*“To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethical concerns surrounding this kind of analytics.”*

**Introduction**

The field of software engineering originates from the 1960s. It arose in response to chronic failures of large software projects to meet schedule and budget constraints. However, it is a young discipline that is constantly evolving at a very fast rate. The rapid development of any field of knowledge brings with it unavoidable fragmentation and proliferation of new disciples. Software engineering being no exception. This phenomenon, as we know it today, is concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them.

In this report, I will discuss the following:

1. The ways in which software engineering can be measured and assessed in terms of measurable data and what data is relevant.
2. Give a brief overview of the computational platforms available to perform this work
3. The algorithmic approaches available to deal with said data.
4. The ethics surrounding this kind of analytics.

I will then proceed to summarize this paper by providing my personal opinion on the matter and form a conclusion.

**Measurable Data**

*“Without data, you’re just another person with an opinion.” – W. Edwards Deming*

Data is a crucial component to consider when analysing the way in which the software engineering process can be measured and assessed. In today’s modern world, there is an infinite amount of data readily available about almost every topic imaginable. It is paramount that this data is measured properly. In this section, I will discuss the need for this measurable data and the issues around data collection, taxonomy and its relevance to measurable data and finally, the various attributes that are necessary to consider when assessing software engineers and their code.

Data is an important, often necessary, part of quality improvement. It can be necessary when identifying or analysing problems or for developing, testing or implementing solutions to those problems. The challenge of collecting software engineering data, specifically, is to make sure that the collected data can provide useful information for the project, process and quality management and at the same time, the data collection process must not be a burden on development teams. Therefore, it is important to consider carefully what data to collect. The data must be based on well-defined metrics and models, which are used to drive improvements. Therefore, the goals of the data collection should be established and the questions of interest should be defined before any data is collected. Data classification schemes to be used and the level of precision must be carefully specified. The collection form or template and data fields should be pretested. The amount of data to be collected and the number of metrics to be used need not be overwhelming. It is more important that the information extracted from the data be focused, accurate, and useful than that it be plentiful. Without being metrics driven, over-collection of data could be wasteful. Over-collection of data is quite common when people start to measure software without an a priori specification of purpose, objectives, profound versus trivial issues, and metrics and models.

If data is not standardized or classified correctly, it can lead to major problems and inconsistent results. A Data Taxonomy is simply a hierarchical structure separating data into specific classes of data based on common characteristics. The taxonomy represents a convenient way to classify data to prove it is unique and without redundancy. This includes both primary and generated data elements.

Taxonomy originates from the Greek dialect. “Taxis” meaning arrangement of division and “nomos” meaning law. It is the science of classification according to a pre-determined system with the resulting catalogue used to provide a conceptual framework for discussion, analysis or information retrieval. To make search and browse capabilities of content, documents or record management systems truly functional, we need to develop taxonomies. Taxonomy also improves performance of data collection and analysis.

Master Data Management is an application of taxonomy. It comprises of a set of processes and tools that consistently defines and manages the non-transactional data entities of an organization. MDM promises not just greater control over consistent reference data; but an ability to manage the relations between data entities in order to generate more effective business knowledge. From this perspective, MDM requires an understanding and agreement about the meaning of terminology.

Another aspect of data in question is that which relates to the software engineer in question. There are approximately six different appropriate headings under which one can review the data of a software engineers’ code. Those being:

1. Number of commits
2. Source lines of code (SLOC)
3. Lead time
4. Technical debt of said data
5. Bug fixing and handling systems
6. Code churning
7. Whether or not the project was conducted by a team or an individual.

Each one of these aspects of the data will alter its value of its relevance to the measurement and assessment of the software engineering process.

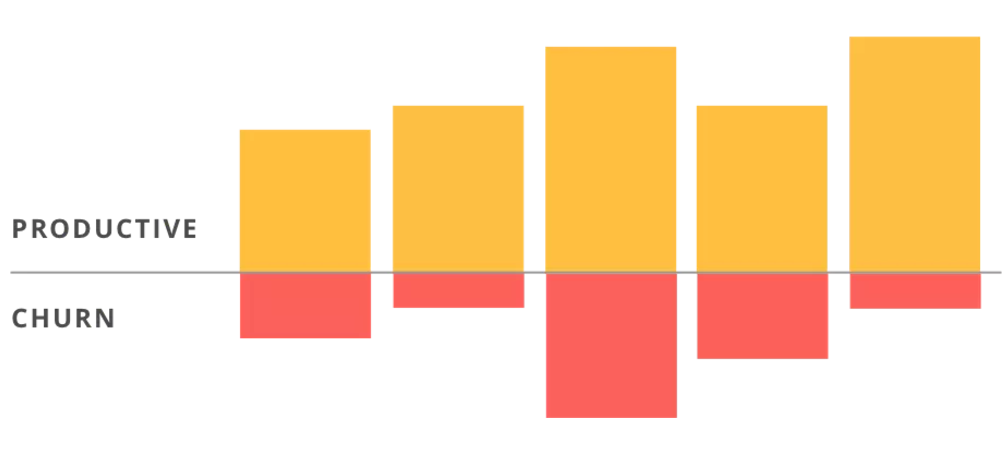
**Number of commits** refers to the amount of times a software engineer contributes to their code. It can vary greatly. This does not specifically distinguish a good software engineer from a poor one as it is merely a method of the amount of times one made significant changes to code. It does, however, provide us with a very good idea of which software engineers are active, when they are most active and effectively, who is doing the most work. This sort of data is very important for managing directors of companies when it comes to promotions, redundancies and progress reviews.

