# Software Engineering – Measurement & Assessment

"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 ethics concerns surrounding this kind of analytics."

## 1. Introduction

This report aims to deal with the way in which software engineering process can be measured and assessed. It will focus on the areas of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethics concerned with this kind of analytics.

## 2. Software Engineering Process

To begin understanding its measurement and assessment we must first fully understand the software engineering process itself. The concept of software engineering was first proposed in the public sphere in 1968 at the NATO Software Engineering Conference as a solution to the then 'Software Crisis'. The crisis being that individual approaches to development did not scale up for larger systems.

"*Our civilisation runs on software*," Bjarne Stroustrup, the creator of the C++ programming language.

Using a basic definition, we can classify software as computer programs and their associated documentation. Programs consist of algorithms which are applied to data. The above quote is hard to dispute when you consider the amount we interact with software during the course of our daily lives. It controls all of the devices that we are dependent on such as mobile phones, laptops, smartwatches and many more. It can prove difficult to think of even a few things we normally do in which software isn't in some way involved.

Software engineering is defined as the applying of the principals of engineering to the software development process. The main difference between it and other branches of engineering is that you are not building a tangible structure. However, what you are developing is used in actual machinery and devices so its effects are very much tangible and of serious importance. Its aims are to break the process down into several different components to improve design, product management and project management.

# 3. Measurable Data

"Provide useful information for project, process and quality management and, at the same time, that the data collection process will not be a burden on development teams."

The above quote illustrates the challenges facing software engineers when collecting data on their processes. They must carefully consider what kinds of data will generate a beneficial effect from being collected. The goals and interests of the data collection must be established at the beginning of the process. The metrics and models that the data are based on are used to drive improvements before any data collection even begins. The most important factor is that the data collected should be focused, accurate and useful rather than a focus on sheer quantity. Without focusing on usefulness, you could end up with a large amount of data that was expensive and time consuming to gather. Even if this takes place in a large company who can take advantage of economies of scale to lower costs, it is still an unnecessary expense to avoid.

In terms of the working environment in a business, the most important measurable variables will be time and money. Furthermore, even these two are heavily connected as time is tantamount to money. These are by far and away the two greatest constraints faced by any business in the development of any software. Due to their significance they ought to be accounted for in any measurement taking place.

There is a large quantity of different techniques to collecting data on the process of software engineering. One should consider what kind of data they are collecting when choosing what particular technique to use. There are various different advantages and disadvantages to each technique. For example, a questionnaire is far cheaper than an interview however a vague question can lead to poor data quality which could be avoided with an interviewer clarifying a question. To the right is a table of various data collection techniques.

| Category | Technique |
|---|---|
| First Degree | Inquisitive techniques |
| | • Brainstorming and focus groups |
| | • Interviews |
| | • Questionnaires |
| | • Conceptual Modelling |
| | Observational Techniques |
| | • Work Diaries |
| | • Think-Aloud Protocols |
| | • Shadowing & Observation Synchronised Shadowing |
| | • Participant Observation |
| Second Degree | • Instrumenting Systems |
| | • Fly on the Wall |
| | • Analysis of Electronic Databases of Work Performed |
| | • Analysis of Tool Use Logs |
| | • Documentation Analysis |
| | • Static and Dynamic Analysis of a system |

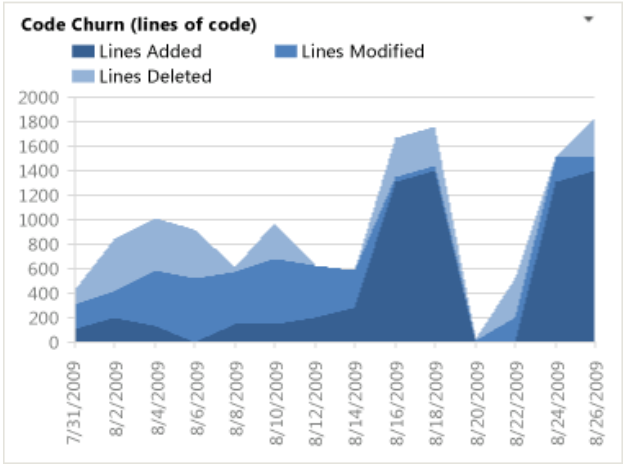
When collecting data, we must be careful to avoid a powerful form of observational bias known as the streetlight effect. This occurs when you focus on easily obtained metrics with little or no impact. Typically, the easier and less controversial a metric was to obtain the less useful it will prove. The best example is analysis of the data in a configuration

management repository. This data is easily accessed and due to its public nature, developers usually have no issue with its use. However, it only give us a very narrow glimpse at the developmental process.

