Measuring the Software Engineering Process

# Introduction

Everything in today’s world is analysed on a daily basis. Low level managers analyse. Global corporations analyse. You and I analyse. It seems we live in a world where everything we do is scrutinised and critically evaluated in a never ending attempt to maximise efficiency and improve quality. Processes are measured and assessed, problems are identified, and action plans are drawn up to tackle issues which may arise. These processes occur in a variety of environments; ranging from the formal setting of the workplace, to more personal context of self-improvement.

The first thing that springs to mind when we speak of measurement and assessment, however, is numbers. Figures. Data. This would be a reasonable connection to make. With the technology available to us at present, we can make unfathomable computations with the click of a button. We have reams and reams of data at our fingertips. Hence, the onus of carrying out these huge computations and sorting through these mounds of data falls on machines and software. Technology ultimately allows us – the user – to draw conclusions from processed data.

We can analyse processes, behaviour and data in this way to identify customer trends, calculate insurance premiums, forecast the weather and detect fraud to name but a few. So, how is this practice applicable to the realm of software engineering? Throughout this report, I wish to identify and discuss the ways in which software engineering practices can be measured and assessed, in terms of what data we can use, where and how we can compute this data, the algorithmic approaches available and the ethics surrounding this area.

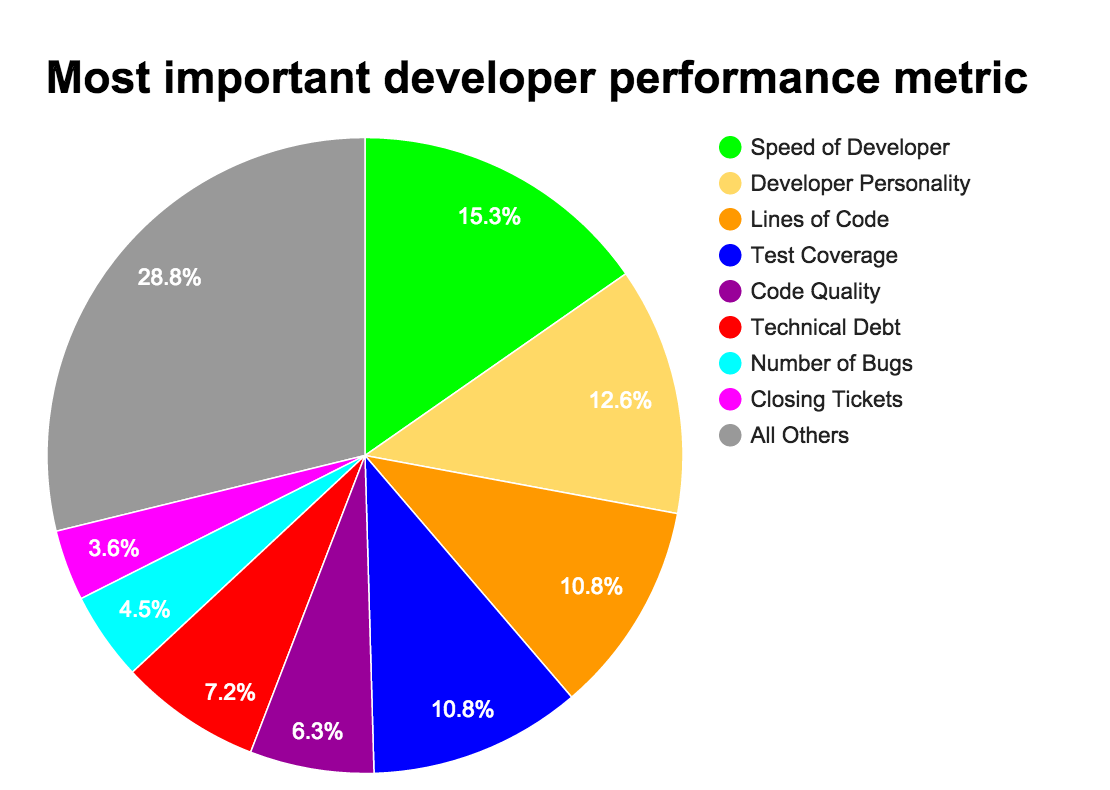
# What data?

First and foremost, before looking at different types of data, we must consider *why* we use data to assess a software engineer’s performance. The simple answer is to monitor work, and make improvements. In general, the aim is to minimise waste – whether that be cost or time – and maximise efficiency and productivity. The more difficult task we face is considering what data do we use to do this? What should be taken under consideration? What data is relevant and important to assessing a software engineer’s performance?

