Measuring Software Engineering Report

Filip Kowalski - 19334250

Abstract

In this document I discuss and research a variety of techniques and tools used for measuring the effectiveness of Software Engineers and assessing Software Development.
The topics covered in this document are the data we can observe and measure, tools and platforms used to gather and analyze this data, the ways we can go about understanding the data, i.e. the techniques used for data computation and the ethical concerns regarding this kind of measurement.

Introduction

Software engineering is the process of obtaining and analyzing user requirements and then designing, creating and testing software that will satisfy those requirements, Fritz Bauer defines Software engineering as “the establishment and use of sound engineering principles to obtain, economically, software which is reliable and works on real machines efficiently”.

This report will explore the concept of measuring Software Engineers and will work on exploring ways in which workers are measured, ranked against each other and how software development methodologies compare against each other.

It is imperative that we use data left behind during the software engineering process to improve the efficiency and effectiveness of that process as time to develop new software shortens as time moves on. “Due to the great diversity of products made by means of software engineering, no single type of dataset, which can be interpreted through metrics, is applicable to all software engineering efforts” - (Using Data to Make Decisions in Software Engineering: Providing a Method to our Madness - sciencedirect.com). This is why this report will not settle on one method or one type of data to rule them all when it comes to messing the SWE (Software Engineering) process and instead, will seek to explore as many areas of this topic as possible as different products pose different challenges that cannot be solved using the same type of data or analysis process.

Software Engineering Methodologies

Before we can analyze data left by SWEs we must first understand the most popular methodologies that are the basis for many methodologies real SWE teams use, Waterfall and Agile.

Waterfall

This methodology involves a linear approach to the development of a product, it is where customer requirements are gathered at the beginning of a project, and then a linear plan is created to facilitate these requirements, it is split into 5 main parts. Requirements engineering, where requirements are gathered from a customer and a requirements document is created. In the design phase, in this part ideas are theorised and are made into concrete requirements. Implementation is where the product gets created. Verification and testing, the solution is verified to match the requirements and is unit tested to make sure ideal operation. Finally, maintenance where the project is deployed and inevitable problems are found and fixed

Agile

This methodology involves an iterative process of software development, where the requirements and solutions evolve through collaboration between a client and the team working on the project, it emphasises working software over comprehensive documentation and Customer collaboration over contract negotiations, it is a move fluid methodology that encourages a team to respond to change in a positive manner rather than following a plan. In this methodology, requirements are developed incrementally as the project progresses and as the client sees their product being developed.

The point here is that measuring an SWE who works in a waterfall team by the amount of failing test cases they write or how many failing test cases they solve is ineffective for 4/5ths of the developments cycle, while in the Agile methodology, especially test-driven development this would be an assessment that could be considered. And measuring an SWE who works in an agile environment by how many requirements they have implemented in a given day is similarly futile as requirements evolve over time and could even be completely different tomorrow too as they are today.

Observable And Measurable data

1. Lines Of Code

Measuring the lines of code a Software developer has written is a very basic way of measurement that has been around for a long time, it is, however, inherently flawed. Software Developers strive for efficiency and readability in their code however this assessment method promotes exactly the opposite of this goal, in addition to making the code hard to read and understand it causes multiple issues depending on the way it's implemented. If more lines of code are written mean the engineer has done a better job or vice versa it promotes inefficiency or bad readability. In the book “The Mythical Man-Month” explains how the Lines Of Code measurement is an inherently flawed mechanism by demanding more lines of code from software engineers and somehow expecting a faster or better result to putting 9 women on the job of pregnancy and expecting a baby in 1 month.

