Name: Brandon Sharkey

Student Number: 15323329

Course: MSISS

Abstract

The following report's objective is to contemplate the ways in which the software engineering method can be evaluated and assessed. I shall endeavour to do so by covering the following four topics: what quantifiable data is used, which computational platforms are offered, the different algorithmic approaches available and finally the ethical and legal implications of such an examination.

The Software Engineering Process

"Perfection in design is achieved not when there is nothing more to add, but rather when there is nothing more to take away"
- Antoine de Saint-Exupery [1]

Software engineering is most aptly described as the engineering discipline that is concerned with all aspects of software construction.

Software Engineers apply the tenets and techniques of engineering to all aspects of software production be that the design, development or deployment of systems, or the ongoing maintenance and management of such systems [2].

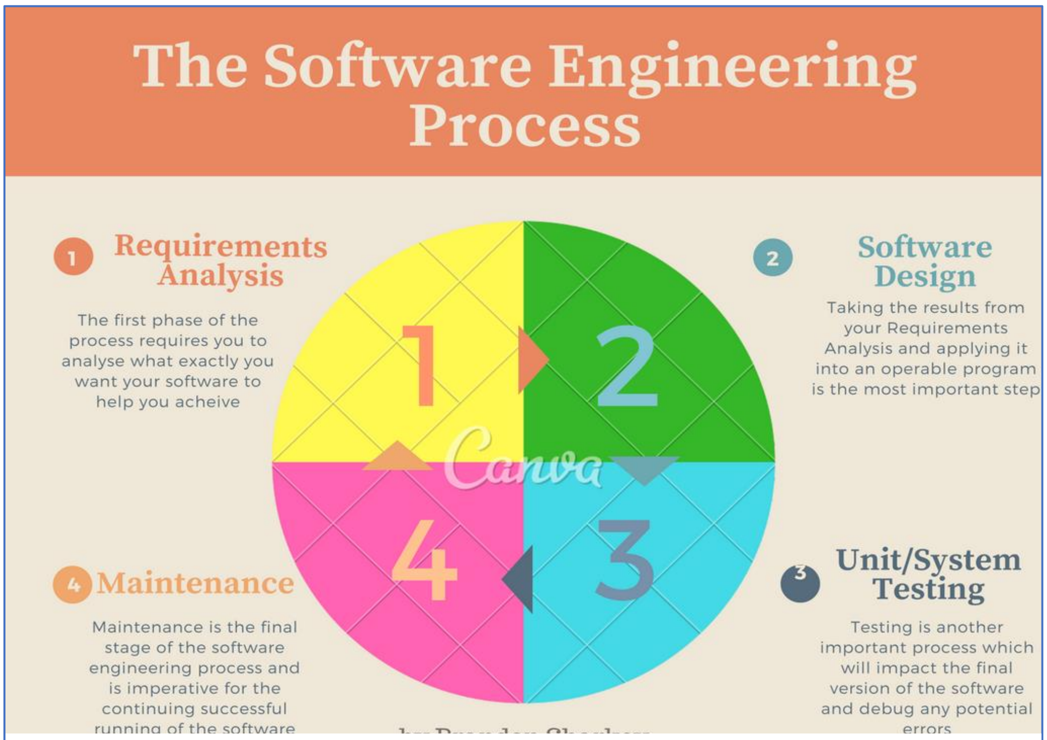

The Software Engineering Process
1 Requirements Analysis
The first phase of the process requires you to analyse what exactly you want your software to help you acheive
2 Software Design
Taking the results from your Requirements Analysis and applying it into an operable program is the most important step
3 Unit/System Testing
Testing is another important process which will impact the final version of the software and debug any potential errors
4 Maintenance
Maintenance is the final stage of the software engineering process and is imperative for the continuing successful running of the software
1
2
3
4
Canva

Fig 1. A basic diagram I created giving a brief illustration of the Software Engineering Process

Software Engineering focuses more on the practices of developing effective and efficient software for use in business and industry while those who are more aligned to the theoretical side of computer science are usually more focused on the theory and algorithms behind the actual software.