## Time and Size data

Two commonplace and familiar ideas about measurement in general are the time taken to complete a set of tasks, and the size of said tasks. These are quantitative measures as outlined in the Personal Software Process (PSP) [1], a framework for people working with software. In theory, it allows software engineers to track development, identify areas for improvement, and ultimately assess and measure performance. When contemplating performance levels, details about time and size are most likely the first things that spring to mind, as can be seen from the pie chart below.

They need to be looked at collectively. If we are told that Usain Bolt ran a race in 9.58 seconds, and Mo Farah ran a race in 26 minutes and 46 seconds, and we are asked who performed better, you would want to know the distance of each race. Of course, the former is the 100 metres and the latter is the 10,000 metres. The size of Mo Farah’s race is much larger, and needed more time to complete, but is an equally impressive feat as Bolt’s. This brings into perspective the critical link between time and size data.

Detailed time logs need to be recorded in order for assessment to take place. This includes keeping note of breaks, time spent checking your Facebook notifications, or anything else that takes your attention from the task at hand. This detail is essential in actually gauging how long has been spent carrying out each respective task. It is also paramount to review these log times in conjunction with the size/difficulty of the task completed in this time. This method is widely used, meaning time logs can be compared against historical data in order to measure performance.

Time (speed of developer) was voted to be the most important performance metric in an online poll among over 300 developers.

However, there are drawbacks. If time and size are the performance metrics in question, what is stopping the engineer from overstating the complexity of the tasks at hand, and always finishing before the deadline? Furthermore, this method measures time efficiency in relation to varying task complexity, but does it really measure code quality? I think not. For these reasons, they should be classed as least important categories of data according to the taxonomy of software engineering measurement.

## Bug Fixes and Unit Testing

Perhaps the data we should be considering when assessing performance lies in bug fixes. A software engineer might said to be performing well if he/she makes a high number of bug fixes per quarter. However, what if the code he/she was writing simply had a high instance of bug occurrences due to mistakes or laziness? After all, it would be easy to “accidentally” create bugs and then fix them just to drive statistics in your favour. Introduction of ratios helps to identify which developers are actually performing well in this scenario.

Rather than focusing on the possibly misleading figure of the number of bug fixes, we could focus on the ratio of defects to lines of code (LOC) written upon product completion. A low ratio here might indicate a more careful development approach and stable code base. Furthermore, if we were to analyse data such as the rate at which solutions are found compared to the rate at which they arrive (Backlog Management Index (BMI)), we might find one engineer to be clearing bugs more thoroughly than others – by minimising the occurrence of bugs reopening (a “fixed” bug which continues to be flagged as an issue).

Possibly the most important aspect of fixing bugs is unit testing. It is critical to carry out rigorous testing – through Junit or otherwise – in order to identify bugs before delivery to the customer or end user. Often, testing in this way is the last phase of development before release. Perhaps we could analyse code coverage data to measure how rigorously an engineer has carried out testing? After all, code coverage is a good indicator of the quality of testing that has been carried out. If we analyse this type of data and discover that a certain software engineer is consistently producing bugged code due unsatisfactory testing, surely this is a cause for concern.

## A non-quantitative approach

So far, I have discussed what *data* we should be analysing to accurately measure and assess the software engineering. What if we didn’t use classic ‘data’ at all? Or at least use data in conjunction with a non-quantitative method of assessment. As we have seen, certain models of monitoring processes and progress have their pitfalls. There are ways of polishing figures to make the software engineering process look like its running a lot smoother than it actually is.

My suggested solution is to incorporate a certain human level of judgement into the measurement and assessment protocols. Of course, it is useful to cite of a variety of datasets, graphs and ratios which indicate that employee A’s performance levels have dipped over the last 3 quarters. But why back up this data with, for example, an independent performance review panel? It would surely be possible to gather feedback on how employee A’s (and all other employees/software engineers for that matter) performance has behaved recently.

Hard data is fantastic. But to rely too much on it is dangerous. Consider a situation whereby employee A’s performance charts are not satisfactory, and following a meeting with the board, the decision is made to fire him/her. Now consider an alternative situation whereby employee A’s performance charts are not satisfactory, but this time, following a meeting with the board, the decision is made to establish a review panel of employee A’ performance. This panel finds that employee A is a top performer, and a hard worker. Somehow, our hard data did not reflect this. The hard data was focusing on employee A’s weaknesses, but did not reflect areas in which he/she excelled. A false negative has been unearthed.

