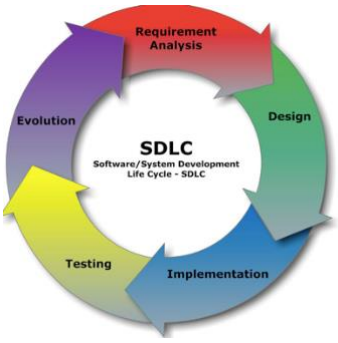
MEASURING SOFTWARE ENGINEERING REPORT

INTRODUCTION

This report will consider how software engineering processes can be measured and quantified. It will include platforms that can perform this measurement, alongside algorithmic approaches available. Finally, it will delve into the question of ethics surrounding this space.

SOFTWARE ENGINEERING PROCESSES

These processes involves dividing software production into individual phases, and steps which will in turn improve projects as a whole. By creating a more rigid structure in approaching the software's design, engineers can produce better quality, more cohesive developments. This in turn will improve the overall design, structure, project management and final product of the software. Any basic software development life cycle must include the following:



https://en.wikibooks.org/wiki/

Specification:

> This involves defining the main requirements of the software and how it should function whilst considering the constraints surrounding this.

Design and Implementation:

> Planning the design and layout of the software and how one should approach creating such software and thereby implementing it.

Verification and Testing:

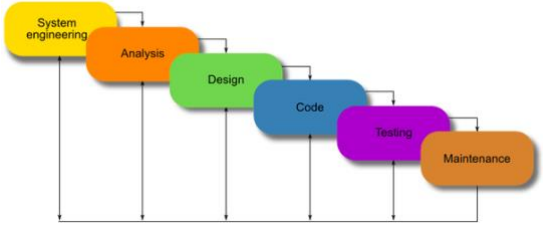
> The software must meet the earlier specified requirements and perform in a way that meets the client's needs.

Maintenance:

> This involves enabling the evolution of the software created, and how it can be modified to meet needs in the future and respond to requirement changes. (Gabry, 2017)

SOFTWARE PROCESS MODELS

These models are representations of the software processes referred to above. Different models enable engineers to tackle projects from different perspectives. Examples include:



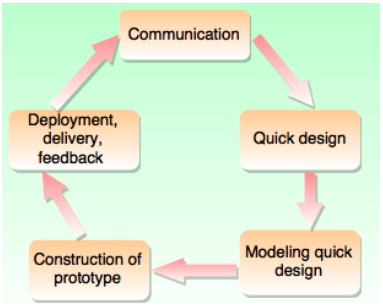
Waterfall Model

This is a sequential approach, in which the phases are completed in a linear manner. (Gabry, 2017)

It is driven by planning the project from start to finish, incrementally scheduling all elements before commencing the project. The advantages include: robust structure and organisation, allowing for early design changes, good for deadline focused development and adaptable for teams in which members come and go (Powell-Morse, 2016). However, some disadvantages include the inflexible design, inability to adapt to client feedback occurring in the latter stages of the project and that testing and error checking occurs later – almost as an afterthought. This must be combatted with an effective error measurement tool.

Prototyping

This process enables checking the feasibility of customer requirements of a design. This is useful when either the developer or customer are unsure of some of the projects elements. (Gabry, 2017) A software prototype can be used in creating the software requirements or in designing the system. It is advantageous as clientele are actively involved in the development, errors can be detected earlier and user feedback enables better solutions. However, it can lead to an implement and repair approach and it may increase the projects complexity and scope outside of original plans (Kumar, 2013). This model is best used when the software developed will interact with a large number of users.



https://www.tutorialride.com/software-te 1

These are just two of countless models available. It is important to choose a model that best suits the types of systems you are developing, which will in turn ease the production process and increase the quality of the final software produced.

