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

CSU33012 Software Engineering
Measuring Engineering

How Software Engineering is Measured and Assessed

Daniel Whelan, 19335045

November 1, 2021

Contents

1 Introduction

This report will provide an analysis on the effect of these areas following and in turn how they impact the analysis of Software Engineering.

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

According to Laplante (2007) "Software Engineering is the systematic application of the engineering approach to the development of software". Following this, software production consists of the following four steps applied in a systematic way.

1. Deciding on the specifications of software.
2. Design and implementation of the software.
3. Testing and verification of the validation of the software.
4. Maintenance of the software.

Hence, when creating a system that is designed to measure the discipline of software engineering, it is important to account for these four crucial activities.

In turn, the system must account for the application of these activities as there are a multitude of different approaches that organisations apply when developing software for use. These include the waterfall, incremental and agile development methods. The Agile development method seems to be the most popular type of development strategy at the moment and is an iterative approach that keeps pace with the dynamic development requirements of the modern age and mainly splits into extreme programming and scrums (Ahmed et al, 2010). As a result Agile Development processes will be the focus of measurement methods throughout this report.

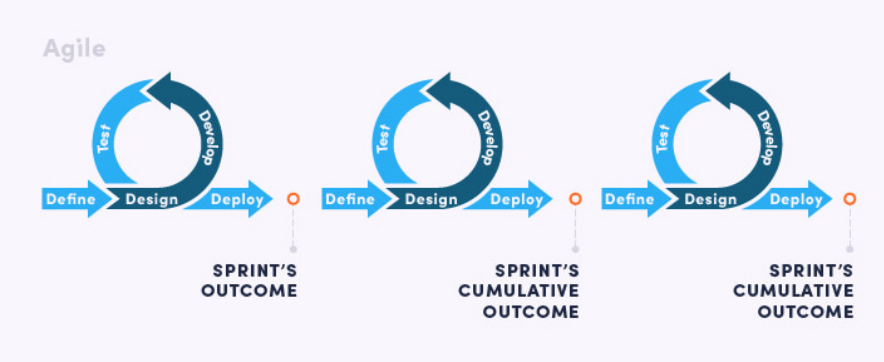

Figure 1: An example of how an Agile Development Procedure may occur during the development of a piece of software.

Since the term 'Software Engineering' was coined by Margaret Hamilton in the 1960s the industry has experienced an explosion of exponential growth. Similar to all industries, many quality control and improvements have been implemented in order to increase productivity and efficiency. However, despite much research and experimentation into the matter a valid and trustworthy measure for the industry has been difficult to produce and in turn many different corporations have implemented their own measure, but struggle to find an exact measure that covers everything (Kaner & Bond, 2004). Finding a suitable metric would allow companies to better distribute their limited resources wisely, and this is of growing importance as more companies embrace software development methods as a more vital element of their business practices.

2. Measurable Data

For all industries, regardless of type, data is the most valuable element required to analyse activity and hence produce a way to improve both processes and their quality. Luckily for software engineering, the data is collected easily through its processes, although, the data that is produced may not be of a comparable nature or at all relevant and hence large-scale improvement of quality may not be possible to produce.

2.1. Types of Measure

There are three types of data that can be used to measure Software Engineering: Product Data, Project Data and Process Data. These can be broken down into attributes and entities. Following this, the attributes can be further broken down into direct attributes and indirect attributes. Direct, or internal, attributes can be directly measured through examination of a product regardless of its behaviour, whereas, indirect, or external, attributes reference how the product relates with its environment. Whilst both have advantages and disadvantages; direct attributes being easier to collect whilst being difficult to interpret, indirect attributes displaying a multitude of effects rather than a singular focus on processes, but require manual input. Therefore, to create a valid measurement system, you must combine both direct and indirect, internal and external, attributes (Xenos & Stavrinoudis, 2008) .

Product Data by far is the most complicated of the measures to collect as it involves collecting information that relates directly to the product, its deliverables, manuals and quality. As these are all variable through the development of a piece of software, the data itself can be quite difficult to define and hence is a complex metric.

Project Data, on the contrary is easily the most accessible data and measure to collect. The only elements of concern when measuring a project are those of the resources that are used during the project. The main entities that are analysed are personnel, environment and tools that are made use of. Even though this data is not unique to just software engineering as a discipline, from the perspective of an organisation, this data is just a necessity to collect, analyse and measure (Xenos & Stavrinoudis, 2008).

The process data of the creation of a software product is another metric which is relatively easy to collect and measure. This is a measure that looks directly at the main activities, mentioned above, that occur during the Software Engineering process, specification, implementation, testing and validifying and maintenance. Looking at these activities allow for the search for best practice during the software engineering process. Some attributes that apply to this measurement include mainly direct attributes, such as time and effort (Xenos & Stavrinoudis).

2.2. Types of Data

The following consists of some of the most straightforward data that one could collect when attempting to measure software engineering

1. Lines of code
2. Number of commits made to a version control provider (e.g. Git)
3. Active Days
4. Code Churn

5. Impact

Lines of code refers to the number of lines of code a programmer working on a software development project creates for an element. However, this proposes a large flow in measure as it is often best practice to create a program that contains as few lines as possible to improve understandability and, often, program efficiency. Hence, adding a metric that implies developers should add more lines of code could lead to a problem with project performance later in the development process.

Number of commits is as flawed as the previous metric in measuring engineering. This is due to the fact that a high number of commits made by a developer highlights that they are active in their programming but hints nothing towards the skill and ability of the programmer. A skilled programmer may do more with less commits and hence the metric would not represent this. Furthermore, it is considered best practice to commit often and actively, so the metric would be affected by this.

Active days is a measure of how much time a software developer contributes code to a specific project, and is similar to the Number of Commits metric. There are a number of flaws with this metric as, firstly, this does not take into account any planning or administration associated with the project. Its purpose is to measure the cost of interruptions to the development of the project. However, it may lead to ethics concerns as it requires a large amount of observation on the developers.

Code churn represents the number of lines of code that were added, deleted or modified over a defined period of time. It can be useful in identifying any problems that may be underlying a project. If there is a high level of code churn, it may indicate that the project is unstable. However, it does not reflect the productivity or performance of a software engineer. For example a high rate of code churn could be attributed to a poor software developer or it could be attributed to poor project specifications and product owners that constantly change their vision for the product, and hence their requirements of the project.

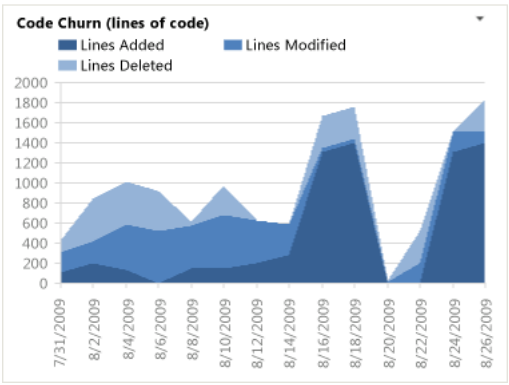

Figure 2: An example of a code churn report from Visual Studio.

Impact refers to the impact any code change on a software development project. It measures how much a code change has affected the processes of the project. A code change that effects multiple files may have more of an impact than a code change that effects a single file. However, this impact can decline massively from the initial changes made to the code to changes made later in the maintenance stage of the software development.

As can be seen, all of these metrics, whilst having some advantages, have many disadvantages when taken alone and hence it is important to ensure many of these data types are taken into account when measuring the activity of a software engineer.

2.3 Conclusion

Naturally, obstacles arise when trying to measure qualitative data, that being it can be very hard to define and standardise said data for measurement. However through correlating data types with others, a form of

quantification can overcome these problems, and in turn produce valuable measures for the software development process.

However, it is important to take into account the effect of this measurement process on the actual human developers. This is a very important aspect to be taken into account, especially for those using the agile development process as the human element of development is one of the priorities of the process. Therefore, a significant effort should be applied on creating a unobtrusive measuring system that does not affect the engineer in their development process. The system should encourage productivity, and avoid diminishing it.

3. Computational Platforms

The data above, once collected must be compiled and analysed in order to be of any use to the organisations and interested parties that wish to view it. In the past number of years, rapid analysis has come to the fore as it allows parties to gain access to the information they want to see in a timely manner and hence respond to any issues or problems that become visible in the reported measurements faster than would have previously been possible. As a result of this, many computational platforms have emerged that allow for this service to occur. However, due to the many variations between organisations such as types of projects, team layouts and individuals, the platforms and measurements differ greatly from organisation to organisation.

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

1. Personal Software Process
2. GitPrime
3. Code Climate
4. Hackystat

3.1 Personal Software Process

Personal Software Process is a structured software development process that was designed initially by Watts Humphrey. Its goal is to help software engineers better understand and improve their performance by developing order in the way that they develop software and aids in tracking their code and even predict future development iterations of the code (Humphrey, 1996). It was initially designed to help developers manage the quality of their products and hence reduce the number of defects in their work. Although studies have proven that increases the overall quality of the software it unfortunately causes major obstruction to the software developer as it requires manual input from the user whilst also relying on human judgement which is seldom objective and unbiased.

3.2 GitPrime

GitPrime is an organisation that gathers information from git repositories, ticketing systems and pull requests and hence transforms them into easy to understand insights and reports. It is very useful for engineering organisations to map their initiatives to proposed outcomes and adjust their processes accordingly (Crunchbase, 2021). This would track similar metrics to those mentioned above, (Lines of Code, Number of Commits etc.), hence all of the problems that come with measuring these will come with this organisation.

3.3 Code Climate

Code Climate is an organisation that develops specialised software for analysis of the software engineering process and the improvement of said process and the quality of its outputs. As stated on their website (2021), “Our mission [is to] empower engineering excellence across people process and code”. Many of their projects focus on combining test coverage and the quality of code committed to give a strong comprehensive overview on a software project for a team of software engineers. Their two major software projects are, Quality, that automates the code review and will improve overall quality of the code, and Velocity, which, similar to GitPrime analyses the GitHub repositories of the owner and collects data on said repositories. However this comes with the same downfalls that came with GitPrime and using multiple platforms would be of a greater benefit.

3.4 Hackystat

Hackystat is an open source framework for the collection of software development production and process data, that aids in the analysis, interpretation, visualisation and annotation of said data. Users of the software typically attach software "sensors" to their development tools that unobtrusively collect raw data from the development that is sent to the "Hackystat Sensorbase" for storage in which the data can be analysed and interpretated (Hackystat, 2021). Although this software is a strong measurement tool, there has been a hesitation in the industry for its uptake. Unlike Code Climate that focuses on product metrics, such as code churn and lines of code, Hackystat relies on constant surveillance of the engineer and collects data on their behaviours, which is a major ethical concern that will be discussed in the ethics section of this report

3.4 Conclusion

Along with the four platforms mentioned there are a multitude of others available for use by organisations and individual developers to track a number of metrics and will visualise the data. However, a vast number focus on the behaviours of developers and hence there is a large overhead in the time, effort and amount of data that is collected. Other platforms require less manual data input and less frequent surveillance that will aid in the ethical concerns that developers may have with the data collection. With a large number of platforms, each with their own benefits and detriments, there is a large amount of importance placed on the platform chosen by an organisation being in line with its goals and principles.

4. Algorithmic Approaches

A trend that is currently on the rise in the analysis and measurement of software engineering is that of its automation thanks to machine learning capabilities increasing over the last 20 years. Machine learning is the study of algorithms that can improve automatically over time through the use of data and experience (Mitchell, 1997). It is derived from pattern recognition and is fundamentally grounded in taking a step away from Procedural programming that is most common in all algorithms. It is used in many forms of analysis in 2021, for example, it is a commonly used speech recognition tool and many implementations of speech-to-text uses machine learning. As such, machine learning enables engineers to make quick analysis of large data sets and rapidly respond to such inputs.

Although there are many approaches to machine learning algorithms, they tend to fall into three broad categories depending on the feedback that is available to the system. These approaches are:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforced Learning

4.1 Supervised Learning

Supervised Learning occurs when the machine is presented with a large base of example inputs and is then showed how these inputs are mapped to their desired outputs. The machine is given a teacher and the general goal of this approach is to gain a general rule that in turn maps inputs to the desired outputs (Bishop, 2006). The theory here is that after a large amount of trial and error the machine will be able to present results that in general match this rule. It is only when a certain level of accuracy is met that then the machine is put into production. Some examples of supervised learning are text classification programs, recommendation systems and regression analysis in statistics.

4.2 Unsupervised Learning

Unsupervised learning is the polar opposite to supervised learning in the sense that no labels are granted to the learning algorithm. This means that the machine is left on its own to structure its inputs and in turn group them accordingly, like the clustering of data points (Bishop, 2006). Instead of responding to the feedback of a teacher, the algorithm identifies common features in the data and reacts based on these features presence or lack thereof in other data sets. The problem with this is that there are no correct answers as such in unsupervised learning, so complete trust must be placed in the algorithm itself. This approach is largely applied to density estimation algorithms in statistics and are hugely important in the field of summarising and explaining data features.

4.3 Reinforced Learning

This involves a computer interacting with a dynamic environment in which a certain goal must be performed. As the program navigates this space, and the program is given feedback, akin to rewarding a dog for performing a trick, which it in turn tries to maximise (Bishop, 2006). This however can be very difficult to achieve as it requires a large amount of memory, especially for more complex implementations of reinforcement learning algorithms. This approach is becoming recognised globally more now than ever with the introduction of autonomous vehicles and has been used for "AI players" in video games for many years.

4.4 Conclusion

Without doubt, the creation and development of machine learning algorithms has aided in the monitoring and measurement of software engineering. With its continuous growth, this area will only become more able to collect and analyse developers data with more accuracy, especially making use of supervised learning and analysing large databases of code written by software developers. However one issue with the machine learning process is the lack of the human element. Even the best algorithms these days do not have close to the awareness and perspective that the human brain brings into data analysis. A rogue variable in a dataset could completely offset the algorithm and hence create issues in the analysis of the data. Therefore, even if they report correctly and accurately 99% of the time, granting too much independence to the analysis these algorithms perform can lead to some inaccuracies that including the human element into could have avoided.

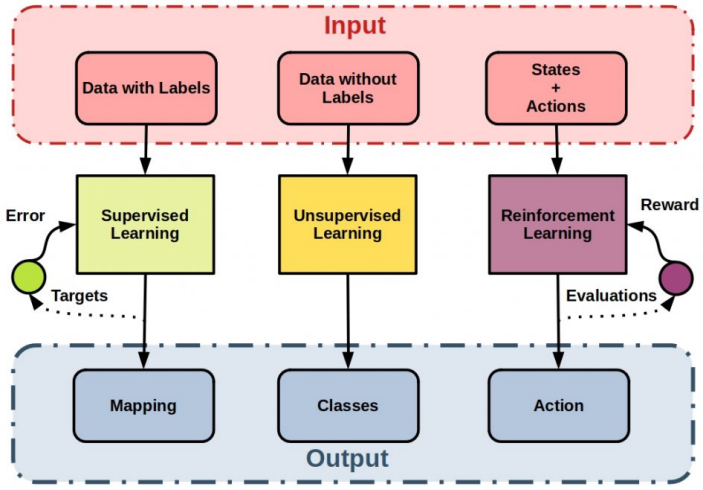

Figure 3: a diagram of the three Machine Learning Approaches discussed

5. Ethics

Over the course of measuring Software Engineering, a certain level of surveillance is going to be required in order to gather the data of the software engineer and analyse it. However, it is important for Organisations to maintain the right of surveillance as without this data, it may be difficult to remain competitive in the industry. Furthermore, although software engineers jobs can be treated similar to a simple standard office job. Contrary to this though, the job of the software engineer consists of making use of several extensive skills and hence measuring the combined work of these skills can be extremely difficult. Following this, the product of a software engineers work has certain standards and expectations to meet, including scalability, compatibility, ease of use, inexpensive among others. Hence, these several factors make the measuring and analysis of software engineering much harder than the measuring of other professions.

5.1 Obligations

In software engineering, the engineer has certain moral and ethical obligations towards society that must be met. Their work must be functioning, transparent and there are many reasons why it must be monitored that will be discussed in this section.

Software Engineers play a vital role in our modern-day digital society, and are required for the continued everyday functioning of our world. The work that these engineers do every day in any company can have massive implications on people around the world. There are programs that are used to correct students tests, to keep track of a person's credit rating and even now there are artificial intelligence programs that can drive our cars for us. Technology is integrated everywhere in the world now and if there is an issue with the code written by an engineer, if it has bugs or there is a cybersecurity weakness, that is a fundamental flaw with how the software interacts with people and as a result could have many negative consequences. Hence, Software engineers hold a lot of power and it is important to ensure that their work is monitored and accounted for.

In 2018, the Association of Computing Machinery (ACM) introduced a code of ethics that should be followed by software engineers worldwide, consisting of General Ethical Principles, Professional Responsibilities, Professional Leadership Principles and Compliance With The Code. It is stated that "*The Code is designed to inspire and guide the ethical conduct of all computing professionals"* (ACM, 2018). However, this code of ethics is not required to be followed under any form of legislation, which highlights the difficulty that governments and regulatory bodies have in keeping up with the development of software. As a result of this it is up to the engineer to hold themselves accountable for any code they develop. Failing this, software engineering must be measured, analysed and monitored to keep software developers accountable.

It must be said that a vast majority of software engineers do not set out to develop code that will do any form of harm. The number of skillsets drawn from by developers are difficult enough to manage and control before taking any form of ethical obligation into account. Following this, it is extremely difficult to determine the good or harm that a piece of software will do in society before it is released to, and interacted with by, the public. As a result, if software engineering is controlled and monitored, it will no longer fall on the software developer to do this themselves.

5.2 Legality

Despite the monitoring and measuring of software engineering being a vital aspect of the development of software, there are many legal aspects that must be followed when performing this monitoring. The EU's General Data Protection Regulation (GDPR) was introduced May 25, 2018 and with it brought a number of obligations to organisations worldwide that collect data from any person related to the EU. It highlights that data collected from a person can only be held with their consent. If this rule is not followed there are large levies and fines that organisations must pay, these maxing out at the higher of 4% of the company's global revenue or a €20 million (GDPR EU, 2021).

Since the formation of GDPR, many other places have begun introducing data protection laws that protect the data of citizens, including the California Consumer Privacy Act (CCPA), Brazil's Lei Geral de Proteção de Dados (LGPD) and South Africa's Protection of Personal Information (POPI), Amongst many others that have been introduced worldwide (UNCTAD, 2021). However, the interesting thing about all of these legislation is that they do not cover the monitoring and collection of data in the workplace, they focus nearly entirely on the collection of personal data. Therefore, if an employee was to do personal work whilst they were in work, the data collected by the company would have to be treated differently to the data collected when monitoring the software engineering process in order to avoid the hefty fines that are imposed on corporations that are in breach of these data protection regulations.

5.3 Data

A significant element of these new data protection laws mentioned above is the fact that these laws allow you to control who has access to your data and in turn manage what is done with said data. This is relevant to software engineers as employees as they have the right to know who or what is monitoring their work and a lack of disclosure can cause a lot of tension in workplaces between management and employees.

As a result of this it is extremely important for corporations to be transparent in what they collect from employees, whether it be lines of code, number of commits etc., how they are collecting this data, are they using a professionally licenced software, like those mentioned above, or will they be using their own in-house monitoring tool. They should also be clear on how the data they collect is being used and what is being done based off of the information collected using these metrics.

By a company being transparent with usage of data, employees can be assured that their work is being treated fairly and impartially and that no discrimination occurs in the measurement of the data. This in turn allows employees to grow naturally and at their own pace as a software developer and potentially move up in the corporation's ladder.

5.4 Impact

The most important aspect of the ethical implications of measuring software engineering is that of the impact it can have on the individual software engineers. The consistent surveillance, collection and analysis of data can be the cause of a lot of stress and pressure for developers and could even possibly lead to an increase in the number of mistakes due to the pressure that is now looming over them.

Following this, it may become extremely difficult for employees to remain innovative under these surveillance conditions. As the monitoring of the data is done in order to improve efficiency, it would become very difficult for a developer to take time away from their work to take a risk and try something new as it would take away from their overall efficiency and the data may reflect this poorly. Despite this, it is widely known that innovation and efficiency are of equal importance in the tech industry as they are both required for a company to stay relevant and important in an industry that continues to grow and change exponentially every day.

In my own opinion, it is important that the surveillance conducted on software engineers is minimally invasive and does not impact on their work in the slightest. Following this, I personally have many issues with the constant collection and monitoring of "big data" that has become the norm in our lives from the likes of Facebook (now Meta) and Google. Therefore, I believe that the data protection laws mentioned earlier in the report should be expanded to not just include personal data, but limit the collection of data in the workplace, over all professions not just software developers, as employees should have the right to claim ownership over their own work, even if not stipulated in a contract of employment. It is very clear that in this day and age the line between data collection and outright surveillance has become very foggy and is most definitely a defining issue in the monitoring of software engineering.

6. Conclusion

This report has discussed in detail the many different aspects that must be accounted for when defining and defining the systems that are involved in the process of measuring software engineering. A small number of the many metrics that could be used to measure software engineering were discussed alongside their advantages, disadvantages and complexities that are involved with their collection. Following this a number of software that are available to use for the measuring of software engineering were highlighted and their aid in making the measurement of software engineering more available was discussed. The rise of machine learning and how this would allow for an ease in the measuring of software development was mentioned and how further development would possibly make measurement of development more feasible and accessible in the future. Finally the ethical concerns regarding the process were discussed and important references to ethical decision making and monitoring rather than surveilling being favourable were highlighted alongside the obligations a software engineer has and the essential need for a measure of software engineering to allow for accountability. In conclusion, the process of measuring software engineering and development is a long, complicated and complex system that has many variables and issues that come alongside designing and developing a system that will perform the required data analysis and visualisation. As a result of all these moving parts, it is important for organisations, when developing these measurement processes, to take into account the needs and required analysis for both the developers and the organisation as a whole and in turn, design a system around these needs.

7. Reference List

- Laplante, P. (2007), "*What every engineer should know about software engineering*", Boca Raton: CRC.
- Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E. & Sarwar, S., Z. (2010), "*Agile Software Development: Impact on Productivity and Quality*", Published in 2010 IEEE International Conference on Management of Innovation &Technology.
- Kaner, C., Bond, W., P. (2004), "*Software Engineering Metrics: What Do They Measure and How Do We Know*", In Metrics 2004, IEEE CS.
- Xenos, M. N., Stavrinoudis, D. (2008), "*Comparing Internal and External Software Quality Measurements*", Proceedings of the 8th Joint Conference on Knowledge Based Software Engineering
- Humphrey, W., S. (1996), "*Using a defined and measured Personal Software Process*" IEEE Software (May, pp. 77-88).
- Crunchbase (2021), "*Organisation – GitPrime*" , accessed from https://www.crunchbase.com/organization/gitprime, Retrieved on (18 Nov 2021).
- Hackystat (2021), "*Hackystat*" accessed from https://hackystat.github.io/ Retrieved on (19 Nov. 21).
- Mitchell, T. (1997), "*Machine Learning"*, McGraw Hill: New York.
- Bishop, C. M. (2006), "*Pattern Learning and Recognition"*, Springer.
- ACM (2018), "*ACM Code Of Ethics and Professional Conduct"* accessed on https://www.acm.org/code-of-ethics Retrieved on (01 Dec 2021).
- GDPR EU (2021), "*What is GDPR"* accessed on https://gdpr.eu/what-is-gdpr/ Retrieved on (8 Dec 2021).
- UNCTAD (2021), "*Data Protection and Legislation Worldwide"* accessed on https://unctad.org/page/data-protection-and-privacy-legislation-worldwide Retrieved on 8 Dec 2021.
-