Surely, using soft data and hard data in tandem would help to eliminate such false negatives (and positives). It is crucial to get this balance right when measuring the software engineering process. Over-reliance on either breeds unnecessary inefficiency/inaccuracy.

# Where to compute?

Having considered what measurable data we should be collecting in order to assess the software engineering process, we must now consider what we do with this data. In an ideal world, one can collect all available information on code defects for example, and use this data to produce plots, graphs and statistics which provide us with answers about where processes can be improved. In reality, it’s not that easy. In most cases, raw data must be computed and processed before we can draw such conclusions, and this is something we cannot do ourselves. So, where should we compute our data?

## Data Analytics

Data analytics is in demand right now. Processes are being scrutinised and put under the microscope now more than ever, as we search for inefficiencies and strive for improvement. But how do we identify problems using data? Through analytics. Companies are out there to provide this service for us. These companies provide a computational platform which allows us to process our raw data, making it easier for us to draw conclusions and answer process questions.

The reasons for using such a service are plentiful. Often, data analytics companies provide a user friendly interface which certainly makes our lives easier. Also, we most likely do not have the equipment to carry out these computations ourselves. Furthermore, outsourcing this service is a cost effective alternative to processing the data ourselves, which is usually carried out on hugely powerful platforms of integrated software. Building or acquiring such an engine is probably not a viable option logistically, never mind financially.

Draw conclusions

*Computational platform*

## Companies and Service Level Agreement

There is a vast array of companies out there that specialise in this area. However, they are not all the same. When considering where to compute, it is important to keep in mind what we are trying to find out. Yes, corporations are classified under the broad spectrum of providing a data analytics service, but many offer varying degrees of utility software. We don’t want to pay for something which we won’t use, or don’t necessarily need.

For example, DataTorrent is a big data company which claims its processing engine can “deal with billions of events per second”. Other business, such as DataHero, lean towards business intelligence and users with limited technical expertise. Since we are looking at the software engineering process, these properties don’t seem to cater us. Code Climate, on the other hand, tends to focus more on the technicalities of software. This includes technical debt, code quality, code review and maintenance.

This seems to be the perfect tool to sort through our data and eventually assess and measure the software engineering process. The service is linked in with GitHub, a development platform which encourages process improvement by facilitating user interaction through public repositories. The very fact that this link exists speaks volumes about Code Climate’s goal here, which is to improve overall code quality. We can use this code quality analysis to not only measure and assess performance, but also improve it.

As mentioned above, code quality is assessed under categories such as technical debt and test coverage. It provides a computational platform which is able to measure the performance of these categories as a percentage. Users can track progress of code quality improvements through summaries, identify trends in their code base, and also make comparisons across projects.

## The Real World

What do these capabilities mean for the real world? It means that we can measure, under specific categories, the quality of code that we have built using a computational platform which is integrated with GitHub – a developmental environment for code storage. In short, software engineering processes can be measured and assessed all in one place using Code Climate. This can be done for one’s own self-improvement, or could be implemented to monitor employees within a corporation.

This doesn’t revolutionise the way in which software engineers are being assessed, but it does show computational platforms are available which provide varied measurement parameters. It is certainly a step in the right direction. Improvements could be made, however, such as the introduction of more assessment categories. Thus broadening and solidifying the reliability of assessment. If implemented on a larger scale, possibly in the work place, work quality must certainly improve. Technically literate managers could easily track underperforming employees/processes, but also identify strong performers.

In light of this, there are services offering more than adequate computational platforms to assess and measure software processes. However, there are always drawbacks. When dealing with data, there is always the looming question of security issues. Leaked personal information can lead to legal nightmares, which will be discussed later on. Also, the way in which we assess the software process here is very one dimensional. Yes, we have a platform which gauges good code quality and efficiency, but there is no real focus on the code’s objectives. In other words, our code might score well statistically, but not do what we set out for it to do.

# What algorithms?

Data analytics, as discussed above, is a very important process in what we’re trying to achieve here – to measure and assess the software engineering process. We must not accept it for the broad term it is, however. The question still remains, what algorithmic approaches are available to us? Which ones are relevant and useful to us? There are infinite ways of approaching a problem, some of which are more suited than others.