There is no standard definition of metrics that have value to a software development team. Every team will have differing needs that sets them apart from others. What needs to be collected and will prove useful to a team will vary depending on their goals. Here are a few examples of metrics than can help developers:

**Lead Time:** Lead time quantifies how long it takes an idea to be fully developed and delivered as software. This can be a very valuable piece of information as it helps focus companies on how responsive they are to customers. For obvious reasons, a company who can adapt quickly to a customer's needs will find itself in an advantageous position.

**Code Churn:** Code churn is the number of lines of code that were modified, added or deleted in a specified period of time. When churn spikes, it can be an indicator of an issue that requires attention. For example, if a manager noticed that one of their developers had a much higher rate than their teammates than it would indicate that this developer is struggling far more than their counterparts. It can also be used to identify issues plaguing the entire team. For example, if a team were to receive very vague directions from a client about some aspect of a project, this would be marked with a large increase in code churn as the team continue to work with little progress.




**Bugs:** This does not just refer simply to how many bugs you find as they will always be present, and the number will vary a lot depending on the project at hand. What is important is the percentage of your time you're spending addressing these bugs. This includes both fixing issues when you've identified them and troubleshooting them when they come up. According to Vlad Giverts, head of engineering for Clara Lending, if you're spending more than 20% of your time dealing with bugs than you probably have a problem with your architecture/design that is draining you resources.

**Efficiency:** Efficiency attempts to measure the amount of useful code generated by a developer. The code churn reveals the amount of ineffective code written, so a developer with a low churn rate is therefore producing more effective code.

**Source lines of code:** This is a very simple measure used to measure the size of the of a program written by a developer by simply counting the number of lines of code written. There are two measurements, physical and logical. Physical merely counts all lines including comments whereas logical attempts to count only statements. The physical measurement is far easier to record. SLOC can be useful when attempting to measure effort however is a very poor measurement of effectiveness. One developer could write far fewer lines than their counterpart but have produced a far more effective solution to a problem. For this reason SLOC is largely regarded as a very poor metric to use to gain any insight.

## 4. Computational Platforms

Once firms have decided on which metrics to use and collected the relevant data, they must decide how and where they will compute it to obtain meaningful results. The field of computing measurable data has undergone dramatic developments from its inception. It has progressed from intensive, manual calculations to a wide market of firms offering automated analytics programs. Historically, metrics were calculated using the Personal Software Process (PSP) and later by automated analytics programs.

The PSP provides developers a disciplined personal framework for developing software. It consists of a set of methods, forms and scripts that allow them to plan, measure and manage their work. It was first described in Watts Humphrey's book *A Discipline for Software Engineering.* Humphrey believed that following a structured development process and constantly tracking their progress would allow a developer to better understand and improve their output. In his book, the version of the PSP uses simple spreadsheets, manual data collection, and manual analysis. This collection and management of data takes a large degree of effort. In one version of the PSP, developers must fill out 12 forms that contained more than 500 distinct values that must be manually calculated. It was noted that 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."

Name: Jill Fonson — Program: Analyze.java

| Date | No. | Type | Inject | Remove | Fix time | Fix defect no. | Description |
|---|---|---|---|---|---|---|---|
| 9/2 | 1 | 50 | Code | Com | 1 | 1 | Forgot import |
| 9/3 | 2 | 20 | Code | Com | 1 | 2 | Forgot ; |
| 9/3 | 3 | 80 | Code | Com | 1 | 3 | Void in constructor |

There can be data quality issues with PSP originating from its heavily manual nature. From this a toolkit called LEAP was created to help fix some of these issues. The data still had to be manually inputted by a user however it offered automation of the PSP analysis and extending the degree of analysis by introducing more complicated regression models. LEAP also enabled developers to not publish their name, so they could keep their privacy. It creates a portable repository of personal process data which you can bring from project to project. This kind of analysis is extremely effective in improving individual performance.