For example take these simple functions that print the contents of an array of integers, each of them behave exactly alike.

```
printIntArray1(Int[] arr) {
    for(int i = 0; i < arr.length; i++) {
        System.out.println(arr[i]);
    }
}
```

```
printIntArray2(Int[] arr) {
    Int i = 0;
    for(while i < arr.length) {
        System.out.println(arr[i]);
        i++;
    }
}
```

```
printIntArray3(Int[] arr) {for(int i = 0; i < arr.length; i++) {System.out.println(arr[i]);}}
```

Each of these functions behaves exactly alike, they are all valid but we can see it is easy to game systems that require more lines of code by simply replacing a for loop with a while loop like in printIntArray2 and many other ways, we can also see the code is less readable. In printIntArray3 a system where a system that requires fewer lines of code to promote efficiency can be gamed by condensing an entire function to one line, this again is less readable than the original code.

Through this means of assessment, the programmer is encouraged to write redundant lines of code or code that is squashed and less readable.

2. Number Of Commits

This metric is usually used to reassure a client that the project is still being worked on, as the client can see changes and additions to the project as it is being developed, again however it is a flawed approach that does not show how far from completion the project is or even if it is changing. This metric also gives no sign of the quality of work committed to the project. For example, one programmer could commit a change to a variable name and another could commit a whole new functionality to a project, they have not done equal amounts of work, but if this metric was to be believed, they have done equal amounts of work. What should or should not be committed to a codebase is a subjective matter, one SWE could commit a change to comment and another might wait until they have more significant changes made to commit along with the change to that comment. It also does not take into account that code could be removed in a commit. Due to these reasons, the number of commits should only ever be used to check that the project is still being worked on and not abandoned

3. Code Churn

Code churn also known as code rework is a metric that indicates how often a piece of code, like a class or function, gets modified, usually within two weeks of that code being committed, high code churn could signify that a team member writes lazy code that needs to be reworked or writes code that doesn't cover all use cases. Code churn however is not

always a problem, all projects will have some degree of code churn no matter how well a team understands requirements or how skilled they are, better solutions can always come up or requirements change and a specific piece of code needs to perform better or more than it did before.
Code churn is a decent metric that can be used as it can find problems with the engineering process a team is using, problems in a team or even can be a symptom of problems in the application. It could also mean that a project is simply difficult so it is important to use this metric in a known context. It also rewards engineers for finding better ways to solve a problem before committing to a solution

4. Code Test Coverage

Test coverage simply measures the percentage of code that is run through tests that an engineer has written, generally, code that has a high code coverage percentage is more likely to work with all use cases and less likely to have undiscovered bugs, this gives an indication of the quality of work a team has produced to the client and managers.

5. Fixed Failing Test Cases Or Bug Fixes

This metric could be especially useful in test-driven development where test cases are written prior to a function that describes the behaviour of the function, the higher percentage of those functions test cases that a developer could solve in a day could indicate their effectiveness or skill level as a software engineer. In conjunction with this, Bugs that are found in the code could be given a priority level and fixing a higher priority bug could indicate progress and hard work from a developer, an extension to this could be if the bugs were tracked and similar bugs came up in the future, those bugs could more easily be solved and similar bugs in code were solved in the past

6. Product Uptime

This metric is easily quantifiable and understandable, simply put, after the project has been made publicly available how much of that time has it spent unavailable to customers or down due to factors not considered in the implementation. Simply, the higher the uptime percentage the better, every time a product is unavailable or goes down that is a distraction to the team that created it as they need to fix it and a cost to the business overall as the product is not making any money. This metric is good if used to measure the reliability and overall skill of a team.

7. Closed Tickets

Many projects monitor tasks and work to be done with tickets for individual developers, one can measure how much work a certain developer has done during a time interval by counting the number of tickets that the developer closes. It is possible that some developers could close a large number of tickets without doing much meaningful work but if the tickets are properly written, i.e. they are of similar size and complexity then it is possible to measure a developers performance by simply counting how many tickets they have closed, or tickets could be given a point system based on the amount of complexity or work required to solve

them, then a developers efficiency could be measured by the number of points gained from completing tickets.