Software Engineering firms want to be able to gather data and information on the operations and practices of their different clients, particularly the clients that clearly participate in the process of developing new software and methods. They can then use these results to further improve their products and cut inefficient code. Though it should be noted that the data these firms seek can be more difficult to quantify than in other areas of engineering as software by its own disposition is impalpable [3].

What Data to Use?

The main issue when it comes to wanting to track the productivity of Software Engineers is figuring out what type of data one should be measuring. Data can be used to predict the development time of a project and identify which developers are more efficient at coding by analysing past performances and using them to predict future likelihoods.

There are some key matters that need to be addressed when looking to choose the data you plan on collecting and analysing: should the data be quantitative or qualitative, is all the data you are looking to collect relevant to what you are hoping to achieve and is the data collection process automated or not?

The main types of data one would look to collect from Software Engineers would all be based around their code production, though there would be other non-code based factors such as their participation with colleagues in meetings, advising on other people's public repositories such as GitHub or aiding on platforms such as Stack Overflow. For my report I shall be going in-depth on these code-based data types.

- Lines of code: While counting the lines of code produced over time by a Software Engineer at first might appear to be a good gauge of productivity it does present many pitfalls; as shorter code which does the exact same as longer code is said to be more efficient. Though the counter view to this is known as the SLOC (Source Line of Code) metric which promotes spreading out code over many lines when it could just as easily have been written well on a single line.

- Commit count: Counting the number of times a developer commits their work to an online repository (e.g. GitHub) over time can provide useful data though it doesn't consider what is being submitted in each commit as some developers might upload many small commits while some may do a few larger ones. It should be encouraged to constantly commit your code online, so it is safe if something happens to your PC, but this may be done all in one commit or over many smaller ones thus negating the impact of the commit count as an indicator of an engineer's productivity.

- Test Coverage: This involves computing the percentage of code covered by test methods. An effective software engineer will make sure that their code can handle all sorts of diverse issues and foreseeing these potential problems is an important aspect of being an effective software engineer and as consequence I think test coverage is a great barometer of a software engineer's productivity.

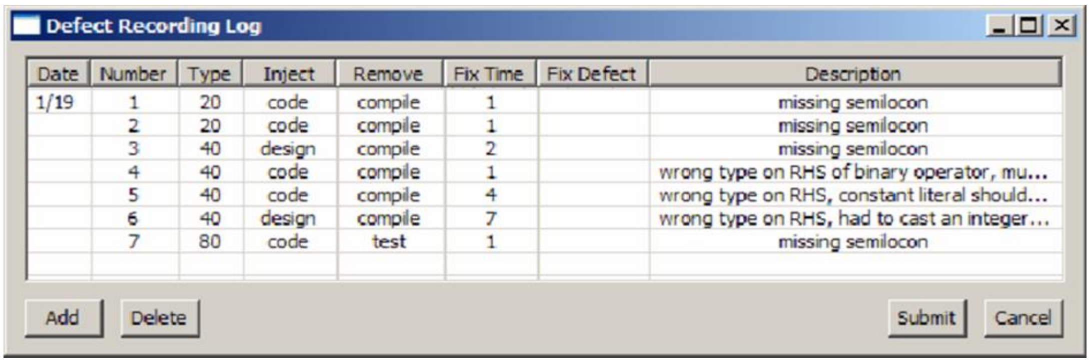
- Technical debt: Technical Debt (applying an easy short-term fix instead of the optimal solution) is arguably the best overall method of measuring a Software Engineer's effectiveness as a low amount of technical debt would show they are forward thinking with their design, not just looking for a 'quick-fix' that will temporarily solve an issue, which could cause further issues down the line. Possessing a technical debt of higher than 10% within their code is usually considered bad-practice for a software engineer.