**Source lines of code** is a software metric used to measure the size of a software program by counting the number of lines in the text of the program’s source code. It is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced. There are two major types of SLOC measures: physical and logical. Physical is a count of lines in the text of the program’s source code including comment lines. Logical attempts to measure the number of “statements”. It is much easier to create tools that measure physical SLOC. SLOC can be strongly correlated with estimating effort. However, functionality not as much. It is a poor productivity measure of individuals, since a developer can develop only a few lines and yet be far more productive in terms of functionality than a developer who ends up creating more lines. It is considered a particularly ineffective method to use when assessing data.

**Lead time** refers to the time elapsed between identification of a requirement and its fulfillment. Lead time can have a serious impact of the quality of data when using it to measure and assess the software engineering process. It can provide a huge amount of invaluable data.

**Technical Debt** refers to a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution instead of using a better approach that would have taken longer. It is another very important factor to consider in the world of software metrics. It can be created by an incorrect data model, low quality data and no hypothesis testing. It is something that all software engineers strive to avoid.

**Bug fixing and bug handling** does not specifically refer to the amount of bugs in ones code as there will always be some present, rather how much time a software engineer is spending on them each week. This includes both fixing issues once they have been identified or troubleshooting issues when they arise. According to Vlad Givetrts, head of Engineering for Clara Lending, if dealing with bugs takes more than 20% of a software engineers engineering time, there is a definite problem with either quality of code or the architecture system in place.

[**Code Churn**](https://blog.gitprime.com/why-code-churn-matters)is the percentage of a developer’s own code representing an edit to their own recent work. It’s typically measured as lines of code (LOC) that were modified, added and deleted over a short period of time such as a few weeks. The primary purpose of measuring churn is to allow software managers and other project stakeholders to control the software development process, especially its quality. When churn starts to spike, this can be an indicator that something is off with the development process. For example, imagine a situation where a developer receives a very opaque set of requirements, like “the app needs settings” — something that’s not uncommon when working with product stakeholders. When this disconnect turns into weeks worth of iteration on the same feature without a lot of forward progress, that will show up as code churn. Churn rate can also help to identify problems with individual developers. For example, a sudden increase in churn rate may indicate that a developer is experiencing difficulty in solving a particular problem or is repeatedly polishing a feature that’s ready for release. A high churn rate may also mean that a developer is under-engaged. Other causes of high churn include an indecisive product team that has the developer running in circles.

Finally, it is important to consider whether the project has been conducted by an individual or by a team as this will alter the expectations of each of the attributes above.

With regards measuring a software engineer from a non-code perspective the following headings are relevant:

1. Communications
2. Emails
3. Meetings attended
4. Pull requests

While these are almost as important to the evaluation of a software engineer, they can be viewed as intrusive and a company should ensure that they have the permission of each and every software engineer before gathering this data.

In conclusion, there is a lot amount of data to be processed and analyzed. There are a huge amount of other methods available but as the modern world of technology progresses, such methods are veering towards sonar and sixth sense. However, these modern approaches could assuage some developers’ fears regarding behavioral data collection and analysis. It is a very broad topic which has huge growth potential but must be developed with caution.