8. Items Covered In A Standup Meeting

Many teams have daily standup meetings where team members answer the same questions every morning, “what did you accomplish yesterday? What are your plans for today? What challenges are you facing?” This metric seeks to quantify developer productivity by finding out if a developer has met or exceeded their own expectations. A developer could be considered productive if in a previous meeting they said they will work on a feature and in the next meeting they said that they finished that feature, in the long run, they could be considered productive if they exceeded their expectations more times than they subceeded their expectations, and less productive if the opposite is true or the gap is smaller between success and failure.

Tools And Platforms Available

Raw data is hard to interpret without it being processed and or visualised in some way to extract meaningful information from the data, some metrics like for example code churn can be easily be collected but are hard to interpret. There are many tools available for analysing and/or collecting raw data.

1. GitHub

Github is one of the most popular version control systems used around the world. GitHub can not only be used as a version control tool but it can also be used to measure the software engineering process. For example, through the GitHub REST API, A lot of information can be found in the history of a repository, in my visualisation of data gathered from the GitHub API I mapped the total number of changes, additions and deletions per commit, per user over the course of a repositories existence.

2. Hackystat

“Hackystat is an open-source framework for the collection, analysis, visualisation, interpretation, annotation and dissemination of software development process and product data” - Google-Code. Hackystat is easily used with tools like GitHub. Hackystate collectors raw data from developers and sends this data to a server to be analyzed. Hackystat collects data in a way that doesn't inform the person about which the data is being harvested. Hackystat gathers data such as the number of commits in the project but also has more advanced tools like calculating code coverage from test classes in a project and the complexity of a project
This data helps to assess the productivity of a software engineer and even the overall productivity of a team but assessing a project as a whole.

3. GoodData

GoodData is a data visualisation tool, it requires pre-collected data however it offers powerful tools to create charts and graphs quickly and easily. GoodData can be used to create comprehensive dashboards and reports to identify a project or individuals strengths and weaknesses with pre-gathered data. This tool can be especially useful for non-technical managers to visualise data about their Software Engineer teams.

4. LinearB

LinearB correlates data from a project using Git and CI/CD tools to provide insights into the health and speed of a delivery pipeline. LinearB automatically tracks git activity in relation to agile sprints and creates estimated delivery times based on the complexity of a project. LinearB can be used to track cycle time, deploy frequency, code refactorisation, even Coding time and Review Time. it can track closed ticket amount per developer. Using these metrics LinearB collects it can compare “Effort” certain team members put toward a project and can track issues such as team members having too many tasks or open tickets.

Using LinearB companies such as intsights have decreased their cycle time by 55%, a tool like this can be extremely useful to managers as it provides pre-collected and pre-analyzed data to a manager and can even determine if work is split unevenly among team members.

5. Haystack

Haystack is similar to LinearB, it analyzes GitHub data and provides team level statistics that help analyze a team and a project. Haystack can collect an visualise metrics such as cycle time, deployment frequency, change failure rate and throughput. Not only can Haystack display such metrics but it can also analyze and provide a diagnosis on what might be bottlenecking progress on a given project and can analyze potential burnout when throughput is high or when code reviews take longer than expected.

Haystack can be less useful to non-technical managers as it doesn't provide the same level of data analysis as LinearB however it is still very useful to analyze a teams performance, weaknesses and overall efficiency.

6. Code Climate

Code climate offers data analytic tools to measure the development process. “Code Climate’s flagship product, Velocity, analyzes all the data from your GitHub repos and provides you with heads-up displays, real-time analytics, and custom reports to give you a clearer perspective on how your engineering team is working.” - Code Climate 2021.

Code Climate uses metrics to summerise different aspects of the code, code climate claims to help ensure full test coverage, decrease code complexity, simplify the reuse of code and ensure that committed code has intrinsic value to the project.