COLLECTING SOFTWARE METRICS

Upon establishing a process and model for your software development, the next issue that comes to light is how does one measure the quality and progress of this project? How do you quantify something as complex as software engineering? These metrics are essential for project management as they allow a quick way to measure performance, set goals and track progress (Stackify, 2017). However, some development teams place more emphasis on actually developing the software rather than measuring its progress. Due to this, metrics must be simple to gather. This data must be easily computable, use consistent units of measurement, be adaptable, cost effective and independent of programming languages (Stackify, 2017). Therefore, it is intrinsic that software developers utilise platforms that automatically measure data and track metrics. However, there is a fine line between collecting relevant data and having a data overload.

You can divide development metrics into four key areas: speed, accuracy, quality and joy. Examples include:

Cycle Time

What: This metric analyses the duration of time spent on a particular issue. This is beneficial as it measures the speed it takes for an element of a project to come to fruition. This can be measured by median type analysed by either type or status.

How: You can utilise an issue management system such as JIRA or Pivotal Tracker. Otherwise you would have to manually record cycle times which would be a lengthy and inefficient process. (NotionData, 2017)

Why: This metric can enable you to understand your teams pace in a different way than sole velocity. This measurement allows you to see how long it takes each element of a project to flow through your team, process and model and can help you identify any potential issues arising in any of these areas. This will enable you to alter your process according to the issues or set expectations and goals for your team.

Velocity

What: This is a measure of the number of elements that have been completed within a specified period which are ready for testing. It is usually referred to as average velocity and is calculated across several intervals.

How: It is usually measured in a metric called 'story-points'. It refers to the size of 'value added work delivered' over a particular period (NotionData, 2017). After measuring this over a number of period you can compute your average velocity per week, for example.

Why: This is the most popular measure of a team's pace in the workplace. It is used to create a basic structure of how quickly you should be reaching expectations. It enables you to forecast when you will finish the project given a backlog, if the team is falling behind, or if the team is ahead of schedule.

Throughput

What: This measures the total output of work by a team. It refers to the number of tasks completed, elements achieved and bugs tackled that are ready for testing or shipping within a designated period.

How: Throughput is often measured in 'tickets'. This is because it could include any feature, bug or task. It is essential to dive deeper and analyse ticket types. This information should align with the projects goals.

Why: This measurement analyses goal alignment. For example, if your current area of focus is that of bugs, you should see a proportionately high level of bug tickets for that period. This metric can aid you with planning as you can compare average throughput to the current workload, enabling you to understand if the team is being overworked or actually underperforming. (NotionData, 2017)

Open Pull Requests

What: A pull request is a method used by developers to notify their team to review the changes they have made to a repository. This metric computes just how many pull requests there are in a particular repository that are yet to be closed. Until someone has reviewed the changes, made adjustments or committed additional code, the pull requests status will remain open.

How: This data can be pulled from platforms such as Github or Bitbucket. It is essential to track pull requests over a period and analyse any trends arising.

Why: The number of open pull requests will influence your process, workflow and overall throughput. In analysing this metric, you can speed up your overall completion time.

Code Churn

What: This metric tells you the rate at which your code evolves. It measures how many tickets have moved backwards in progress during a given period.

How: It can be measured from project tracking tools such as JIRA or Pivotal Tracker.

Why: It allows you to visualise your development process and how effective it is. It enables you to analyse the quality of your code before delivery. For example, if your code churn increases as the deadline approaches it shows that your code volatility is increasing which is unfavourable. (CodeScene, 2015)

Other valuable metrics include: burndown, backlog health, number of customer support issues, work in progress and iteration flow, mean time to repair, application crash rate, active days, defect removal efficiency.

Soft Data Approach

Above I have discussed a number of technical metrics to measure software engineering data. However, there is a lot more to software engineering than just writing code and bringing software to completion. This includes vital skills such as good communication, teamwork, initiative, proactivity, interpersonal skills and highly analytical. All of these skills cannot be measured by hard data alone. It is not sufficient to solely judge software developers on the analysis of their hard data. This alone would not be a suitable reason for awarding a promotion. However, good hard data analytics combined with a soft data approach and an element of human judgement would be a complete and appropriate manner in weighing up an engineer's entire job performance. The area of soft data is a tricky one as you must ensure you have the developers' permission and are adhering to all data protection laws when gathering information on their soft skills. Areas of measure include: communication, email correspondence, meetings attended, agility metrics and pull requests.