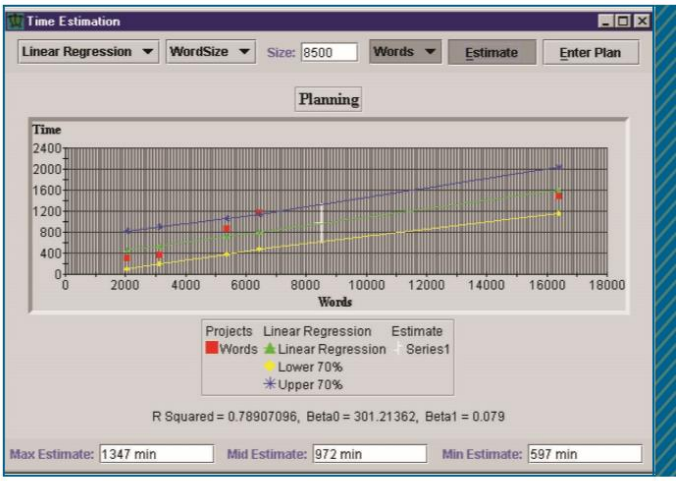
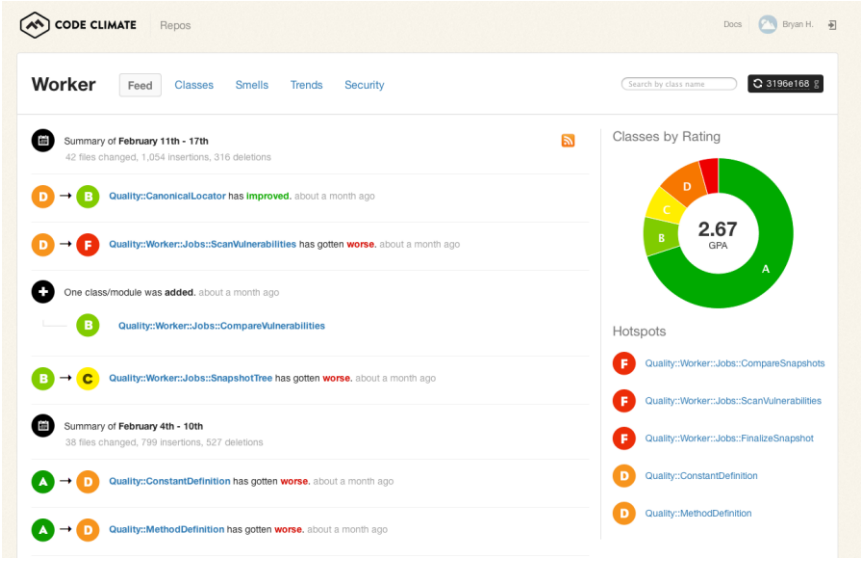



**FIGURE 2.** The time estimation component in the Leap (*l*ightweight, *e*mpirical, *a*ntimeasurement dysfunction, and *p*ortable software process measurement) toolkit. Unlike the PSP, no Leap analytics are paper-based.

The next major development was the production of the Hackystat project which 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 analysis. It includes four main design features. The first is both client and server-side data collection. This typically means recording individual developers' activities on their local workstation as well as server- or cloud-based activities. The second feature is unobtrusive data collection. One of the most frustrating aspects of manual data collection for developers is having to do some work and then being interrupted to have to record it. Ideally users shouldn't notice that data is being collected, and the system shouldn't be affected by network availability. The third feature is fine-grained data collection which means it can collect data on a second-by second basis. Hackystat supports a measurement called buffer transition which collects a data instance every time the developer switches the active buffer from one file to another. It can also track a developer as they edit a method, constructs a test case for that method, and invokes the test, which yields insight into real-world test-driven development. The fourth feature is both personal and group-based development. As well as collecting their personal development data, developers can define projects and shared artifacts which represent group work. Hackystat can track the interplay among developers when, for example, they edit the same file.

The field of measuring software development has now grown into a huge market of companies that provide platforms to developers with greatly varying features which allow them to easily analyse and display their performance graphically. Some examples of these companies we will look at in more detail are Code Climate and Codebeat.

Code Climate is an automated code review tool. It takes in code and assesses it on a number of different levels then makes recommendations of what can be improved. They offer a service in conjunction with Github which can show a user their code coverage, technical debt and progress report amongst many other features. It can support many languages, technologies and frameworks as well as being renowned for its reliability, stability and test coverage. Code Climate provides a user friendly, intuitive way of accessing and viewing the types of data discussed earlier. However, it is expensive when compared to its rivals.



