trying to identify the category of a new observation, new supplied data point, among a set of categories set out by the labelled training set. In software engineering measurement these would be the previously defined and then supplied measurement metrics by the company.

The regression method instead outputs a numerical value within a range, trying to predict the continuous outcome based on the value of a multitude of supplied predictor variables in the training set i.e. forecasting how efficient a software engineer is with an outputted score based on the collected data points of their frequency of commits.

Unsupervised Learning

In unsupervised learning only input data is provided in the training set, there is no labelled outputs for the algorithm to compare and aim for. This allows for the algorithm to find unknown patterns in the data set without pre-existing labels. Therefore, unsupervised learning is known as knowledge discovery which is very useful when conducting exploratory data analysis.

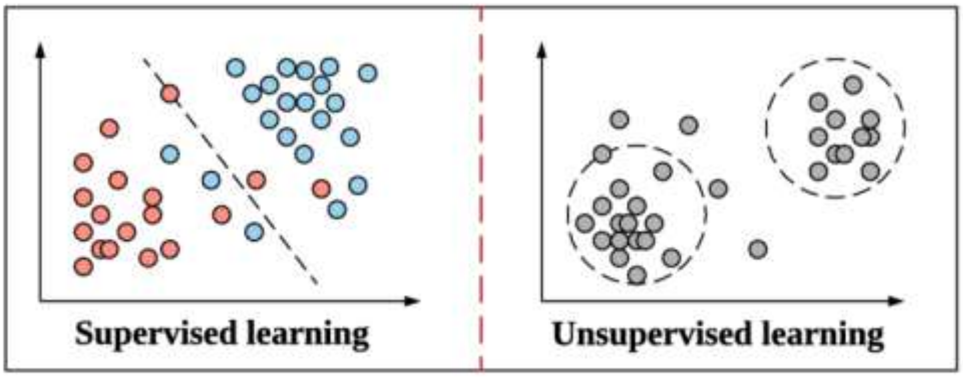


Figure 6 – Visualisation of the classification of SL vs. the clustering methods of UL

Reinforcement Learning

Reinforcement learning is very similar to reward-motivated behaviour commonly studied in psychology. Here it is the training of machine learning models to make a sequence of decisions in an uncertain potentially complex environment so to receive some reward.

Ethical and Moral Concerns

Within the world today, the discussions surrounding data collection and analyses is a very controversial one. Ethics at its core is the clarification and recommendation of the concepts of right and wrong conduct.

In this report I have gone into detail about the ways a software engineer's performance is measured through the various metrics, platforms, methods, and computational analyses of the data collected. However, the question must be asked is this tracking and analyses of the efficiency and performance of software engineering teams and possible actions taken after interpretation of the results ethical? In my opinion I believe the answer is not a clearly defined yes or no.

The idea of collecting data and analysing it to try and find weak spots within project development and improving upon these faults, I believe, at its core is a good idea and should be considered completely. However, in reality this analyses process is not always appropriately done as the real-world industry is much more complex than text-book methodology. A big issue that comes to light with the idea of tracking and trying to measure software engineer's performance is the invasion of privacy connected to gathering and tracking the individual's data. Another common issue not discussed frequently is the unnecessary pressure this tracking puts on developers to perform at a high standard continuously. At times this push can be successful for overall company performance but oftentimes just results in software engineer burnout because of unhealthy levels of stress effecting both mental and physical health.

I do believe that the tracking of the metrics previously discussed in this report would aid the performance of software engineers and project development however I strongly am of the opinion that companies must do so cautiously and not at the expense of their staff.

Conclusion

To conclude, in this report I have discussed the various ways in how to measure and assess the software engineering process, through various measurement metrics, methods to improve process efficiency, platforms to help analyse data and quickly progressing algorithmic methods. I also touched on how all of this accumulates into the ethical considerations that must be taken by companies when working with and conducting performance reviews of their software engineers.

References

Software Engineering Definition - https://www.techopedia.com/definition/13296/software-engineering
Content for the history of software engineering - https://en.wikipedia.org/wiki/History_of_software_engineering
Cycle Time metric (including referenced figure 1) - https://linearb.io/cycle-time/
Code complexity metric - https://blog.codacy.com/an-in-depth-explanation-of-code-complexity/
Agile method definition - https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm
Software Development Methods (including referenced figure 2, 4 and 5) - https://ca.insight.com/en_CA/content-and-resources/2016/07152016-types-of-software-development-models.html
Scrum Research - https://www.atlassian.com/agile/scrum
Scrum Sprint Photo (figure 3) - https://wac-cdn.atlassian.com/dam/jcr:709d95de-f4d4-4faf-aa9f-4bead10e888b/sprint_cycle-c.png?cdnVersion=140
Haystack and Pluralsight Flow comparison – https://www.usehaystack.io/haystack-alternatives/haystack-vs-pluralsight-flow-gitprime-a-pluralsight-flow-gitprime-alternative?utm_campaign=Competitor%20Ads%202.0&utm_source=google&utm_medium=cpc&utm_content=Pluralsight%20Flow&utm_term=gitprime&gclid=Cj0KCQiAt8WOBhDbARIsANQLp94b1x3DDJOkLA3RJiwPhRkuAjM12N_ZPuLPZXV7rZzw55E2lgSs_R4aAvjHEALw_wcB
Machine Learning and types of algorithms (SL, UL, RL) - https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0
Classification Definition - https://www.sciencedirect.com/topics/engineering/classification-method
Regression explanation in ML - https://builtin.com/data-science/regression-machine-learning
Supervised vs Unsupervised Figure 6 - https://www.researchgate.net/figure/Examples-of-Supervised-Learning-Linear-Regression-and-Unsupervised-Learning_fig3_336550812
Reinforcement Learning further explanation - https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/
Ethics and moral definition - https://www.sciencedirect.com/topics/psychology/moral-philosophy