**Where to Compute?**

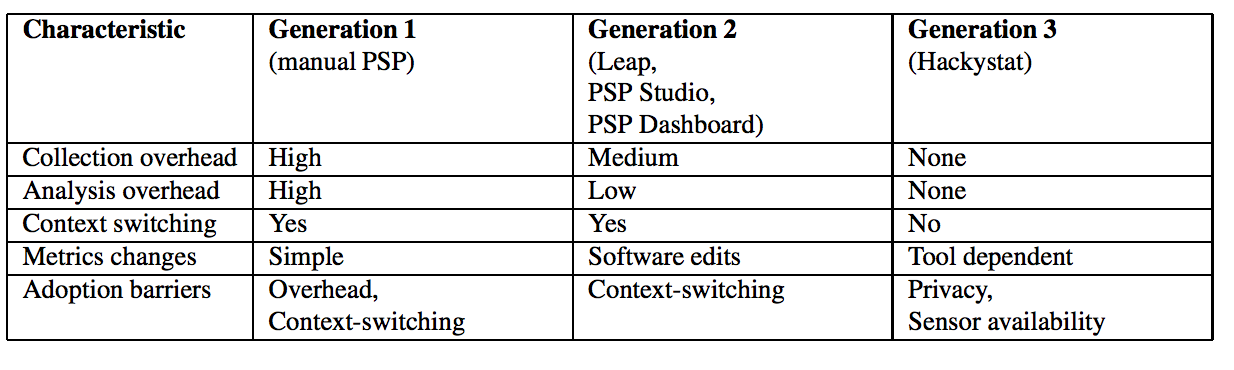
Obtaining all of this measurable data is all well and good provided there exists competent platforms to compute such data for us. The field of computing measurable data has undergone extreme developments from beginning as something that had to be calculated by hand to evolving into becoming automated. In this section of this report, I will strive to explain the historical advancement of automating the computational platforms, I will discuss the most popular methods in use today and finally, I will bring to light some of the companies who have achieved huge success by providing these platforms and analytics as a service.

In 1995, Watts Humphrey authored A Discipline for Software Engineering, a ground-breaking text that adapted organizational-level software measurement and analysis techniques to the individual developer along with a one semester curriculum. These techniques are called the Personal Software Process (PSP). Such techniques aimed to automate the collection of human data which would, in turn, eliminate human error and human effort.

Humphrey’s version of the PSP uses simple spreadsheets, manual data collection, and manual analysis. Collecting and managing this data takes substantial effort. Interestingly, Humphrey actively embraced the manual nature of the PSP: “It would be nice to have a tool to automatically gather the PSP data. Because judgement is involved in most personal process data, no such tool exists or is likely in the near future”. More fundamentally, Humphrey viewed his predefined PSP processes as a bootstrapping method. In the book, he exhorts developers to modify the forms and procedures he presents to address specific circumstances and needs.

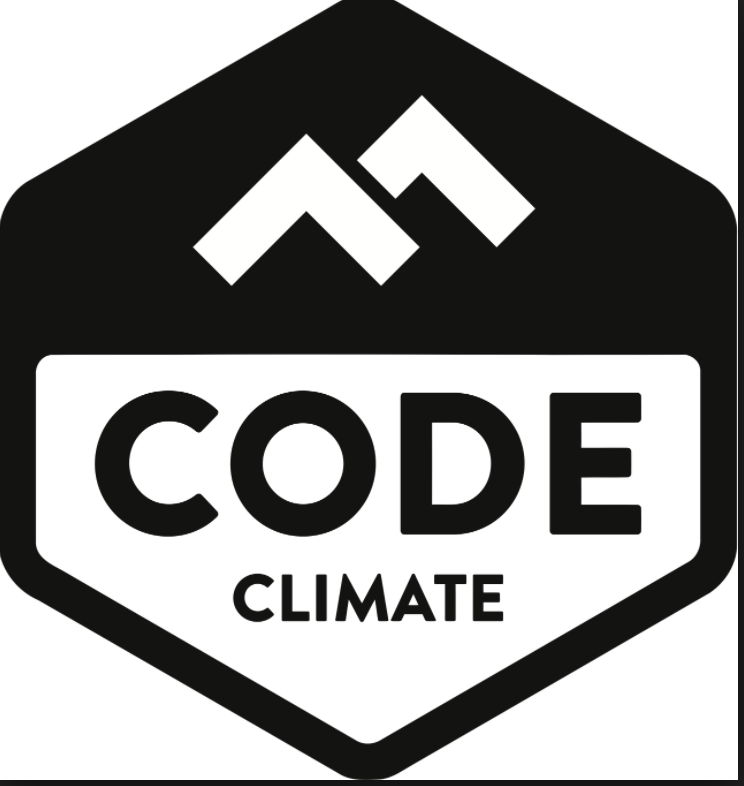
However, after substantial research was conducted into this process, it was concluded that the manual nature created problems for significant data quality problems. To address this problem, Leap toolkit was developed. Leap stands for Lightweight, Empirical, Anti- measurement dysfunction and portable software process measurement. It attempts to overcome the data quality problems associated with PSP by automating and normalizing data analysis. Although the developer still manually enters most data, the toolkit automates subsequent PSP analyses and, in some cases, provides analyses (such as various forms of regression) that the PSP doesn’t provide. It attempts to avoid measurement dysfunction by enabling developers to control their data files. It maintains data about only the individual developer’s activities and doesn’t reference developers’ names in the data files. Leap data is also portable. It creates a repository of personal process data that developers can keep with them as they move from project to project and organization to organization. By introducing automation, the Leap toolkit makes certain analytics easy to collect but others increasingly difficult. All data collected can then be analyzed by Hackystat, a data collection tool which was developed at the University of Hawaii.