Agility Metrics

This metric would measure how quickly teams adapt and respond to change, in the form of decision making, planning and organising. Agile metrics analyse how a team progress and deal with setbacks throughout the development cycle. An agile team needs to be able to nimbly adjust to changing customer wants and needs. Therefore, it is vital that we measure this ability. Extreme Programming is a software development discipline that enables people to produce higher quality software more productively. This methodology values communication, listening, feedback, respect and courage. These principles are based on the values that foster decisions in a system development project.

UTILISING SOFTWARE METRICS

How this data is collected is not half as important as deciding how to utilise the data in an advantageous manner.

Link Metrics to Goals

This involves the software development team correlating key metrics with specific targets. These could include minimising the number of lines of code, reducing errors and bugs, increasing the number of software iterations or speeding up the completion of tasks (Stackify, 2017). This helps to keep software teams on track while optimising the software they are producing thereby enhancing user experience.

Track Trends, Not Numbers

Metrics are favoured as they display often complex processes in simple figures. Once a metric target is met, it is easy to declare a win for the team. However, simple metric targets do not offer much information on how these measurements are trending. (Stackify, 2017) Single metrics are insignificant in comparison

to the trend they are a part of. Analysis of how these metrics are trending can offer greater insight into the health of the team, their progress and their process. A graph showing the trend of progress towards reaching a goal or project completion is much more encouraging and insightful than analysing a single measurement at one point in time.

Other important areas to consider include: only using software metrics that lead to considerable change and progress, and setting shorter measurement period for metrics to gain more insights.

Benefits of Utilising Software Engineering Metrics

It is vital that firms conduct the analyses outlined above and utilise them in the most beneficial manner possible. In doing so, there are a number of commercial benefits. It enables firms to increase their return on investment and reduce costs. It also gives them an idea of what type of developer their company values most. It enables identification of areas or processes that need to be improved, while improving productivity and debugging performance. It facilitates teams to set accurate goals and identify processes and techniques to achieve them, whilst measuring their performance. All in all, metrics harvest a culture of continuous improvement and will ultimately produce better software systems for clients, and better developers, and enhanced software engineering processes.

HISTORY OF SOFTWARE MEASUREMENT

PSP

Obtaining all this data is all well and good, however if it is not collected in a productive manner it can take away from the timeliness and efficiency of a project. The way in which productivity and metrics have evolved over time is drastic. Over 20 years ago, the novel “A Discipline for Software Engineering” was published, written by Watts Humphrey. This was a monumental text as it adapted organisation-wide software measurement and analysis methods to the work of an individual developer or team. This adaptation is called the Personal Software Process (PSP). This process strived to automate the collection of human data, in turn, eliminating human error. This process involved simple data collection using spreadsheets, manual data collection and analysis. However, this presented a huge task with a large time and effort cost. Sadly, at the time there were no computational platforms that could automate this process. Humphrey expressed “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”. This model is utilised by every software engineering firm and team even to this day.

LEAP

After PSP radicalised the software engineering field and many people put this process into action, cracks began to show. Due to the manual nature of the data collection there was significant room for human error and data quality issues. To combat this issue LEAP toolkit was developed. LEAP stands for Lightweight, Empirical, Anti- measurement dysfunction and Portable software process measurement. It is a method of analysing and automating data analysis, tackling the issues of human error. Although data still had to be manually entered, this toolkit automated the PSP computations and provided additional analysis PSP could not. This data was portable and anonymous linking to a repository of personal data. This allowed developers to carry it with them as they moved from project to project, team to team or organisation to organisation. Nonetheless, the LEAP process while improving on PSP, was not perfect.

Hackystat

A more advanced data collection tool named Hackystat was in turn developed at the University of Hawaii. This process has four main design features:

Client and server side data collection: This enables individual developer's activities to be tracked at their desk but also on server- or cloud based activities.