7. Embold

“Embold is a tool to analyze, diagnose, transform and sustain software efficiently” - Embold, Embold uses A.I. and machine learning technolog to prioritize issues in a project and suggest ways to solve them. Embold analuzez data from a code base and can can give a summary on the efficiency of code inside of the codebase as well as vulnerabilities, code issues, code duplication and more. Embold helps a team work more efficiently by helping spot issues that can cause major problems in later development. Embold integrates with tools such as Bitbucket, GitHub and Visual Studio.

Techniques For Data Computation

The tools mentioned above all use some sort of algorithm to manipulate and analyze data. May that be humen written functions and analysis programs or AI, In this part of the report i will focus on how developers use different types of machine learning to automate their data analysis.

Through AI we can “program” a computer to perform certain tasks without having to hardcode the tasks ourselves, most of the above tools would employ some kind of machine learning.

Machine Learning

Hewlett Packard Enterprises defines machine learning as “a sub-category of A.I. and effectively automates the process of analytical model building” they also state that “The sheer volume and variety of data being generated as humans and other environmental forces interact with technology would be impossible to process and draw insights from without the speed and sophistication of machine learning.”

At its core machine learning is a computer looking at data and identifying patterns, which is exactly what we are looking for in implementation to measure Software Engineering.

A machine learning algorithm must be trained however and there are 4 main ways of training a machine learning algorithm.

1. Supervised learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforcement learning

In the next section I will focus on supervised, unsupervised and reinforcement learning, as semi-supervised learning is just a compromise between supervised and unsupervised.

1. Supervised learning

According to javatpoint supervised learning is a type of machine learning in which machines are trained using well-labelled training data that already has a correct output associated with it, and the machine learning algorithm is used to find the mapping function from input to a given output. Once the mapping function the machine learning algorithm comes up with provides satisfactory answers it can be used to predict the output for new inputs with unknown outputs. Supervised learning can be split into two techniques, Linear regression which is used in predicting and finding relationships between quantitive data, and classification, where an algorithm can process the data into specific groups or “classes” of input.

Some common supervised learning algorithms include 1. Logistic Regression and 2. K-Nearest Neighbours

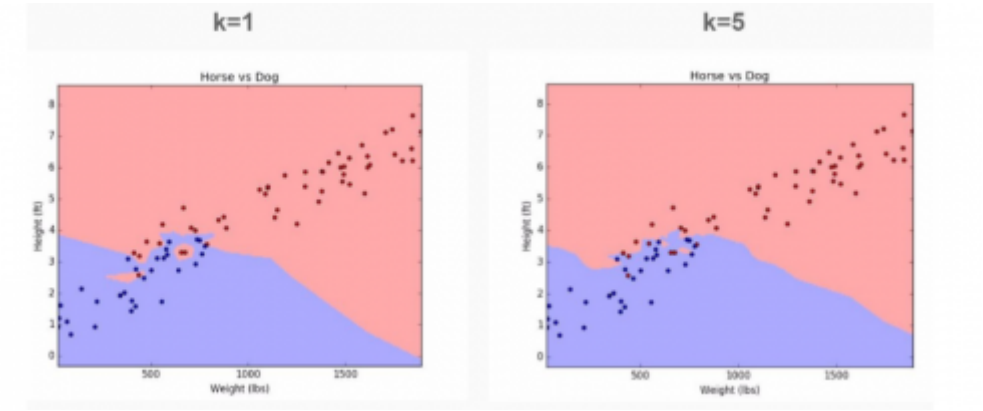
1.1 Logistic Regression

Logistic regression is used to predict the probability of a target variable given an input variable, this type of supervised learning algorithm is a classification algorithm and it means that there are only two possible outcomes for a given input variable. For example, given a credit card transaction the algorithm might determine there is a 15% chance the transaction is fraudulent and 85% that it is not, in this case, there are only two outcomes, fraudulent and not fraudulent. It is one of the simplest machine learning algorithms, and is perfect for use cases where there are only two outputs for a given input (true or false, win or lose, 1 or 0, etc).