Hackystat implements a service oriented architecture in which sensors attached to development tools gather process and product data and send it to a server, which other services can query to build higher-level analyses. Hackystat includes four important design features. The first is both client and server-side data collection. Modern software development typically includes individual developers’ activities on their local workstation as well as server- or cloud-based activities. The second feature is unobtrusive data collection. For developers, one of the most frustrating aspects of manual data collection is the loop of doing some work and then interrupting it to record what they worked on. An important requirement for Hackystat was to make data collection as unobtrusive as possible. Users shouldn’t notice that data is being collected, and the system shouldn’t make assumptions about network availability. The third feature is fine-grained data collection. By instrumenting client-side tools, Hackystat can collect data on a minute-by-minute or even second-by second basis. For example, Hackystat supports a measurement called buffer transition—collecting a data instance each time the developer changes the active buffer from one file to another. Hackystat can also track a developer as he/she edits a method, constructs a test case for that method, and invokes the test, yielding insight into real-world test-driven development. The fourth feature is both personal and group-based development. Besides collecting their personal development data, developers can define projects and shared artifacts to represent group work. Hackystat can track the interplay among developers when, for example, they edit the same file.

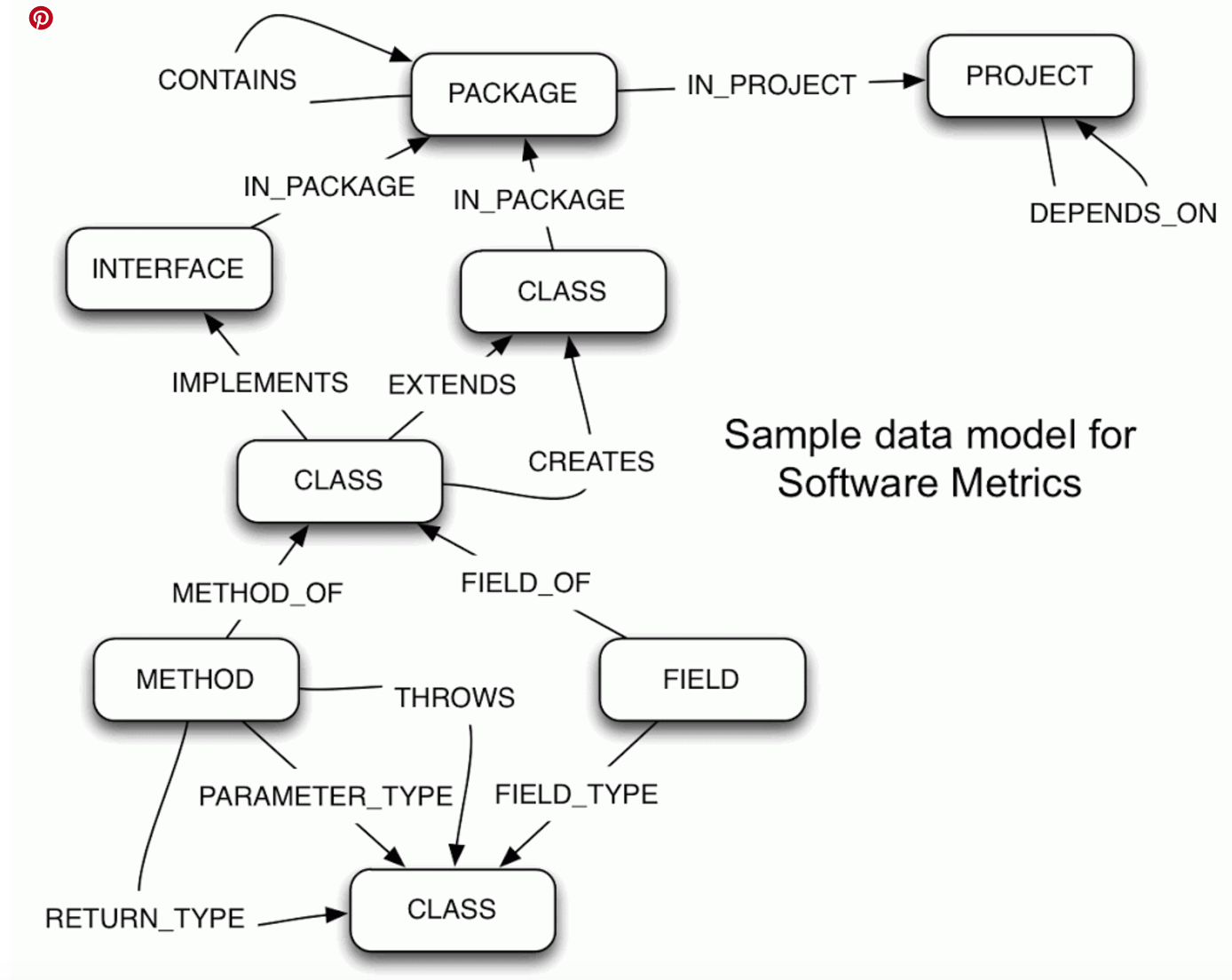


The field of measuring software engineering has effectively started a whole new market. One which consists of companies who provide different platforms with various different features which allow software engineers to analyse all relevant data in a user-friendly, graphical way. Some of the most well-known companies that supply software which collects and analyses measurable data are as follows: Code Climate, Gitcop, Gitcolony, Codebeat, Semmle, Teamscale, Black Duck, Codebrag and Phabricator