Unobtrusive data collection: This involves eliminating the frustration of having to interrupt work in order to record what one was working on. Hackystat enables users to work freely without involving themselves in the tracking of their work and their data being collected.

Fine grained data collection: This refers to client-side tools enabling data collection at a frequent basis whilst collecting data on every aspect of a developers work. Also whilst capable of updating at intervals as current as second by second.

Personal and group based development: It does not just collect the developer's personal data, but also analyses the projects they work on and the teams they participate in.

COMPUTATIONAL PLATFORMS

These processes have led to a whole new industry of data analytics companies which offer software engineering measurement facilities and services. This allows software engineers to analyse their performance in a user friendly, hands-off, graphical approach, in which they do not have to worry about the lengthy process of collecting and compiling this data.

Codebeat

This data analytics firm covers most major technologies and programming languages. It is a simple, open source code review tool. It has grown a lot in recent years and takes user feedback immensely into consideration when implementing new features. Advantages of this platform include: good customer service, small but well documented API which aids management and allows metrics customisation. However, some drawbacks include not having a security issues measure in place and no CSS/SCSS analysis. (Ozimek, 2017) The table below shows how the evolution of code beat implements the data collection and analyses methods described previously.

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

Codacy

This platform, founded in 2012, supports over 65,000 developers and has a simple and user-friendly interface. This platform is utilised by major players such as Paypal and Adobe. It also allows you to define your own patterns and automates to be checked automatically. Advantages include allowing you to define issue based goals, allowing security issue checks, docker analysis, commit browsing, incredible flexibility, time estimation for fixing each issue and documents issues clearly with examples to explain each case. Some cons include some incomplete documentation and no issues searching. (Ozimek, 2017).

CodeClimate

This is a very well established and stable computational solution. It is used by over 100,000 projects, analysing over 2 billion lines of code on a daily basis! It performs automatic Git updates meaning there is nothing to install. They also provide instant notifications when any important issue or problem arises via email. They cover a huge number of languages and frameworks, are trusted by big players, very stable, have a good user interface, provide trend charts and have a well-maintained test coverage feature. This being said it is one of the most expensive tools on the market, has an unpredictable API, does not have detailed issue descriptions and doesn't not have a facility for issue searching.

Competitors include: Gitcop, Gitcolony, Semmle, Teamscale, Black Duck, Codebrag and Scrutiniser.

ALGORITHMIC APPROACHES

There are many algorithmic approaches available in analysing software engineering metrics. As I study Management Science and Information Systems, rather than Computer Science, a lot more of our focus is on statistics, probability and data analysis. Due to this the algorithmic approaches, I will discuss will be those which are familiar to me and ones which I have used in passed data analytics assignments and modules. Machine learning is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to learn from data, without being explicitly programmed. This is an approach which I am extremely familiar with and of which there are three different types:

Supervised Learning

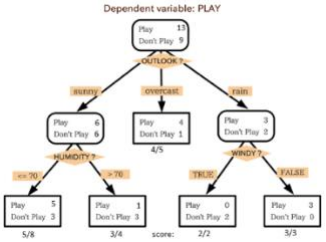
An overwhelming majority of machine learning when in practice, utilises supervised learning. This algorithm consists of a dependent variable which is predicted from a given set of predictors or independent variables. Using these variables, we can generate a function that maps inputs to desired outputs. This process is repeated and keeps learning with samples of increasing sizes until a desired level of accuracy and reliability is obtained.

Examples of these algorithms include: Linear Regression, Decision Tree Analysis, K Nearest Neighbour and Logistic Regression. The algorithm I will discuss and use most often is decision tree analysis.

Decision Tree Analysis:

This approach is best suited to classification problems. It works for both categorical and continuous dependent variables. (Ray, 2017) It uses a tree-like structure to display a number of possible decisions and

their outcomes in a graphical manner. It allows you to approach the problem in a simple and systematic manner to arrive at a logical conclusion.