1.2 K-Nearest Neighbours

The K-Nearest neigbours is another simpler technique used in machine learning, it is commonly used because of its simplicity and fast calculation time. Let’s assume that our data can be graphed as a point on a graph, given sample input the K-nearest neighbours algorithm will determine which class the input should belong to by looking at its position on the graph, using the assumption that similar data points usually exist close to each other it will classify the input. This algorithm is used by companies such as Netflix or Hulu to suggest movies that are similar to what you have watched before.

The challenge with the K-Nearest Neighbours algorithm is choosing the right value of K, as the value of K is decreased the predictions are model makes become less stable, and if we increase the value of K our predictions become more stable and more likely to make more accurate predictions, eventually, however, the model will create an increasing amount of wrong predictions. This way it is very difficult to tune K to a number that gives us the highest amount of correct predictions. The main disadvantage with K-Nearest Neighbors is that it gets significantly slower as the number of variables increases.



https://datascienceplus.com/k-nearest-neighbors-knn-with-python/

2. Unsupervised Learning

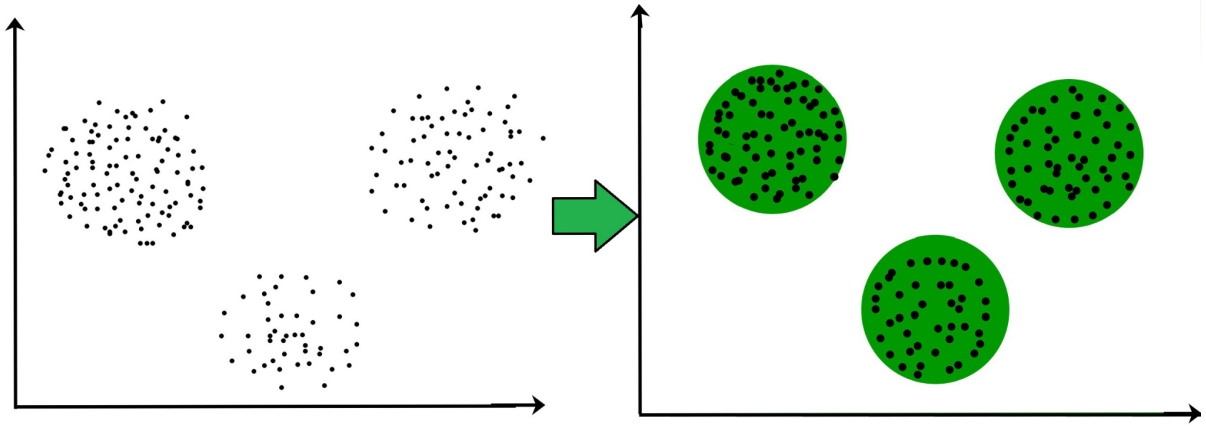
Hewlett Packard Enterprises defines Unsupervised Learning as when a computer is provided with unlabelled data and extracts previously unknown patterns or insights from it. Unsupervised Learning is different to supervised learning as there is only input data provided, and no output data. An algorithm that uses unsupervised learning is Clustering, in which the computer would find all similar data points within a dataset and group them into “clusters”, for example, this can be used to group users of a website into clusters and serve these users similar ads. Below i describe clustering in more detail.

2.1 Clustering

Clustering is the process of dividing data points into a number of clusters or groups so that data points in the same groups are similar and data points in one group are dissimilar to data points in another group. Basically, these groups are a collection of data points that have been grouped based on their similarities.

For example, in the graph shown below, we can see that there are many data points grouped together, we can divide these into groups or clusters as shown. Then given a new data point the algorithm will try to place the new data point into a cluster that has other similar data points.

Clustering is considered to be one of the simplest and most popular unsupervised algorithms



3. Reinforcement learning

“Reinforcement learning is about taking the right action to maximize reward in a particular situation” - geeks for geeks. For example, in its simplest form reinforcement learning can be used to train an A.I. to solve a maze where an action could be moving from one tile to another and the reward would be to solve the maze from start to finish. Given a starting state for the maze, a machine-learning algorithm will choose to make a move and give the next state, if it makes the right move a user may choose to “reward” the model and if it makes a wrong move the user may choose to “punish” the model based on the output state of the maze. In this way, the model will learn from its “mistakes” and begin to be able to solve mazes.