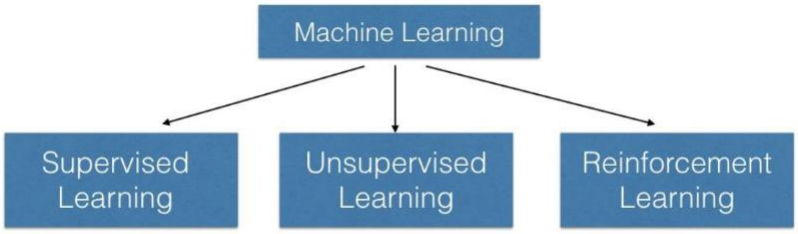

Codebeat is a simple, free, open-source code review tool. It is a dynamically growing tool that covers multiple major technologies and programming languages. It has substantially evolved recently currently possessing metrics customization, measuring tools with their own algorithms, great team management functionality, excellent support system from the Codebeat team and has a smart but well-documented API, which facilitates management. It is used by large software development companies such as jtribe and Perk.

| **Characteristic** | **Generation 1** (manual PSP) | **Generation 2** (Leap, PSP Studio, PSP Dashboard) | **Generation 3** (Hackystat) |
|---|---|---|---|
| Collection overhead | High | Medium | None |
| Analysis overhead | High | Low | None |
| Context switching | Yes | Yes | No |
| Metrics changes | Simple | Software edits | Tool dependent |
| Adoption barriers | Overhead, Context-switching | Context-switching | Privacy, Sensor availability |

## 5. Algorithmic Approaches

Over the last few hundred years most of our growth can be attributed to technological innovations. Identifying and locating defects in software projects is difficult work. Especially, when projects are of enormous size, this task becomes very expensive with sophisticated testing and evaluation mechanisms required. But it is crucial to measure software in a continuous and disciplined manner as it provides so many advantages such as accurate estimation of project costs and schedules and the improvement of product and process qualities. Detailed analysis of software metric data also gives significant clues about the

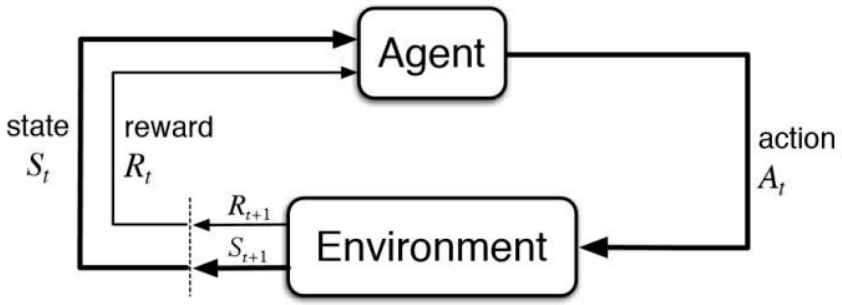
locations of possible defects in a programming code. This section focuses on the relevant algorithms used in this measurement and analysis, mainly focussing on machine learning. Machine learning has a significant advantage of being unbiased, whereas humans even experts instinctively use their intuition and expertise, which may be biased.



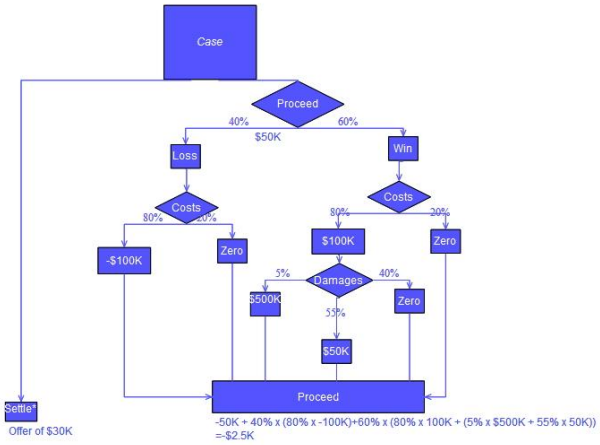

Supervised machine learning is the most popular type. This occurs when you have and input and output variables such y = f(x). The goal is to be able to predict the output variables using input variables not includes in the original set. It does this by constantly minimising the sum of square errors. It earns its name from the process of an algorithm learning from the initial dataset. The algorithm iteratively makes predictions on the training data and is corrected which it learns from. This is completed when the algorithm achieves an acceptable level of performance. There is a constant search for the minimum of the sum of square errors which is used to generate the most accurate results.