- Debugging: Determining how long an engineer spends debugging code during a period is another useful method in estimating an engineer's productivity though it too is not without its pitfalls as an engineer may have more bugs to fix due to the fact he is not as proficient at coding as another engineer who doesn't create as many bugs. It also may be the job of a certain engineer in a team to focus solely on debugging and thus he would have spent much more time debugging code than fellow members on the team, though I do believe this is a better gauge of an engineer's performance than line and commit count.

In my opinion, the aforementioned five code-based data types would be the most applicable for measuring a software engineer's productivity, while within these five some would still be more preferable than others, none by-themselves would be considered to be *'the perfect metric'* but in combination they would provide as close to such a metric as possible.

Where to Compute?

There are numerous computational platforms available today where one can go to have their data analysed, previously the PSP (Personal Software Process) was the 'go-to' method for computation but it isn't as efficient as today's automated programs and is therefore usually overlooked by the larger firms.



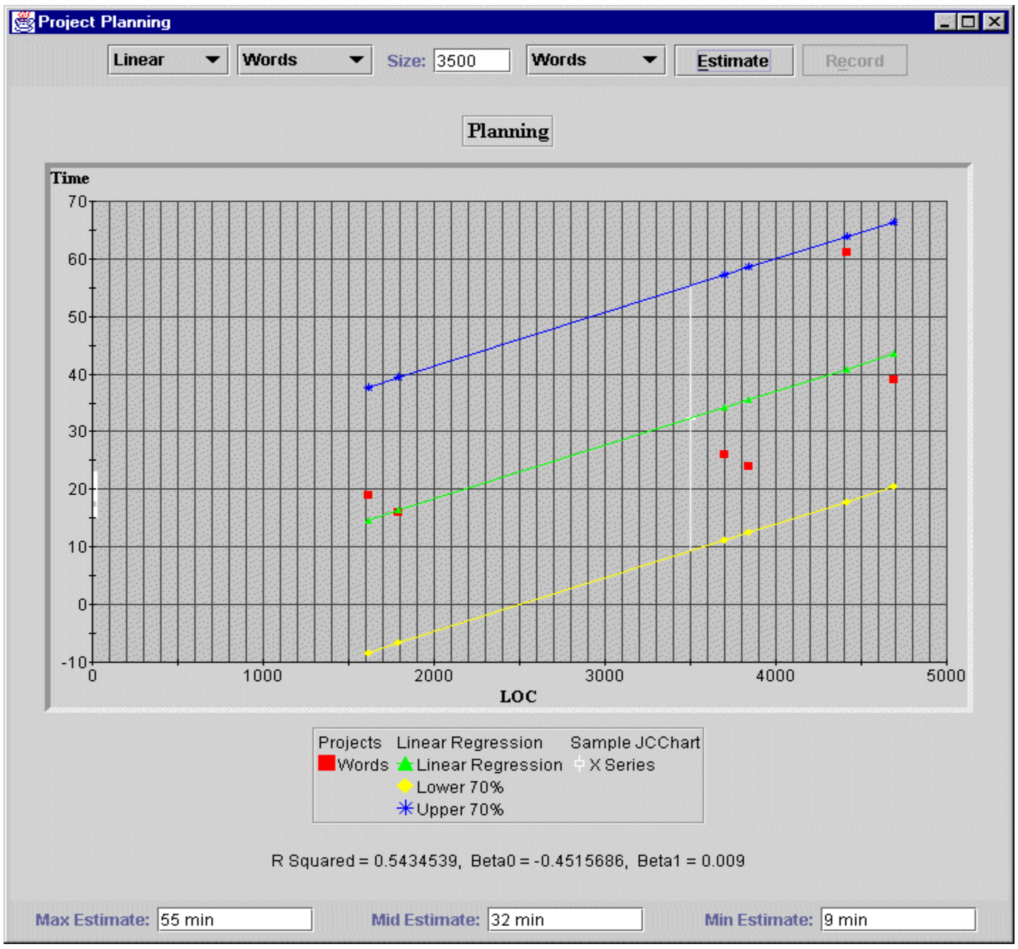
Date	Number	Type	Inject	Remove	Fix Time	Fix Defect	Description
1/19	1	20	code	compile	1		missing semilocon
	2	20	code	compile	1		missing semilocon
	3	40	design	compile	2		missing semilocon
	4	40	code	compile	1		wrong type on RHS of binary operator, mu...
	5	40	code	compile	4		wrong type on RHS, constant literal should...
	6	40	design	compile	7		wrong type on RHS, had to cast an integer...
	7	80	code	test	1		missing semilocon

Fig 2: An example of a typical Defect Recording Log used in PSP [4].