## Brute Force V Heuristic

Decision making is at the heart of using algorithmic approaches to tackle a process or problem. Assume we have already established the quality that is important to us – accuracy, efficiency, cost etc. One route way of approaching a process, such as the software engineering one, is a brute force algorithm – exhausting every single possibility and distinguishing between those results which are satisfactory or not. This method considers every eventuality, but is far from cost/time efficient. Furthermore, we may struggle to classify the software engineering process test every possibility.

An obvious alternative is to use a heuristic algorithm – one which will find a solution faster, but may sacrifice accuracy in doing so. A heuristic, when applied to the software engineering process, will find us a viable process without the guarantee that this process is optimal. In any case, there seems to be a trade-off between time efficiency and algorithm accuracy here. A middle ground of sorts is desirable in such a situation.

Turning our attention elsewhere, there is always the option of building our own algorithm, using mathematical modelling. Often, a high degree of expertise is needed in order to do this. Take formal specification, for example, which uses mathematical formulas and equations as a basis for building an accurate algorithm. Often, software engineers do not have the capabilities to use such a complex technique. An equally probably scenario is that it may be too difficult to take real life problems and model them using mathematics.

## Machine Learning Algorithms

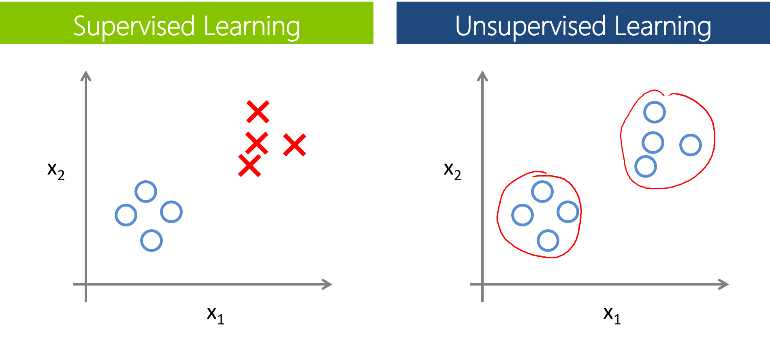
This is where machine learning, or computational intelligence comes into play. This is not to be confused with artificial intelligence, which operates in a binary fashion. Of course, not everything in life comes as a series of 0s and 1s. Computational intelligence on the other hand, has the ability to work with incomplete and unclear information – similar to how the human brain works. The idea is that machines can ‘learn’ to make predictions and identify patterns without being explicitly programmed to do so, using data driven methods.

Supervised learning algorithms are those which use training data, which maps known inputs onto known outputs. The purpose of this is to ‘train’ the machine to eventually deal with a truly unknown input and produce an output using a predictive model. All data used is labelled. An example of supervised learning is logistic regression.

Consider the following example of logistic regression in practice; we are given data from 92 people on a) what they weigh (lbs), b) whether or not they smoke (1 or 0), and c) if they have high blood pressure or not (1 or 0). Using logistic regression, we can model our given data to predict for an unknown input. We can estimate the probability of person A having a low resting pulse given they weigh 190lbs and smoke. We can make a classification on a subject provided with incomplete data. This is a very powerful tool. K nearest neighbour and decision tree analysis are two more examples of supervised learning.

An unsupervised learning algorithm find patterns and identifies irregularities by itself – it does not use predefined training data. It does not use class labels either. It is easier to obtain this kind of unlabelled data, and is a more realistic method to use in research, such as the research carried out which helped to uncover that Kawasaki Disease is linked to large scale wind currents. An example of unsupervised learning is Principal Components Analysis (PCA) and clustering.

PCA is a dimension reducing technique, used to identify associations among variables. It is used to describe a set of correlated variables ( in terms of a new set of uncorrelated variables (, where m < n, and B1 encapsulates more variation than any other combination of the original data An. It is used in a variety of fields ranging from biology to quantitative finance.

Consider the following example of PCA in practice; an investor has placed money in 100 stocks and wants to predict which stock prices will rise and fall. Considering the stocks individually, he would have to consider 100x100 complex interaction of variables/factors. He carries out PCA and finds that 3 Principal Components (PCs) explain the stock’s movements. This may allow him to draw conclusions about what the PCs mean, and where he should invest more heavily.