Code Climate is an example of one of these platforms which I will discuss in detail as it is the one that I concluded to be most appropriate and efficient in measuring and assessing the data of a software engineer. This computational platform is used by over 100,000 projects. They analyse on average 2 billion lines of code daily. Code Climate incorporate fully- configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous and ubiquitous. Some features provided by this platform are:

* Automated Git Updates- Nothing to install. Code Climate runs every time a new commit is pushed.
* Activity Feeds- Up-to-the-minute information so a company can see when and how code changes.
* Instant Notifications- Major security and quality changes pushed to where employees work: email, Campfire, HipChat, and RSS feeds.
* Team Sharing- Instant access for a whole team to maximize code visibility across projects.
* Duplication Detection- Fuzzy matching algorithm finds DRY-violations that human reviewers might miss.
* Email Notification- Instant email notifications to let a company can know when new security and code issues arise
* Security Dashboard

Codebeat is a second data analytics company that I will discuss in greater depth so as to highlight the necessity and role of computational companies in today’s world. It is a simple, free, open-source code review tool. Codebeat is a dynamically growing tool that covers major technologies and programming languages. It has evolved substantially within the last few months. It supports most of the languages, has metrics customization, uses measuring tools with own algorithms, provides very good support system from the Codebeat team and has a smart but well-documented API, which facilitates management.

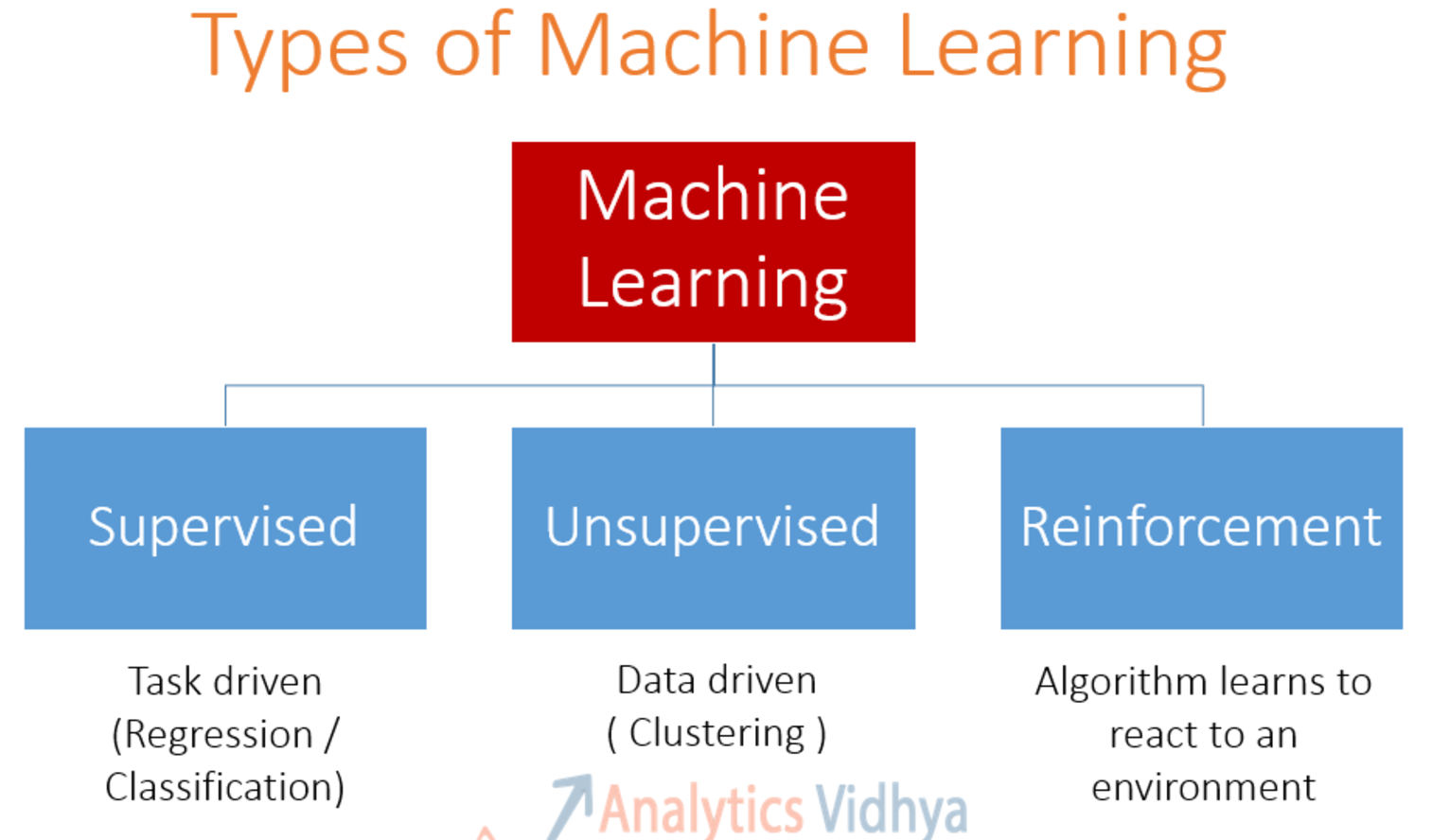
An example of a graphing feature available to display this huge amount of data is Neo4j. Neo4j is a database which stores, computes and presents visualizations of a variety of software metrics and other types of software analytics such as method call hierarchies, transitive clojure, critical path analysis, volatility and code quality. The graph structure that would accommodate the information quite well would be a direct representation of the concepts in the software projects, consisting of projects, packages, classes, interfaces, types, methods, fields and containing relationships like dependencies, usage, creation, containment, calls, coverage, etc.