PSP provided Software Engineers with a rigid personal framework for developing software. It's creator Watts. S. Humphrey said of its purpose: "The PSP strategy is to improve the performance of practicing software engineers. A disciplined software engineering organisation has well-defined practices and its professionals use those practices and strive to improve their personal performance and hold themselves responsible for the quality of the products they produce and most importantly they have the data and self-confidence required to resist unreasonable commitment demands" he described the process as consisting of "a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work" [2]

PSP requires a lot of manual input on behalf of the engineer as according to Humphrey it is essential to collect "detailed metrics of the development time, bugs discovered and corrected at all development stages". The hope of PSP is that eventually you will be able to find and fully minimise all the inefficiencies present at all steps of the software engineering process, although the initial use of the PSP system will take a lot of added effort from the engineer due to the manual entry of all records and could thus have an impact on their productivity, which is why as previously mentioned it is no longer commonly used.

The LEAP toolkit was developed in an attempt to help automate the data analysis of PSP and by extending the degree of analysis of PSP by introducing more complicated regression models and thus increase the productivity of the engineer, though the engineer would still have to manually enter his/her data. LEAP created a personal repository of a developer's process data that they would be able to carry over as they move from one project to another. and extending the degree of analysis than PSP by introducing more complicated regression models.



Linear | Words | Size: 3500 | Words

R Squared = 0.5434539, Beta0 = -0.4515686, Beta1 = 0.009

Max Estimate: 55 min   Mid Estimate: 32 min   Min Estimate: 9 min

Fig 3. A Screenshot of the LEAP UI taken from the University of Hawaii website [5].

The University of Hawaii built 'Hackystat' as a more secure and privacy orientated version of LEAP. Following the entering of the data, Hackystat would allow developers to process and analyse their data automatically.

PROM (PRO Metrics) is another tool automated for collecting and analysing software metrics and PSP data. It was created by Eindhoven Technical University and it collects both code and process measures by using plug-ins that collect data from the tools the developers use.

The difference between the two data tools is PROM analyses the whole development process including non-code related topics, providing an analysis of all members of the team of developers. Whereas Hackystat focuses primarily on code related activities and the individual software developers.

There are many more similar tool kits available that have varying levels of automation, going from fully automated toolkits such as Hackystat and PROM to fully manually toolkits such as PSP and Jasmine; though all of these toolkits do tend to have rather high adoption barriers compared to other toolkits such as Agile, Sonar and Ohloh, according to Philip M. Johnson in his report *'Searching Under the Streetlight for Useful Software Analytics'*.

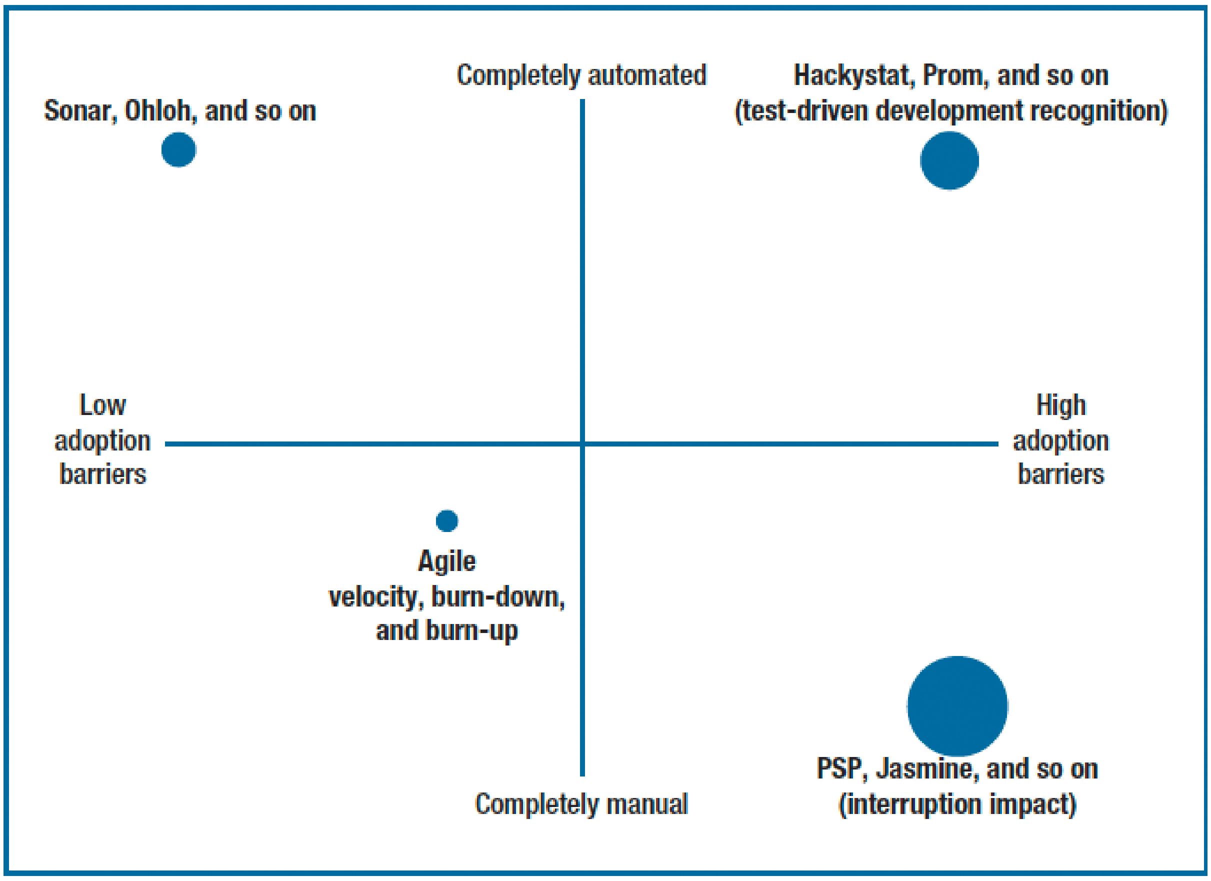


Fig. 4 A Diagram taken from Philip M. Johnson's report 'Searching Under the Streetlight for Useful Software Analytics' comparing the different software analytics toolkits and their varying levels of automation and adoption barriers [6].

Deciding on which toolkit to use all depends on the levels of automation you wish to have in analysing your data as well as how difficult the adoption barriers are to overcome for you and your firm.

What Algorithms to Use?

Artificial Intelligence

“the capability of a machine to imitate intelligent human behaviour”

- *Merriam-Webster Dictionary [7]*

In recent times AI’s ability to perform tasks usually reserved for humans has grown exponentially, replacing jobs once typical held by humans, be that in an automated car factory or a fast-food restaurant.

Though the one area that human intelligence trumps that of AI is in the creative fields as AI will solve a problem efficiently and effectively when handed that problem but coming up with a problem to attempt to solve is where human intelligence is required, as Michael Osborne of Oxford University stated, “for both the UK or the US, almost 90% of creative jobs are at low or no risk of automation.” [8] meaning for the foreseeable future AI won’t be able to identify problems of its own accord. For the Software Engineering Process this means that AI could process computations we pass to it to help measure productivity, but it would be unable to create new methods of measuring efficiency at least for the time being.

Computational Intelligence

“Computational Intelligence is an offshoot of artificial intelligence in which the emphasis is placed on heuristic algorithms such as fuzzy systems, neural networks and evolutionary computation.”

- *UK Workshop on Computational Intelligence, University of Manchester [9]*

Computational Intelligence’s heuristic algorithms are designed to solve problems faster and more efficiently than a typical algorithm might, by forgoing what might be the optimal solution. This quicker solution will be less accurate that a typical algorithm, so it should be noted not to fully regard its results as 100% correct.

Dissimilar to Artificial Intelligence, Computational Intelligence does not attempt to bring the ‘human element’ into the calculations, as in it does not attempt to give machines what we would call ‘emotional intelligence’. Some examples of the aforementioned ‘heuristic algorithms’ are ‘fuzzy systems’, ‘neural networks’ and ‘evolutionary computation’.

‘Fuzzy Systems’ attempt to bridge the gap between the binary logic of computers and the logic of humans by taking values between 0 and 1 as compared to *either* 0 or 1 as in a binary system. It deals with imperfectness, neither 0 or 1, thus the name ‘fuzzy’.

The ‘Neural Network’ is a set of heuristic algorithms modelled on the human brain and central nervous system. The algorithms in a Neural Network adapt and ‘learn’ over time as the more data it evaluates the more accurate the system becomes at successfully predicting future data. Most computers need to have the problem defined and the steps laid out before they can solve it but with Neural Networks the hope is that, like humans, Neural Networks will learn with time and practice.

‘Evolutionary Computation’ is based on the theory of natural selection where only the strongest survive. Uses for this are found in the fields of optimisation where traditional mathematical techniques are not typically suitable to solve a wide range of problems in this field.

Applying these Algorithms to Measure Software Engineering

It seems as if Computational Intelligence has all the hallmarks of being a very efficient method of measuring the software engineering process in the near future.

Previously the most comprehensive analysis was achieved when an engineer employed the aforementioned PSP as while it was very inefficient, taking excessive lengths of time for an engineer to fully record their data, apply the algorithms and then apply their findings to their own personal process it did give them a very in-depth look at their own inefficiencies. PSP also offered engineers total independence in selecting and customizing their own analysis, picking the metrics that best suited them, though this did come at a great time expense.

In the present day we can try and solve this problem by taking the engineer completely out of the equation and have the data be collected automatically though there can be no assurances that the data being collected would be the most effective at highlighting the engineer’s inefficiencies. Computational Intelligence would be able to fix this issue by bringing in tools that can adapt and change to deal with the differing data that comes in, the previously mentioned heuristic algorithms such as fuzzy logic and neural networks would be exceptionally useful at this type of adaptive behaviour.

When measuring the software engineering process from the point of view of an employer wanting to figure out which employee is the most proficient, the algorithms used would vary from the PSP approach where a single engineer would be measuring their own efficiency and therefore would be able to use the criteria of their choosing but when applying an algorithm for an overall prospective across a number of different engineers I firmly believe that a CI based algorithm would be the only ‘fair’ algorithm to use as it could adapt to deal with each unique engineer’s role within the system as the primary use of measuring an engineer’s proficiency should be to better them by minimising their inefficient behaviour, not comparing them to other software engineers.

Ethics

So far in this report my focus has been on the process of collecting the appropriate data, where we should go to compute it and what algorithms would be most applicable to the data. The ethical implications of this practice are something that I haven’t yet shone a light on. Data ethics has turned into a major topic in the last few years, as data breaches have become more prominent in global media, the Sony hack in late 2014 being an example of a data breach that gained widespread attention and which lead to numerous implications for the media giant.

The main issue is that it is hard to have a reasonable expectation of privacy nowadays with your personal information being held by large companies who can sell it on to advertisers to better sell to you. Having all our data in the hands of big companies does help tailor experiences to us but at the cost that private or sensitive information is being held by a company whose only responsibility is a fiduciary one to their shareholders.

“The bigger the network, the harder it is to leave. Many users find it too daunting to start afresh on a new site, so they quietly consent to Facebook’s privacy bullying.”