## Real Life Application

Machine learning algorithms are the most applicable when it comes to the software engineering process due to the fact that we are dealing with real life and real people. How can they be implemented in the real world? Supervised learning such as logistic regression would certainly help in assessing processes. We can take a set of labelled training data, such as number of commits, LOC written, key strokes etc. and use it to ‘train’ the machine to gauge how different inputs of each variable impact the software engineering process in a positive or negative way.

For example, let’s assume project A contains the following data: 461 LOC, 64 commits, 4 team members and 6 days to complete. Using our logistic regression model, the software engineering process employed here only scores 53%, because it took too long to complete and not enough commits were uploaded. This hypothetical conclusion has been drawn from results extracted from our training data, which describe the optimum levels of different inputs that result in the software engineering process operating at maximum efficiency.

Unsupervised learning algorithms may also be a useful tool. The software engineering process is composed of a variety of complex variables which interact with each other in many ways. Perhaps carrying out PCA may succeed in reducing the dimensionality and complexity of measurement, as we discover that, for example, 78% of variance is accounted for in PC1. Imagine we realise that this variance is an overall measure of brainpower working on a project, and larger teams improve process efficiency by combining knowledge.

Both machine learning algorithms are adequate assessment tools, but which is best? Supervised learning is easily monitored and can yield accurate results, but good training data values must be found, which takes time. Big data is an issue for sure. Furthermore, we don’t actually know what labels are to be included in training data in the first place. There are so many things to consider. For all we know, the weather on a particular day might influence measurement and performance. If training data does not reflect real life scenarios, unknown inputs will be classified incorrectly.

Unsupervised learning assumes nothing and uses no training data. Its data driven approach focuses only on available data, and unearths patterns by itself. This research orientated approach is a better method, in my opinion. I think keeping an open mind is important when considering the software engineering process. If we assume nothing to begin with, we remain impartial to what the results will yield. Using this method, different data can be added and removed at will (without the need to create new training data). There is also space for human interpretation, which eliminates the possibility of false positives and preconceived assumptions which may arise in supervised learning.

# Ethics

Throughout this report, the main focus has been on the logistics of collecting data, where we compute it and using what method. This has all been scrutinised in an attempt to interpret the ways in which the software engineering process is measured and how this could be improved. The ethics of this practice is something that hasn’t been touched on. Data ethics has turned into a major issue in the last number of years, as data manipulation and security concerns have become more prominent in global media.

The growing concern is that perhaps there is no concrete privacy line when it comes to data. Take Facebook for example. The social media giants implement personal ads beside your news feed, showing products specifically tailored to you. The type of information they have on users which allows them to do this is astounding. According to a study in 2016, Facebook is able to identify users who plan on buying a car, what groceries a user buys, types of vacations a user tends to go on, and relationship details etc. This begs the question – is anything *actually* private anymore?

“Business managers who collect, store, manage and process our data are not being held to any ethical standards” – (K. Fung).

## At a Corporate Level

Studying a company is a good way to tie these ethical issues in at a corporate level, thus analysing how they are relevant when considering the software engineering process. It may seem that a record is kept every time we sneeze, such to be the extent that our data is constantly being collected and stored. Maybe such rigorous data collection can be a positive thing.

A company called Steelcase, which innovates in state of the art, high tech office furniture, has produced a desk chair which monitors everything from heart rate to posture. The chair displays information on an iPad, encouraging the user when to take standing breaks and employ breathing exercises based on the data gathered. Personal, almost intrusive data has been used to good effect here, to help maximise user comfort and sustain productivity.

In another example, an organisation by the name Humanyze has produced a smart badge tracker that employees wear around their necks. This badge records conversations, measures tone of voice and tracks employee movements among other things. This information is uploaded for managerial analysis. The aim is to quantify social interactions in a bid to increase teamwork and engagement, improve collaboration and delivery, and better performance. The idea behind this product is great, but this sort of data collection crosses the line in my opinion. Sure, employers want to measure process efficiency but imagine an employee not being able to speak his/her mind during work, for fear of repercussion from recordings?

## The Privacy Line

This is where a privacy line needs to be re-established. In the business examples above, invasive information has been gathered, but there is a major difference in what is done with it. Steelcase simply uses the data to interact with the use to maximise comfort, which would inevitably promote productivity in the workplace. Humanyze takes this data and gives it to your superior for detailed analysis. It almost resembles spying!