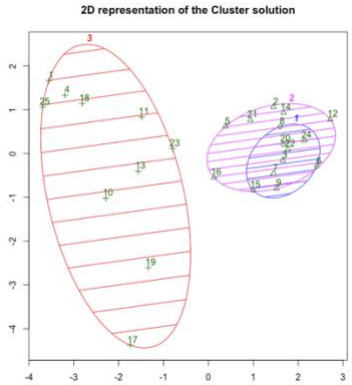


Unsupervised Learning

In this style of approach, we do not have any outcome variable to forecast. We solely have input variables. The aim is to model the underlying structure or distribution of the data. It is a much more ambiguous and unstructured approach then supervised learning as there is no specific outcome, just a number of conclusions that can be drawn from identifying the data's structure. It is used for dividing a population into different groups and types. Examples include: Principal Components Analysis, Hierarchical Clustering and K-Means.

K-Means:

Essentially this algorithm identifies a particular number of clusters within a given data set. It is particularly useful when analysing multivariate data. It is an iterative process in which the algorithm iterates through dividing the dataset into different numbers of clusters and deciding which number of clusters captures the most variability and best fits the dataset and it's distribution. Aside is a K-Means cluster analysis I recently performed regarding protein consumption in different European countries. As you can see there are clearly three clusters of data visible giving a k value of 3, i.e. 3 clusters.

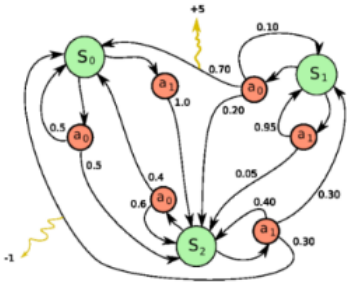


Reinforcement Learning

In this algorithm, the machine is trained in making specific decisions in a stochastic environment. It trains itself continually using trial and error calling on both past and present experience. If the problem is modelled correctly the reinforcement algorithm can converge to the global optimum with some time. This is ideal for making business decisions as this algorithm can aid you in reaching the optimal solution. An example of this is the Markov Decision Process.

Markov Decision Process:

This is a model in which a mathematical framework is used in modelling decision making when the resolutions are in part random and in part down to the decision maker. The process involves a set of possible states, actions, a reward function and a description of each actions affects within each state. Once a decision is made the new state must be defined with the relevant consequences. This process repeats until a goal objective has been reached.



Machine Learning algorithms are just some of a multitude of ways in which one can undertake software engineering measurement analysis. The approaches outlined above are accurate ways of analysing data but they are not sufficient when measuring non-code data, as it is difficult to quantify. Each of these models provide an easy solution to difficult problems. However, each come with their own level of volatility as each algorithm incorporates its own unique assumptions. However, despite this slight doubt I believe these algorithms are still advantageous as they save time and eliminate a lot of potential for human error. Personally, I believe it would be useful to use a number of different types of algorithms when approaching an analysis. This would lead to a more well-rounded and accurate interpretation whilst

using people to interpret the accuracy of the information presented by these algorithms. These algorithms clearly show huge impact since the birth of the 1995 PSP model. As these algorithms are radically developing and this is an industry of huge growth, it poses the question of just how much of software engineering processes – and all processes in general - will be automated in years to come.

ETHICS CONCERNS

According to Oxford English Dictionary, ethics is defined as *"moral principles that govern a person's behaviour or the conducting of an activity."* In recent years computers and their software systems have become an intrinsic part of everyone's daily life. This inadvertently implies that software systems are a part of every industry and used by billions of people every single day. Due to this, software engineers are under a huge amount of ethical and moral pressure.

Data Sovereignty

This refers to the concept that data obtained from data subjects in a digital form is liable to the laws of the country in which it is located. Regions such as Canada and the EU have strict data residency laws in order to protect their citizens information. 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. This causes hassle when large organisations are adopting the cloud as it requires a huge amount of data management in order for no rules or regulations to be breached. As there are turbulent times ahead, particularly for Ireland as Brexit looms, data sovereignty will be a vital area of discussion. The onward result of this being that businesses themselves end up struggling to understand the often-incongruous requirements, complex laws and regulations that exist.