- *Evgeny Morozov [10]*

Employee’s Right to Privacy

Employers (under the auspices of the ‘DATA Protection Act 2003’) must be careful that any data they store does not intrude on the rights of an employee. If an employer hoards the personal information of its employees without the proper precautions being taken to secure the information they will be in breach of this act, which could lead to legal ramifications for the employer as well as possibly being very traumatic for the employee in question.

An example would be if the employee is a member of a protected group under the ‘Employment Equality Act 2015’ [11] and they do not wish to disclose this information. If there is a data breach at the company due to negligence on the side of the employer in not properly making sure the data is secured they would be liable to a lawsuit from any affected parties.

Even though an employee is being paid, unless specified in their contract, the employer has no right to any non-work related personal information that the employee does not wish to disclose.

One of the most famous examples of an individual's data being breached was in the case of murdered teenager Milly Dowler whose voicemail was hacked by journalists from News of The World, giving false hope to her family that she was still alive as they could see her voicemail had been recently listened to [12]. In the end the ensuing legal fallout lead to the closure of the News of The World and a lawsuit worth over £100m amid claims of other phone hacking incidents by journalists working for the News of The World.

It is evident that while recent legislation has helped give a solid framework to what proper data protection should be, with the global nature of data and the internet and the fact that not every country has proper data protection in place we definitely need to move away from our current unilateral approach and encourage all countries to have a global data protection standard to minimise data mismanagement and breaches by preventing them from bypassing laws by operating from a country with little to no data protection legislation.

Conclusion

To summarise, this report has outlined the key attributes of how to measure the software engineering process. If the correct data is measured, an efficient and reliable platform chosen and algorithms that employ computational intelligence used; I believe you will be able to effectively measure the software engineering process allowing you to assess the efficiency and productivity of software engineers. This method should only become more effective as computational algorithms become more readily optimised and available.

With regards to the ethical implications of companies possessing vast quantities of both their consumers and employee's data, it truly is astonishing with how much of your personal information is in the hands of such a small number of private companies and while Ireland does have some reassuring data protection legislation there is definite need for a global standard for data protection as laws can be skirted too easily be simply having their company and data stored in a different country.

References

1. Antoine de Saint-Exupery, 1998. *Volume 3, Number 3 - 2 March 1998*. [online] Available at: http://www.firstmonday.org/ojs/index.php/fm/issue/view/90
2. Watts S. Humphrey. 1995. A Discipline for Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
3. Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
4. Yuan, Xiaohong & Vega, Percy & Yu, Huiming & Li, Yaohang. (2005). A Personal Software Process Tool for Eclipse Environment [online] Available at: https://www.researchgate.net/figure/221610663_fig4_Figure-5-Defect-Recording-Log
5. Csdl.ics.hawaii.edu. (2017). LEAP | Collaborative Software Development Laboratory. [online] Available at: http://csdl.ics.hawaii.edu/research/leap/
6. Philip M. Johnson, "Searching under the Streetlight for Useful Software Analytics", IEEE Software vol. 30, 2013
7. Merriam-webster.com. (2017). Definition of ARTIFICIAL INTELLIGENCE. [online] Available at: https://www.merriam-webster.com/dictionary/artificial%20intelligence
8. Rachel David. (2015). Can robots truly be creative and use their imagination?. [online] Available at: https://www.theguardian.com/technology/2015/oct/10/can-robots-be-creative
9. Ukci.cs.manchester.ac.uk. (2017). UKCI 2011 - Introduction: What is Computational Intelligence. [online] Available at: http://ukci.cs.manchester.ac.uk/intro.html
10. The Data Governance Institute. (2013). Privacy and Security Quotes - The Data Governance Institute. [online] Available at: http://www.datagovernance.com/quotes/privacy-security-quotes/
11. Irishstatutebook.ie. (2017). Employment Equality Act, 1998. [online] Available at: http://www.irishstatutebook.ie/eli/1998/act/21/enacted/en/html
12. BBC News. (2017). Phone-hacking trial explained. [online] Available at: http://www.bbc.com/news/uk-24894403