Unsupervised machine learning also contains a set of input variables but lacks the set of output variables. It gets its name from the fact there are no correct answers and no way of correcting its output. Its output is intended to give us groupings of the data by clustering and association which attempts to help us understand the data. It does this by establishing relationships between the data points.

Reinforcement learning makes use of dynamic programming to allow machines to automatically determine the ideal behavior in a specific context. For example, when presented with a choice, a or b, if the chosen option leads to a negative result, the machine will return to this point and choose the other option until it receives a positive result. It repeats this process, learning from its previous choices. Simple reward feedback is all that is required for the machine to learn.

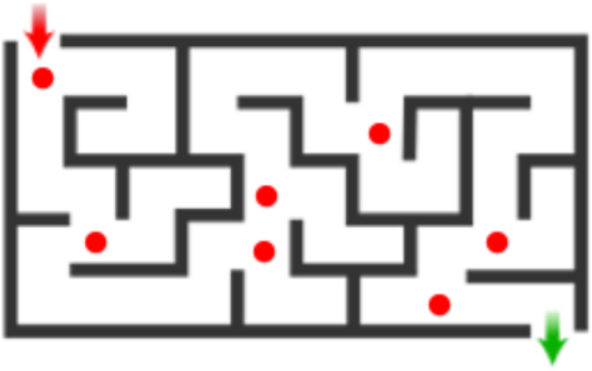


An example of supervised learning is decision tree analysis. This is a tool for decision support that uses a tree-like graph which shows decisions and their possible outcomes, costs and benefits. It shows the yes/no questions you are faced when making a decision. It allows you to approach the problem in a structured and systematic way and arrive at a logical conclusion.




An example of unsupervised learning would be K-means clustering. It aims to group n observations into k clusters. Centroids are placed apart from each other as to expect where clusters will appear. It goes through the dataset and select which datapoints have the smallest distance to each centroid and assign them to a centroid thus. It takes an average position of all the points assigned to a centroid and moves the centroid to this point. Then it repeats this process until no points are reassigned.


An example of reinforced learning would be the Markov process. This is an algorithm Where outcomes are partly random and partly under control of a decision maker. As described above it must make decions and define the new state it is in. It learns from incorrect decisions and adjust until the correct solution has been found. Each time it makes a mistake it learns from it and will not make it again. A good example of this process would be navigating a maze.



## 6. Ethics

In order for firms to compete in the highly competive field of software development they can't afford to ignore the value the value of measuring and collecting data on the performance of their employees. However conflict can arrise when employees feel like their privacy is being violated or they are being unfairly scrutinised. Too much scrutiny can start to have an effect on their mental welfare and the quality of their work can dip. It is incredibly important to abide by all existing legislation

involving data protection and ensure all staff are aware of what kind of data is being collected about them and how it is being used.

From a legal standpoint, data protection has massively increased in importance for companies in recent years. There has been an introduction of new EU wide general data protection rules from the 25th May 2018. GDPR emphasises transparency, security and accountability by data controllers and processors, while at the same time standardising and strengthening the right of European citizens to data privacy. There are huge implications to not following these rules with firms who are in breach of them liable to be fined the larger of €20 million or 4% of annual turnover. Naturally with the consequences being so severe it is abundantly clear as to why ensuring everything is above board and transparent when it comes to data protection is of the utmost importance to large companies.

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. Sometimes it can be legally ambiguous which country's law is applicable. For example, in Ireland, if the cloud server center of a company is located in Ireland, then your data is safe and secure within Irish legislation. However, if your company is based in several countries but has a presence in Ireland, it can be hard to tell which laws apply. The result of this being that some businesses themselves can end up struggling to understand the conflicting requirements, laws and regulations that exist.

An important question we are faced with today is who really owns our data? When someone posts something to Facebook do they own this information or is it Facebook's property. Legally this is grey area as most of our existing laws focus on physical property in terms of ownership. Facebook even now in their terms of service acknowledge they will keep all of your data even after you have deleted your account. It is a tricky issue as the amount of data we create is enormous. Every time you turn on your phone and open an app you are leaving behind a trail of information that someone is looking to use and profit from.