Ethical Concerns

When measuring software engineering we are evaluating the performance of real software engineers, this can, and will pose ethical concerns. When collecting data from the GitHub Rest api a lot of personal information can be found, such as passwords, email addresses and companies people work for. In the sections below i will briefly discuss the ethical concerns in the following headings. Respecting workers autonomy, achieving equity and protecting privacy.

Respecting Workers Autonomy

Meaning that when collecting data about the software engineers we are analyzing that we need to obtain their consent, the workers we collect data from need to understand the purpose, or why we need their data, the risks associated with what we are doing with their data and the consequences if their data give bad results what the consequences will be. Doing this will respect their freedom of choice and allow them to make an informed decision about their data. This is Fine in the context of companies collecting their workers data as the workers can make an informed decision about their data. However in contexts such as my GitHub access there is now way I could get consent from everyone that I got data about. Tools discussed above such as HackyStat can be used on vast amount of data from GitHub without the permission of the owners of the data, this poses ethical concerns regarding people autonomy and free will however we would not be where we are in terms of software engineering without analyzing that data.

Achieving equity

Data could be collected from a single group of people based on factors such as race, ethnicity, country of origin, socioeconomic class, gender and more. People can be harmed from unrepresentative data collected from underrepresented populations, algorithms such as the machine learning algorithms discussed above can be trained on these unrepresentive datasets, this can bee seen in the real world example of the amazon AI recruiting tool that showed a quantitative bias against women.

Protecting Privacy

As discussed above collecting data can potentially pose a wide range of risks to individuals and workers the data is collected about. Is tracking time coding a privacy concern even though these workers are being paid for their time? Is gamifying the job of software engineering by creating metrics used to quantify the performance of software engineers a good idea? When does tracking a software engineer's performance at work become a breach of privacy and can software engineers expect privacy when they are being paid for their time. These questions and more are posed when we collect, quantify and use data to determine statistics about people, and these questions are subjective, it all depends on the context and the individual.

Conclusion

In this report I have explored two of the most common software engineering methodologies, researched and explained some of the observable and measurable metrics we could use to measure the software engineering activity, explored some tools that are available to gather and visualise data relating to software engineering, looked into some techniques that are available for computing data, mainly different types of machine learning and finally i addressed some ethical concerns regarding the gathering, processing and use of this data.

TEMP BIBLIOGRAPHY

Using Data to Make Decisions in Software Engineering: Providing a Method to our Madness →https://www.sciencedirect.com/science/article/pii/B9780124115194000136

The Mythical Man-Month: Essays on Software Engineering - a book by Frederick Brooks

https://code.google.com/archive/p/hackystat/ → hackystat

https://www.usehaystack.io/ → haystack

https://linearb.io/ → linearB

https://codeclimate.com/ → code climate

https://www.geeksforgeeks.org/clustering-in-machine-learning/ → clustering diagram

Machine learning information from hp → https://www.hpe.com/ie/en/what-is/machine-learning.html?jumpid=ps_e4gzhdv13n_aid-520061736&ef_id=CjwKCAiAh_GNBhAHEiwAjOh3ZD7xgdC6B25_zFf58nwOItIq_qCKVAhAkoXkXtVTcs-fGD_YmpjkWRoCwrkQAvD_BwE:G:s&s_kwcid=AL!13472!3!558204158356!e!!g!!types%20of%20machine%20learning!14743716126!132037249627&

Supervised learning →
https://www.javatpoint.com/supervised-machine-learning

K-Nearest Neighbours picture →
https://datascienceplus.com/k-nearest-neighbors-knn-with-python/

Amazon ai →
https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G