Data Protection

Data analytics raises a number of ethical issues, especially as data monetization is becoming more and more lucrative. The ease at which firms can collect and share data impacts the ethical framework. As data analytics methods become more and more advanced it becomes hard for data protection law, for example to keep up. (Uria-Recio, 2018) This is why data protection is a key error of concern regarding ethics. A key data protection acts the – EU General Data Protection Regulation (GDPR) was brought in only on the 25th of May this year. This strengthens data subjects' rights and subjects all organisations within the EU to follow a lengthy list of requirements. Software engineers should note the 'privacy by design' approach (McCreanor, 2018). This means that before software is even developed the client needs to know how and if it meets the requirements of data protections. The GDPR lists 8 data subjects' rights but there are three which are most relevant to software developers:

1. *Right to Access*
   This regulation gives data subjects the authorisation to access any data that an organisation has about you. In order to facilitate this, subjects' information must be easily accessible and readily available upon request. Software developers must allow for this when meeting client requirements.

2. *Right to Rectification*
   Data subjects' are entitled to amend any inaccurate information an organisation may have about them. Organisations must respond to emendation requests within thirty days. Thereby, it will save

clients a lot of time and energy if they can allow their users to make these changes themselves, such as through a user account in which data subjects can view and update their personal information. This is something for developers to keep in mind when creating software with a large amount of user input.

3. *Right to Erasure*
   This right enables data subjects to request an organisation deletes the data they hold about them. This right comes into play if the data is not being used for the purpose it was originally collected, if the data was unlawfully processed, if it must be erased to comply with a legal obligation, if there's no legitimate interest for continuing the data processing or if the data is possessed for any service provided to a child. A data subject can withdraw their consent they previously gave to have their data collected. Due to this right software engineers must plan for this and have mechanisms in place to allow immediate deletion.

If software engineers build software with access to personal identifiable information (PII), they too become a data processor. This poses a number of requirements and regulations that must be met and adhered to. However, GDPR does not apply to data that is not related to a person, such as product or accounting data. Regardless, all systems should be GDPR compatible. (Santala, 2017)

How Much Data Collection is Too Much?

The main question the springs to mind when discussing the ethics of data collection and analystics, is how much is too much? For example, Facebooks 2.1 million users obviously had some idea that in signing up and sharing information on the social platform that this data would be analysed for ad targeting. However, recent news of Cambridge Analytica accessing data from the accounts of users who not only downloaded a personality prediction app but also their Facebook friends who had the privacy settings to allow it certainly caused heads to turn. Although only 305,000 people participated in the quiz and consented to having their data mined, their Facebook friends brought the estimated number of those affected to 87 million. However, Facebook had revoked the ability to obtain data from consenting users friends without their permission in 2015. This was certainly the worst PR crisis Facebook ever faced, with their shares dropping 18% in 10 days. Facebook quickly responded by updating their privacy policies and set out a plan to shut down enabling third party data providers to offer their advertising directly on facebook. Even though this is common practice in the industry it was done to take a hard stance on protecting data subjects' rights. Showing, just how serious of an issue data privacy is. Cambridge Analytica's ability to bypass regulations has many users considering their data and privacy in the digital space more than ever. This has led to the politicisation of privacy as consumers become more aware of how firms monetize their personal data. However, for platforms like Facebook they need advertising to sustain their business and they rely on users to trust them enough to use their data for advertising purposes, in turn fuelling their business. Meanwhile users rely on Facebook to communicate with friends and family. This co-dependent relationship causes the ethics of data collection and privacy to become blurry. Although the Cambridge Analytica story may have been a revelation to users, many marketing firms were well aware of the mal practice. For example, in the EU, the GDPR regulations previously discussed applies to organisations outside of the region if they monitor the behaviour of EU data subjects. Breach of these regulations could lead to a penalty as high as $24.6 Million dollars or 4% of global annual revenue, an even larger figure. (AMA, 2018)

CONCLUSION

To conclude, I believe the software engineering process is one of extreme complexity which poses a myriad of questions. I believe the key to successful engineering is in identifying the process and model for developing your software that is most beneficial. In having a structured plan outlined prior to developing systems alongside a number of specific goals teams can create software more efficiently. Upon beginning the development, teams must measure how they are doing in relation to the goals they previously outlined. I believe the best metrics to use are ones which contain a combination of hard and soft data approaches, as this will give a more well-rounded analysis. This will enable them to measure the quality of their system in terms of code, but also the health of their team in relation to human factors and abilities. It is essential to have these data metrics measured in an unobtrusive manner.