My personal opinion is that the measurement and assesement of software metrics is an ethically grey area. It is perfectly acceptable for a company to harvest large amounts of data on the performance of their staff and tracking progress on their many projects. This data is crucial in increasing their efficiency, improving their output and generating greater profits. Providing all legal obligations are met and there is an understanding amongst all staff so that they are fully aware of what is being measured and tracked there is little room for ethical concern. The understanding is important as a working environment where distrust and discord is rife between management and employees is a terrible working environment which hinders success. When a new assesement process is being considered, the opinion of staff should be considered as new measures can lead to tension especially if they are

felt to be unjust or overly intrusive. Backlash can occur when people feel they are being monitored and scrutinised such as when Sgt. McCabe presented his report on bad practices in the Gardai and was met with extreme opposition.

However it is easily possible to overstray the bounds of acceptabillity. It is possible for huge companies to ammass enormous quantities of data on individuals that far exceeds what is necessary or appropriate. Companies are able to mine data that reveals information about their private healthcare so they can discern which employees are trying to conceive or have diabetes etc. This invasion of someone's private life that has little to no bearing on their performance in the working environment represents a huge ethical mistep in my opinion. While I feel it is permissable to collect as much data as you want relating to work as you want, I believe it is a step too far when you are examining someone's personal information to this degree. If the data is not immediately relevant to the workspace then employers should have no business collecting and examining it.

## 7. Conlclusion

In brief, this report has outlined the various components associated with the measurement and assesement of software engineering process. It has examined the measurable data, the platforms available, the algorithmic approaches and the ethical concerns. It is clear that there are many ways to go about this process and that is an extremly crucial element of the development process. However despite its importance great care must also be taken to ensure no ethical boundaries are crossed as it so easy to go too far in terms of harvesting data and creating a terrible working environment.

References:


1. Unr.edu. (2018). *What Is Software Engineering?*. [online] Available at: https://www.unr.edu/cse/prospective-students/what-is-software-engineering [Accessed 19 Nov. 2018].
2. Softwareengineerinsider.com. (2018). *What is Software Engineering? | A Common Question*. [online] Available at: https://www.softwareengineerinsider.com/articles/what-is-software-engineering.html [Accessed 19 Nov. 2018].
3. Kan, S. (2014). Metrics and models in software quality engineering. [Place of publication not identified]: Addison-Wesley.
4. Lethbridge, T., Sim, S. and Singer, J. (2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. Empirical Software Engineering, 10(3), pp.311-341.
5. Searching Under The Streetlight For Useful Software Analytics IEEE Software (July 2013) by Philip M. Johnson
6. Stackify. (2018). *What Are Software Metrics and How Can You Track Them?*. [online] Available at: https://stackify.com/track-software-metrics/ [Accessed 19 Nov. 2018].
7. Stackify. (2018). *Development Leaders Reveal the Best Metrics for Measuring Software Development Productivity*. [online] Available at: https://stackify.com/measuring-software-development-productivity/ [Accessed 19 Nov. 2018].
8. W.S. Humphrey (1995), A Discipline for Software Engineering, Addison-Wesley
9. Codeclimate.com. (2018). *About | Code Climate*. [online] Available at: https://codeclimate.com/about/ [Accessed 19 Nov. 2018].
10. Codebeat.co. (2018). *Automated code review for mobile and web*. [online] Available at: https://codebeat.co/ [Accessed 19 Nov. 2018].
11. Brownlee, J. (2018). *Supervised and Unsupervised Machine Learning Algorithms*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ [Accessed 19 Nov. 2018].
12. Analytics, B. and Codes), E. (2018). *Essentials of Machine Learning Algorithms (with Python and R Codes)*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ [Accessed 19 Nov. 2018].
13. Dataprotection.ie. (2018). *GDPR - Data Protection Commission - Ireland*. [online] Available at: https://www.dataprotection.ie/docs/GDPR/1623.htm [Accessed 19 Nov. 2018].
14. WhatIs.com. (2018). *What is data sovereignty? - Definition from WhatIs.com*. [online] Available at: https://whatis.techtarget.com/definition/data-sovereignty [Accessed 19 Nov. 2018].
15. Lifehacker.com. (2018). [online] Available at: https://lifehacker.com/you-dont-own-your-data-1556088120 [Accessed 19 Nov. 2018].