Of course, in most cases, the motives behind meticulous data collection and analytics are noble, but we cannot assume this for every scenario. Problems will arise if we allow access to such in-depth analytics in other fields, such as job interviews. Without explicitly asking, and given enough data, it is possible to answer questions which shouldn’t be allowed to be found out.

Consider a job application, where the employer is not allowed to directly ask about the ethnicity of the candidate, but asks 101 other things which indicate to him that the applicant is of a different ethnicity to him. Suppose the employer is a closeted racist, and fails to offer the applicant the job, despite being the most qualified. The same scenario can arise with gender. We need to restrict the availability of such detailed data in certain scenarios to limit the occurrence of incidences listed above.

## Image result for data leaks factsData Leakage and Other Concerns

There is also the unlikely, but deeply harmful and possibly dangerous situation that personal data can be leaked. Suppose your hospital records, which contain details of a recent diagnosis, have found their way into an insurance company’s hands. This might hinder your ability to obtain insurance in the future. Especially in light of the events in 2010, where major insurers such as Aviva and Allianz were found to have illegally stored personal information in a vast database, which was used to calculate premiums and produce insurance quotations.

As can be seen, both the number of leaks and number of leaked records are increasing since 2011, with the latter rising more swiftly.

Sometimes data protection issues can lead to legal battles and court cases, often leading to lengthy hearings which consume tax payer’s money. An example of this occurred only last month, in October 2017 when Facebook were accused of breaching EU data privacy right by transferring data from the EU to the US. Facebook maintain they did nothing wrong, while Max Schrems’ (the Austrian lawyer who made the allegations) accusations were described as “well-founded” by a High Court Judge.

These situations arise because data protection is such a grey area. There are National laws, but no global consistency – a clear flaw in the world we live in today. Not only this, but cybersecurity hackers are rapidly becoming more intelligent and destructive. In 2015 alone, 177,866,236 personal records were exposed due to security breaches. That’s over 2% of the global population. Security is a major factor when debating what data should be exchanged in cyber space.

We need data in order to measure and assess any process. Software engineering is no different. I completely support the idea of using data to analyse processes and make improvements, but one must also think of the dangers and implications. We may be able to use this information to improve developer performance and identify process flaws, but people are entitled to privacy.

Having said that, I do believe that measurement in itself is not a problem. It is a necessity in business. However, due to the risks outlined above, data protection laws – of whatever severity deemed necessary – need to be set out in stone and standardised. Global protocols should also be established to provide a benchmark of data security standards. These standards would allow measurement to take place in a more secure environment, where the stance on data protection is clear and strictly enforced.

# Summary

The software engineering process is complicated one. What information do we need in order to assess it? Performance can be measured in many ways, but the key lies within combining soft and hard data. We use taxonomy to classify which metrics are most important according to the specifics of assessment, and place more emphasis on these. In combining this with a non-quantitative approach, we can use human reasoning and logic to support, or debunk, what the data tells us.

The missing piece between raw data and drawing conclusions is a computational platform which allows us to process data, and make it legible. Conclusions and inferences about process improvement could not be made without this. Data analytics comes in here, as we often do not have the power to compute this data ourselves. Software specific companies should targeted, such as Code Climate. The business, which is linked up with GitHub, encourages development, code analysis and process improvement through statistics and visualizations.

This answers the questions of what data to use and where to compute it, but what algorithmic approach should we take? Of course, different algorithms will yield more accurate results than others, but this will almost solely be based on what process we’re curious about. Computational Intelligence is an attractive proposition for our purposes, as we can make specifications based on our data. Machine learning algorithms possess attractive properties, in that they can make predictions about process efficiency based on data inputs, which can be classified or not.

There are limitations to what we can measure, however. In an ideal world, we would have complete access to information required to maximise process efficiency. Due to threats such as cyberattacks and data leaks, as well as a person’s fundamental right to privacy, the software engineering process must be measured using fuzzy, incomplete data. No matter how much time and money we invest in studying methods of assessment, it is currently not possible to achieve perfection in the software engineering process, meaning human initiative and creativity are still paramount to high-level performance.