**What Algorithms?**

According to a survey carried out by the Standish Group, an average software project exceeded its budget by 90% and its schedule by 222%. This survey took place in mid 90s and contained data from about 8-000 projects. These statistics show the importance of measuring the software early in its life cycle and taking the necessary precautions before these results come out. For the software projects carried out in the industry, an extensive metrics program is usually seen unnecessary and the practitioners start to stress on a metrics program when things are bad or when there is a need to satisfy some external assessment body.

Identifying and locating defects in software projects is difficult work. Especially, when project sizes grow, this task becomes expensive with sophisticated testing and evaluation mechanisms. On the other hand, measuring software in a continuous and disciplined manner brings many advantages such as accurate estimation of project costs and schedules, and improving product and process qualities. Detailed analysis of software metric data also gives significant clues about the locations of possible defects in a programming code. This is the third element of my Measuring Engineering report. I will discuss the process of measuring software engineering paying heed to the relevant algorithms that are available which can analyze the data we have collected. I will mainly discuss machine learning processes and how it aims to provide techniques that improve the state of data. Machine learning has the advantage of being unbiased, whereas experts instinctively use their intuition and expertise, which may be biased.

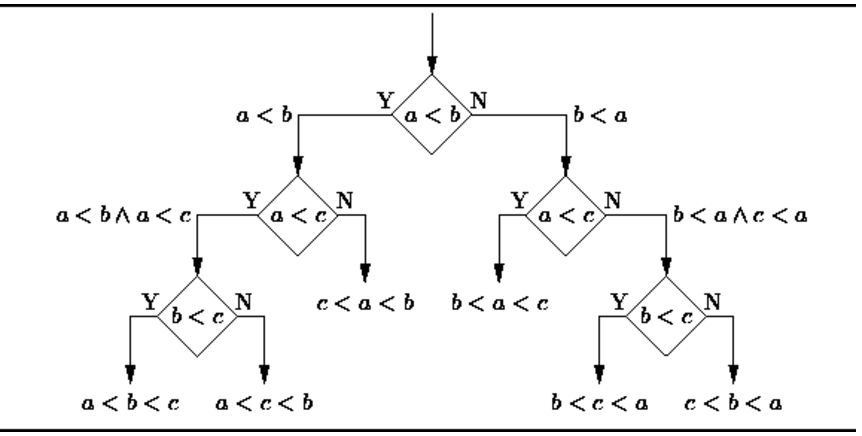
Machine learning algorithms can be divided into three broad categories: supervised learning, unsupervised learning and reinforcement learning.