Firms must identify and computational platform that will work best for them. Although I have mentioned a few, such as Code Climate and Code Beat there are a huge sea of competitors in the market. It is key to identify a platform that provides measurement that will specifically benefit your firm and utilise an algorithmic approach suited to the data you wish to analyse.

However, the biggest issue I see in all of this is selecting and utilising metrics with a computational platform that extracts, collects and analyses data in an ethical manner. It is important to do your research on the regulations you must adhere to and the rights you are entitled to. As an engineer, they become data controllers the minute a user interacts with their platform, however they are also a data subject as they are having their personal metrics tracked, monitored and scrutinised. It is vital that we strike a balance between awareness of regulations, not over or under regulating and adhering and adapting to these rules. In an ever-automated world, while everything is becoming more computerised we must remember to have an air of human judgement and intuition surrounding all aspects of the development process. All aspects outlined will contribute to high performance.

BIBLIOGRAPHY

AMA, 2018. *The Murky Ethics of Data Gathering in a Post-Cambridge Analytica World.* [Online]
Available at: https://medium.com/ama-marketing-news/the-murky-ethics-of-data-gathering-in-a-post-cambridge-analytica-world-33848084bc4a
[Accessed 18 November 2018].

CodeScene, 2015. *Code Churn.* [Online]
Available at: https://codescene.io/docs/guides/technical/code-churn.html#
[Accessed 02 November 2018].

Gabry, O. E., 2017. *Software Engineering — Software Process and Software Process Models.* [Online]
Available at: https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc
[Accessed 31 October 2018].

Kumar, S., 2013. *What is Prototype model- advantages, disadvantages and when to use it?.* [Online]
Available at: http://tryqa.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/
[Accessed 31 October 2018].

McCreanor, N., 2018. *Menu 3 things software engineers need to know about the GDPR.* [Online]
Available at: https://www.itgovernance.eu/blog/en/3-things-software-engineers-need-to-know-about-the-gdpr
[Accessed 15 November 2018].

NotionData, 2017. *8 Essential Software Development Metrics for Team Productivity.* [Online]
Available at: https://blog.usenotion.com/8-essential-software-development-metrics-for-team-productivity-273737868960
[Accessed 01 November 2018].

Ozimek, Ł., 2017. *Comparison of Automated Code Review Tools: Codebeat, Codacy, Codeclimate and Scrutinizer.* [Online]
Available at: https://www.netguru.co/blog/comparison-automated-code-review-tools-codebeat-codacy-codeclimate-scrutinizer
[Accessed 08 November 2018].

Powell-Morse, A., 2016. *Waterfall Model: What Is It and When Should You Use It?.* [Online]
Available at: https://airbrake.io/blog/sdlc/waterfall-model
[Accessed 31 October 2018].

Ray, S., 2017. *Essentials of Machine Learning Algorithms (with Python and R Codes).* [Online]
Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/
[Accessed 08 November 2018].

Santala, A., 2017 . *What Should Software Engineers Know about GDPR?.* [Online]
Available at: https://www.infoq.com/articles/gdpr-for-software-devs
[Accessed 15 November 2018].

Stackify, 2017. *What Are Software Metrics and How Can You Track Them?.* [Online]
Available at: https://stackify.com/track-software-metrics/
[Accessed 31 October 2018].

Uria-Recio, P., 2018. *5 Principles for Big Data Ethics.* [Online]
Available at: https://towardsdatascience.com/5-principles-for-big-data-ethics-b5df1d105cd3
[Accessed 10 November 2018].