# Bibliography

1. Watts S. Humphrey (2000), “The Personal Software ProcessSM (PSPSM)”. [online] Available at: <https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf> [Accessed 21 Nov. 2017].
2. Brian York (2017), “The Best Developer Performance Metrics”. [online] Available at: <https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0> [Accessed 21 Nov. 2017].
3. Steven Kan (2002), “Metrics for Software Maintenance | Software Quality Metrics Overview | InformIT”. [online] Available at: <https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf> [Accessed 21 Nov. 2017].
4. Phillip M. Johnson (2004?), “Practical automated process and product metric collection and analysis”. [online] Available at: <https://pdfs.semanticscholar.org/0f5e/d29477e40ecb95298c759372eac7088cc60f.pdf> [Accessed 21 Nov. 2017].
5. Bob Brown. (2017). “13 big data and analytics companies to watch”. [online] Network World. Available at: <https://www.networkworld.com/article/2970498/big-data-business-intelligence/13-big-data-and-analytics-companies-to-watch.html#slide5> [Accessed 22 Nov. 2017].
6. Codeclimate.com. (2017). [online] Available at: <https://codeclimate.com/product> [Accessed 22 Nov. 2017].
7. S. Malathi (2011). “An Algorithmic Approach for the Implementation of Analogy-X for Software Cost Estimation”. [online] Available at: <https://www.researchgate.net/profile/Sridhar_Seshadri2/publication/288829203_An_algorithm_approach_for_the_implementation_of_analogy_X_for_software_cost_estimation/links/5685134308aebccc4e114533/An-algorithm-approach-for-the-implementation-of-analogy-X-for-software-cost-estimation.pdf?origin=publication_list> [Accessed 25 Nov. 2017].
8. Brett Houlding (2017). “Logistic Regression”. Scss.tcd.ie. [online] Available at: <https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides12.pdf> [Accessed 25 Nov. 2017].
9. Beril Sirmacek (2017). [www.create4D.com](http://www.create4D.com) “What are the advantages and disadvantages of a supervised learning machine?” [online] Available at: <https://www.quora.com/What-are-the-advantages-and-disadvantages-of-a-supervised-learning-machine> [Accessed 25 Nov. 2017].
10. UC Health - UC San Diego. (2011). Kawasaki Disease Linked to Wind Currents. [online] Available at: <https://health.ucsd.edu/news/2011/pages/11-10-kawasaki-disease-and-wind-currents.aspx> [Accessed 25 Nov. 2017].
11. Stanford University – “Unsupervised Learning”. [online] Available at: <https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/unsupervised.pdf> [Accessed 25 Nov. 2017].
12. Brett Houlding (2017). “PCA”. Scss.tcd.ie. [online] Available at: <https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides3.pdf> [Accessed 25 Nov. 2017].
13. Kaiser Fung (2015). “The Ethics Conversation We’re Not Having About Data”. [online] Available at: <https://hbr.org/2015/11/the-ethics-conversation-were-not-having-about-data> [Accessed 27 Nov. 2017].
14. Parkkinen, J (2016). “Data Needs Ethics!” [online] Ouzor.github.io. Available at: <http://ouzor.github.io/blog/2016/12/21/data-ethics.html> [Accessed 27 Nov. 2017].
15. Caitlin Dewey (2016). “98 personal data points that Facebook uses to target ads to you”. [online] Available at: <https://www.washingtonpost.com/news/the-intersect/wp/2016/08/19/98-personal-data-points-that-facebook-uses-to-target-ads-to-you/?utm_term=.108d9dc04223> [Accessed 27 Nov. 2017].
16. Stephen Barrett (2017). “Measuring Engineering”. [online] Available at: <https://www.scss.tcd.ie/Stephen.Barrett/teaching/CS3012/measurement.pdf> [Accessed 27 Nov. 2017].
17. Stephen Mangan (2010). “Insurers investigated over illegal data storing”. [online] Available at: <https://www.irishtimes.com/news/insurers-investigated-over-illegal-data-storing-1.624926> [Accessed 27 Nov. 2017].
18. Mary Carolan (2017). “High Court asks ECJ to examine Facebook case”. [online] Available at: <https://www.irishtimes.com/business/technology/high-court-asks-ecj-to-examine-facebook-case-1.3242468> [Accessed 27 Nov. 2017].
19. DiGiacomo, J. (2017). “Hacking Statistics for 2015 and 2016: How Bad Will 2017 Be?” [online] Revision Legal. Available at: <https://revisionlegal.com/data-breach/2017-security-breaches/> [Accessed 27 Nov. 2017].
20. Heimdal Security Blog. (2017). “All About (Concealed) Data Leakage for Users Like You and Me”. [online] Available at: <https://heimdalsecurity.com/blog/all-about-data-leakage/> [Accessed 27 Nov. 2017].