**Supervised learning**

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output. The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

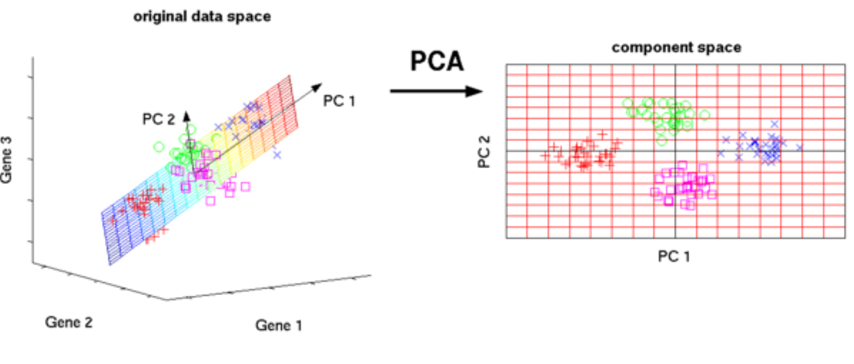
It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answer, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance. Supervised learning problems can be further grouped into regression and classification problems. Examples of supervised learning algorithms are linear regression, decision tree analysis, KNN and logistic regression.

**Decision Tree Analysis:**A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance-event outcomes, resource costs, and utility. From a business decision point of view, a decision tree is the minimum number of yes/no questions that one has to ask, to assess the probability of making a correct decision, most of the time. As a method, it allows you to approach the problem in a structured and systematic way to arrive at a logical conclusion.

**Unsupervised Learning**

Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data. Unsupervised learning problems can be further grouped into clustering and association problems. Examples of unsupervised learning algorithms include clustering, k-means and PCA.

**Principal Component Analysis**: PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Some of the applications of PCA include compression, simplifying data for easier learning, visualization. Notice that domain knowledge is very important while choosing whether to go forward with PCA or not. It is not suitable in cases where data is noisy (all the components of PCA have quite a high variance).



**Reinforcement Learning**

Reinforcement learning it the third type of machine learning. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as its reinforcement signal. This behaviour can be learnt once and for all, or keep on adapting as time goes by. If the problem is modelled with care, some Reinforcement Learning algorithms can converge to the global optimum; this is the ideal behaviour that maximises the reward.

**T**his automated learning scheme implies that there is little need for a human expert who knows about the domain of application. Much less time will be spent designing a solution and all that is required is someone familiar with Reinforcement Learning. Examples of reinforcement machine learning algorithms are Markov decision process, Dynamic Programming and Monte Carlo Methods.

**Markov Decision Process** provide a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under control of a decision maker. The algorithm is placed in a certain state/ condition and from that be given certain actions it can take with consequences for each. It must make decisions and define the new state it is in. This repeats until the goal objective has been realized. Each time the algorithm makes the wrong decision, it learns from said decision and not repeat it. A good example of this algorithm in practice is a maze. The system moves through the maze systematically learning and recognizing how to get out of it as quick as possible.

While all of these models solve various problems, which make the users life exponentially easier, it must be mentioned that they all contain an element of doubt. Each one makes unique assumptions and it might not be obvious how to navigate and identify which assumptions are reasonable and which are incorrect. Thus, resulting in false positive/ negatives. The effort required to track down these incorrect results can be immense. According to a Barkly survey of IT administrators, 42% of companies believe that their users loose productivity as a result of false positive results.

This leads us to wonder how accurately these techniques can actually address the kinds of questions that people have about software engineering? In my opinion, while they do sometimes produce false results, the advantages associated with correct results immensely outweigh the disadvantages. The machine learning techniques save huge amounts of time and effort. They are also usually a lot more accurate as they eliminate human error. One should practice caution whilst developing such algorithms. I think it is an area that will undergo the most dramatic changes in the forthcoming years. With the rapid increasing improvements in AI and computational intelligence platforms, it is only a matter of time before almost everything used in the modern world is automated.

**Ethics**

The final aspect of this analysis is, naturally, to ask the question:

Is all of this ethical?

Ethics relate to the moral principles that govern a person’s behaviour or the conducting of an activity. The ethics associated with software engineering are very strict. Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems. Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.

Data sovereignty is the concept that information that has been converted and stored in binary digital form is subject to the laws of the country in which it is located. It can sometimes be ambiguous what specific law governs where. For example, in Australia, if the cloud server center of a company is located in Australia, then your data is safe and secure within Australian legislation. However, if your company is international but has a presence in Australia, all relevant data may be subject to be stored elsewhere. As Brexit becomes a reality, the discussions surrounding this topic have often failed to provide the clarity needed. The onward result of this being that businesses themselves end up struggling to understand the often-incongruous requirements, laws and regulations that exist.

