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

CSU33012: Software Engineering
Measuring Software Engineering

Ahmed Hamed Aly, 18308279

November 14, 2020

Contents

1 Measurable Data

Measuring software is vital in the discipline of software engineering[1]. People involved in IT fields are always facing new and advanced technologies along with a very competitive market for various other things in the discipline. While doing so, they also have to worry about the reliability of the product, the stability of the product, the testing of the product. This produces a need for software measurement to aid for these worries.

Although software measurement alone cannot solve these problems faced by people in IT, it can help elucidate and focus your understanding of the problems. Also, when the software measurement is successful, sequential measurements of the quality attributes of products and processes can provide an effective basis for initiating and managing process improvement activities.

There is different data that can be acquired on an individual software engineer. For this report, I have divided it into three categories: Code - data about the type of work the engineer is doing, External - data about the work environment, and Internal - data specific to the person being measured.

1.1 Code

1.1.1 Quantity

The simplest way to measure the amount of work done by a software engineer is through the concept of Source Lines Of Code (SLOC)[2]. This is a software metric used to measure the size of a computer program by counting the number of lines of text in the program's source code. SLOC is primarily used as an indicator for programmers to see how much work remains to be done for the program to be complete. This metric is also used to estimate the productivity and maintainability of the program after the software has been produced.

Although this is a really good indication of the work done, there are several issues with this method. Simply measuring performance by the volume of code produced does not take into account the format of the code written, for example, the following snippets are identical in their effect:

```
1   for (int i = 0; i < 100; i++) printf("hello");
```

Figure 1: Lines Of Code (LOC)

```
1   int i = 0;
2   while (i < 100) {
3       printf("hello");
4       i++;
5   }
```

Figure 2: Logical Lines Of Code (LLOC)

Simply measuring the volume of code produced by a software engineer could lead to people fill in their performance statistics by writing very detailed code.

There are two primary types of SLOC measures:
a) Physical Lines of Code (LOC)[3]: This is a count of lines in the program's source code text that includes comment lines and blank lines unless the lines of code in a section consisting of more than 25% blank lines.

b) Logical Lines Of Code (LLOC): Measures the number of "statements", however, their specific definitions are related to specific computer languages i.e. a simple logical LLOC measure for C-like programming languages is the number of statement-terminating semicolons.

It's much easier to create tools that measure physical LOC. Furthermore, physical LOC definitions are easier to explain than LLOC.

1.1.2 Bugs

One method used to analyse code quality is by measuring how many errors or errors exist in an average line of code[4]. This is usually measured in errors per n lines of code and provides an idea of the quality of the software being written, as well as the skill of the engineer writing it and their familiarity with the tools they are using.

Fixing bugs can often be a more complicated process than writing the original code. This process requires understanding the problem code and its interaction with other parts of the software. Some errors can be very difficult to understand, let alone fix them, which can lead to slow resolution[5]. During this time, the amount of code a developer produces can be quite low, making it difficult to measure their performance because it is dependent on the difficulty of the problem that is again difficult to measure.

1.1.3 Performance

Another method to analyse the quality of code is by measuring the software engineers performance over time.

One way to measure work performance is to consider the amount of code produced versus time. That is, evaluate SLOC or LLOC as a function of time. This information can be very useful in assessing the performance of a software engineer, but it also has some pitfalls, e.g. it does not take into account the difficulty of the job.

Another metric that can be used is the "churn rate,"[6] which is the percentage of a developer's code that is a modification of his most recent work, ie how much of their code they have to rewrite. A high churn rate may indicate that the code being written is of poor quality, while a low churn rate may indicate that the code was written with higher quality. The churn rate can also indicate how clear the purpose of the project is, as constantly changing goals lead to a higher churn rate.

Commit frequency could also be used to gain insight into the software development process[7]. The number of commits in a day, as well as the pattern of commits over time, could be used to analyse performance. For example, does an engineer commit very few times a day or does the amount of work they do reduce towards the end of the week.

1.1.4 Review

Pull requests or code reviews can also provide useful insight into the work being done by a team of software engineers[8]. Because pull requests allow other team members commenting on code written by others can be an opportunity to gain insight system problems in the team and also find ways to improve team efficiency.

During the code, it checks the code readability, as well as its transparency commented, can also be measured. While this can be quite subjective, it can help improve processes by driving the change of coding standards.

How feedback is delivered to the original developer during the download a request can also indicate how well the team is working together. An example could determine if the feedback is constructive or just critical.

1.2 External & Internal Factors

1.2.1 Workplace Environment

[9] Variables such as workplace temperature and levels of light sources can affect the performance of a software engineer. Office temperature has a measurable effect on productivity, and too high or too low temperatures have a detrimental effect on the productivity of those working in the environment. The type and quality of lighting in the workplace can also have a significant impact on individual performance. If the ultimate goal of measuring performance is to improve it, this should also be considered.

Conversations between software engineers in the workplace can be analysed to gain insight into the tone or length of the conversation[10]. Besides, the content of the conversation can be analysed to infer how focused the programmers are in their work. This information can also provide some insight into team dynamics by providing detailed information about how colleagues treat each other in the workplace.

1.2.2 Health

Measuring your body posture while working can provide information on software engineers focus as well as provide recommendations for improving comfort, performance, and long-term health. By combining this information with other data such as heart rate, respiratory rate and blood oxygen levels, it was possible to measure stress and establish links between stress and performance.[11]

2 Computational Platforms

Many computational programs aid in the measurement of software engineering processes such as GitPrime, Leap and Hackystat. These will be the platforms that I will talk about in-depth.

2.1 Platforms

2.1.1 GitPrime

[12] GitPrime is a cloud service that analyses data collected from a version control system used by a company or an individual. They claim to analyse the large number of variables surrounding the code being produced, as well as pull requests and the type of interaction between engineers in pull

requests. In doing so, they claim to be able to provide insight into the performance of a particular engineer or team over time. They provide data on speed, type of work done, churn etc.

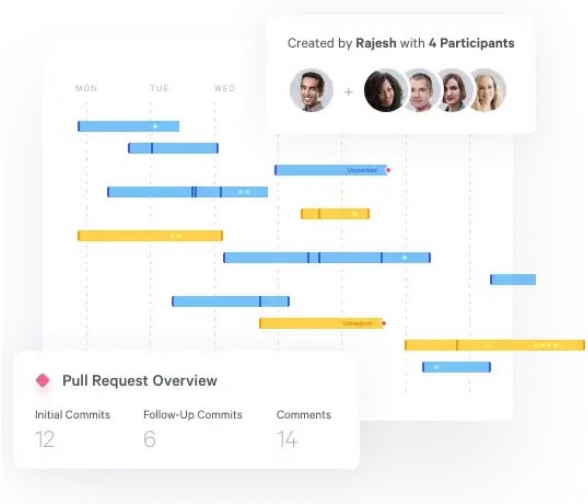

Figure 3: Usage of GitPrime

2.1.2 Leap

[13] The Leap toolkit was created to provide lightweight, empirical, anti-measurement and portable approaches to software improvement. If software engineers use Leap, they can collect and analyse personal information about time, size, defects, patterns, and checklists. There are two main activities on the Leap platform. They collect basic data and perform Leap analysis. These can be augmented by definitions, checklists and benchmarks. Finally, these central and/or secondary activities can be targeted at individual skill acquisition and improvement or group review of work products.

2.2 Frameworks

2.2.1 Hackystat

[14] Hackystat is an open-source platform that is used to automatically collect data and analyse product data and software engineers. The goal of the Hackystat platform is to provide an extensible engine that can completely reduce the overhead of collecting a wide variety of software engineering data. Also, the platform includes a set of analysis tools that can make a useful report. Hackystat is widely used in application areas. This may include school education, software engineering for high-performance computer systems.

2.3 Personal Software Process

[15] The Personal Software Process (PSP) is a software development process designed to help software engineers better understand and improve their performance by tracking the projected and actual development of their source code. Watts Humphrey created this platform to apply the basic principles of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to software development practices by a single developer. The idea behind this platform is to provide software engineers with the process skills necessary to work in a team software process (TSP) team

3 Algorithmic Approaches

3.1 Machine Learning Analysis

Machine learning is an application of artificial intelligence (AI) that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed.

Machine learning focuses on getting the computer program to be able to grow and learn from themselves rather than from through programmed instructions that tell the program what to do. Machine Learning is divided into 3 types of algorithms:

1) Unsupervised Learning[16]
2) Supervised Learning[16]
3) Reinforcement Learning[17]

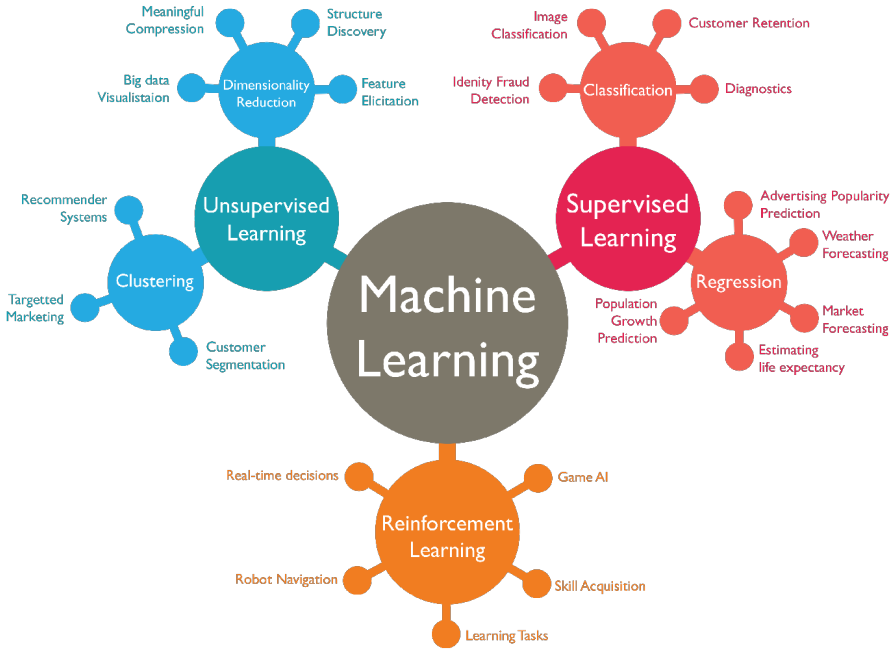

Figure 4: 3 Types of Machine Learning

3.1.1 Supervised Learning

Supervised learning is the most commonly used machine learning in practice. The way supervised learning works is that you have an input (x) and an output variable (Y), and you use the algorithm to learn the input-to-output mapping function i.e. Y = f (X).

The goal of supervised learning is to accurately estimate the mapping function so that when new inputs (x) appear, the output fields (Y) can be predicted for that data.
This is called supervised learning because the learning algorithm process based on the training data set can be viewed as an overlook teacher and oversight teacher. We know the correct answers, the algorithm iteratively forecasts training data and is corrected by the teacher. The learning process ends when the algorithm reaches a satisfactory level of performance.

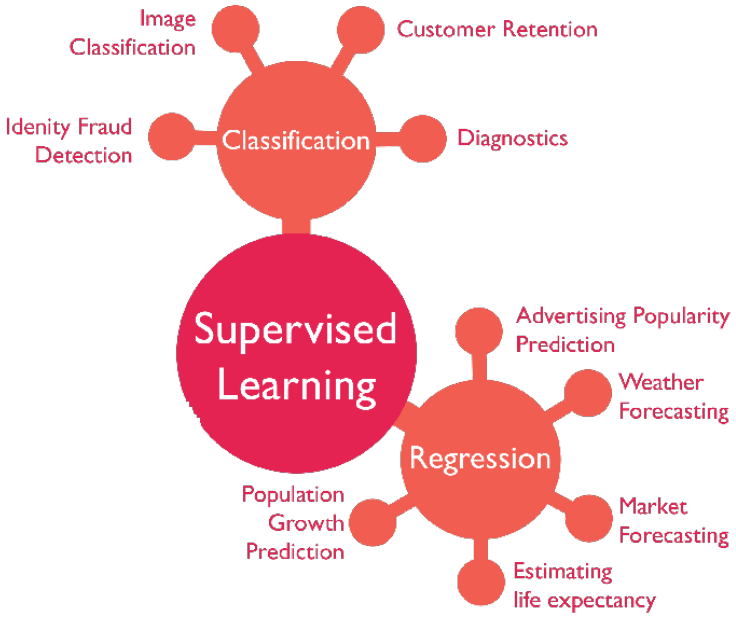

Figure 5: Supervised Machine Learning

However, as a result of this type of machine learning, two kinds of problems can arise: 1) The classification problem This involves taking input data and deciding which class/category the input belongs to based on the training data. This is the case when the predicted result is a category, such as Red, Blue, etc. For example, an algorithm that sorts vehicles by colour or type.

2) Regression problem Regression is about estimating the relationship between variables. It is used in cases where the expected result is a real value, such as weight, currency, etc. For example, an algorithm that determines energy consumption over a specific period for a given country.

3.1.2 Unsupervised Learning

Unsupervised learning is a type of machine learning algorithm that is used to conclude from data sets consisting of inputs without tagged responses. Starting by analyzing a known set of training data, the learning algorithm creates an inferred function that can predict output values. The system can set targets for each new contribution after sufficient training. The learning algorithm can also compare its output to the correct intended results and find errors to modify the model accordingly.

Cluster analysis is the most popular type of unsupervised learning. Used to find hidden patterns and grouping based on data analysis. Clusters are modelled using a similarity measure that is defined by metrics such as Euclidean or probabilistic distance.

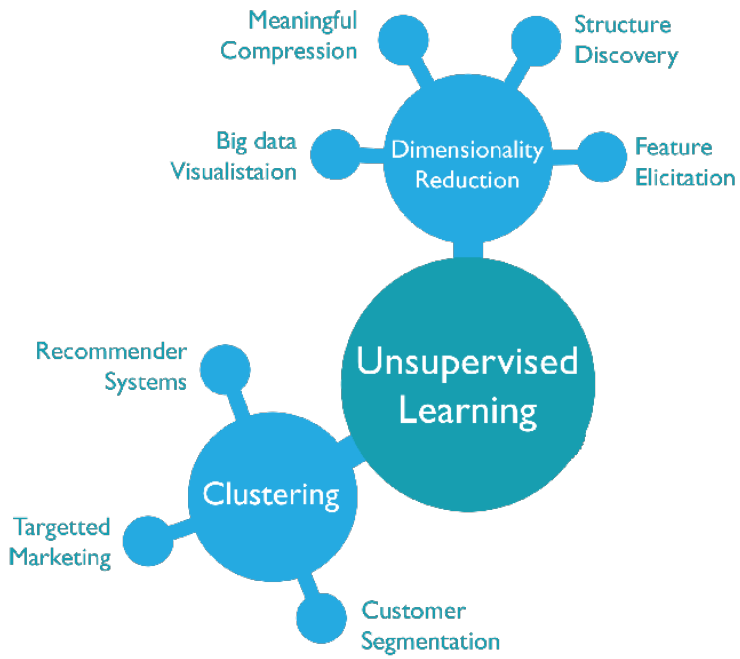

Figure 6: Unsupervised Machine Learning

3.1.3 Reinforcement Learning

Reinforcement learning is a type of machine learning algorithm that allows a machine to learn its behaviour based on feedback from its environment. This behaviour can be learned once and for all, or it can be adapted over time. If the problem is created carefully, some gain learning algorithms may converge to the global optimum; this is the ideal behaviour that maximizes the reward.

Figure 7: Reinforcement Machine Learning

This automated learning scheme means there is no need to hire an expert who knows your field of application. The advantage of this type is that we spend less time designing the model as there is no need to manually create complicated rule sets. Moreover, all that is needed for this is the academic proficiency with reinforcement.

4 Ethical Concerns

4.1 Gathering Data

Gathering data based on code is of little ethical importance as the engineer already provides the data[18]. Since this data is a work provided by a developer, it is ethical to analyse it, however, a software engineer could manipulate it to appear more productive, as seen in the previous SLOC example.

The collection of personal and health data is highly unethical. The continual collection of data such as heart rate or blood pressure gives you a huge insight into a person's life, which in my opinion an employer should not have access to. Collecting this data could allow employers to control aspects of a person's life outside of work, for example, if a smartwatch or Fitbit shows that an employee has not exercised recently, his employer could be informed. This leads to the question of whether the employer should be allowed to set expectations of what is appropriate in terms of exercise or sleep? Another example would be employee voice data recording, person-to-person call recording. This type of monitoring is very invasive and can put a lot of pressure on employees to say the "right thing". It could also give the employer the ability to control people's thoughts and opinions.

Once collected, this data should be safely stored due to its sensitive nature. There have been many large-scale data breaches in recent years, such as the Equifax case[19] where social security numbers, dates of birth, addresses and even driver's licenses were stolen from 143 million Americans. In August 2013, every Yahoo customer's data, more than 3 billion accounts, was stolen. Given the scale of these corporations and the sensitivity of the data they had access to, it can be assumed that they will take steps to adequately safeguard this information.

However, if these huge companies have proven vulnerable to attacks, it can be reasoned that if large amounts of software companies were collecting and storing their employee's data, such as health data, there is a good chance it would be stolen. This data does not have to be stolen by someone outside the company, several people in the company may have access to this sensitive information. It would be very important to choose carefully who should be able to access this information.

Data persistence is another problem, many companies keep multiple backups of their data, this is essential to prevent data loss. However, if the company had a large amount of personal information about the employee and then that person leaves the company, should they be allowed to retain the data?

4.2 Analysing Data

This can be boiled down to: Is it okay to trust a machine to measure a person?[20] There are many problems with the techniques used to analyse this data, many of which are related to machine learning. One of the main problems with using this algorithm is that it cannot guarantee correct results. For example, researchers were able to trick image recognition programs into identifying cats as guacamole. In another more serious example, the self-driving test vehicle failed to recognize people crossing the road and killed pedestrians. Although the vehicle's sensors have detected pedestrians in advance, this is still feasible. This highlights the risk of allowing the computer to make important decisions.

Another problem with using these technologies is that we have very little ability to see the inner workings of such an algorithm. This can make troubleshooting very difficult. This shows a key problem with machine learning, the lack of control over what the software does. These algorithms have also been shown to be biased, if the algorithm were trained primarily on data (voice data of intercourse conversations) containing male voices (who now make up the majority of software engineers), it could theoretically induce a tendency towards men and women below. This difference may only be very subtle, but it can still have a significant impact on the perceived skill or value of the software engineer. Given these concerns, it is reasonable to question whether the results obtained from analyzing these data are an accurate measure of the individual's performance.

4.3 Location Privacy

Location-Based Tracking Systems (LTS)[21] use a wide variety of different technologies to record the location of objects and especially users. The use of LTS affects the security and privacy of users. When using an app that accesses your location, you don't know what other apps or companies also know about your location, which adds to the security and privacy concern of users. This allows them to use your data and then recommend ads to you based on your location. For example, by developing mobile apps that track a user's location in real-time, such as Snapchat, in-app ads can even be tailored to the user's current location.

References

[1] https://link.springer.com/article/10.1007/BF02249053.

[2] https://en.wikipedia.org/wiki/Source_lines_of_code.

[3] https://www.coursehero.com/file/23009608/SLOC/.

[4] https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio.

[5] https://dev.to/wemake-services/best-engineering-practices-how-to-fix-a-bug-58g5.

[6] https://en.wikipedia.org/wiki/Churn_rate.

[7] https://softwareengineering.stackexchange.com/questions/74764/how-often-should-i-do-you-make-commits.

[8] https://jserd.springeropen.com/articles/10.1186/s40411-018-0058-0.

[9] https://indoor.lbl.gov/sites/all/files/lbnl-60946.pdf.

[10] https://hackernoon.com/how-you-can-measure-and-evaluate-performance-of-software-engi

[11] https://dev.to/jacobjzhang/solving-the-health-problems-of-software-engineers-3h3j.

[12] https://www.pluralsight.com/product/flow.

[13] https://www.semanticscholar.org/paper/Leap%3A-a-%22personal-information-environment%22-for-Johnson/e63a598fac89d00f156232e7e22b879bb1326f63?p2df.

[14] https://hackystat.github.io/.

[15] https://en.wikipedia.org/wiki/Personal_software_process.

[16] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algor

[17] https://en.wikipedia.org/wiki/Reinforcement_learning.

[18] https://www.hrmagazine.co.uk/article-details/the-ethics-of-gathering-employee-data.

[19] https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-w
html.

[20] https://medium.com/coinmonks/ai-doesnt-have-to-be-conscious-to-be-harmful-385d143bd31

[21] https://en.wikipedia.org/wiki/Location-based_service.