In my personal opinion, I believe that the measurement of software metrics is neither ethical nor unethical. It is a grey area. With the development of AI and the alterations that it will bring to the human race as we know it, I believe that we must tread with caution. Provided a company is analyzing relevant data such as productivity, effort or performance, then it is ethical. For example, such data may highlight a particular employee who is not working hard enough or is not suitable for the job and may be subject to dismissal. The company can then utilize this information appropriately. Another interesting example of useful information ‘Big Data’ has revealed, is that contrary to the popular myth that employees need a strong work-life balance to be engaged. It has been discovered that engagement levels rise when employees are mission- driven which would lead to increased work hours and reduced personal time. Thanks to data analytics, organizations are in a stronger position to study potential candidates and pinpoint with amazing accuracy those who have the capability to do the job, the capacity to learn new skills that may be needed in the future and who are a good match with the culture of the company.

However, this is unfortunately, not always the case. It is becoming increasingly common for companies to analyze very personal, private data. Data which should not be known to the public. This allows companies to make predictions as to how this employees will perform in the future. A particular example of this that I came across during my research for this report was in health care. Health care analytics companies can now mine workers’ medical claims, pharmacy claims and search queries to figure out if an employee is trying to conceive, if they are already pregnant, if they have diabetes or even if they may need back surgery. They can sometimes know these things before the employees themselves. This information is then sold onto other companies such as Target or Walmart who use it to decide on promotions etc.. While this appears outrageously unethical, it is in fact legal. While there are some laws in place, such as The Health Insurance Portability and Accountability Act of 1996, to try and prevent exactly this from happening. There are no laws which govern the tracking of search queries and insurance claims. This undoubtedly blurs the lines between ethical data measurement and unethical data measurement. I think that this sort of data collection is a violation of employee privacy. It is providing answers to questions that should never have been asked in the first place. It questions the fidelity of software measurements as a field.

In order to overcome these problems, one should strive to remove the concept of data security from being a grey area to that of black and white. Boundaries, laws and legislation should be put in place.

**Conclusion**

To conclude, I believe that the measurement of software engineering as a process is one that brings with it a huge amount of questions. Questions such as what data is the best to measure? How should one collect such data? Where should one store this data? How one can study this data? And, finally, is the collection and examination of this data ethical? I believe I have addressed these questions sufficiently.

The aim of this report as a whole was to discuss and analyze the process of measuring software engineering. I have done so under the headings of measureable data, computational platforms, algorithmic approaches and ethics. I believe that they give a well- rounded view of the field. One should approach this area with caution. As aforementioned, it is one surrounded by a huge number of questions. Such questions blur the lines between ethical and unethical, between right and wrong. The development of this field is one that has undergone rapid expansion in the past decade. ‘Artificial Intelligence’ and ‘Machine Learning’ are buzz words that are becoming increasingly common today. However, it appears that few genuinely understand the true meaning of them and the potential they have to alter our world irreversibly. And, while the principal component of such change is for the good, there is inevitably an element of harm. It is alarming, the extent to which humans can be tracked and analyzed in the modern world today. And, in my opinion, the responsible implementation of this power is up to us.

**Bibliography**

1. <https://books.google.ie/books?hl=en&lr=&id=lx_OBQAAQBAJ&oi=fnd&pg=PP1&dq=data+security+in+software+metrics&ots=_UhSRkXL-z&sig=lJt6p5jKiRsOabzwSfrAZ0i8RB4&redir_esc=y#v=onepage&q=data%20security%20in%20software%20metrics&f=false>
2. <https://dl.acm.org/citation.cfm?id=559784>
3. <http://fowler.ucsd.edu/modeling_dynamical_influence.pdf>
4. <https://www.compact.nl/en/articles/data-quality-assessment/>
5. <https://spacetimeinsight.com/five-aspects-of-data-quality/>
6. <https://leankit.com/blog/2015/11/lead-time-metrics-why-weekends-matter/>
7. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf>
8. <https://ac.els-cdn.com/S1071581984710731/1-s2.0-S1071581984710731-main.pdf?_tid=61a76b52-d057-11e7-8192-00000aacb35d&acdnat=1511446118_673f20afbc7e650524f00c81b781d87b>
9. <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1231&context=sei>
10. <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1201249>
11. <http://seari.mit.edu/documents/theses/PHD_GROGAN.pdf>
12. <https://martinfowler.com/articles/useOfMetrics.html>
13. <https://stackify.com/track-software-metrics/>
14. <http://www.whaqualitycenter.org/Portals/0/Tools%20to%20Use/Collecting%20Data%20and%20Information/Data%20Collection_%20When%20is%20it%20Needed%20R%202-12.pdf>
15. <https://www.fastcompany.com/3014837/with-big-data-companies-can-predict-your-success-before-yo>
16. <https://www.weforum.org/agenda/2016/10/top-10-ethical-issues-in-artificial-intelligence/>
17. <http://ieeexplore.ieee.org/document/1201249/>
18. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf