MEASURING SOFTWARE ENGINEERING

FERGAL MURRAY – 18323603 – FEMURRAY@TCD.IE

1. ABSTRACT

This report considers the ways in which the software engineering process can be measured and assessed. It will provide an overview in terms of measurable data, the computational platforms and algorithmic approaches available to perform this work and the ethical concerns of such analysis.

2. INTRODUCTION

Before beginning this report on measuring and analysing software engineering, it is important to define what software engineering is. This term refers to the practice of applying engineering principles to software development and encompasses all elements of developing software including designing, writing, testing and maintaining software products.

A few people are credited with coining the term. These people include Fritz Bauer who held a NATO conference entitled “Software Engineering” in 1968 (Randell, 2019) and Margaret Hamilton, who used the term during the NASA Apollo missions to bring legitimacy to the practice. (Cameron, 2018)

Software engineering was born in the 1960s, in response to the extreme advancements in computer technology and its capability in the 50s and early 60s. These advancements were not accompanied by research into how to best utilise these capabilities, resulting in many failures to meet schedules and remain in budget. This issue was called the “software crisis” and was due to the individualist nature of the practice in its nascent years.

Since the 1960s, it has become apparent that larger scale projects demand a team that can work fluidly and flexibly. Moreover, larger projects require more programmers which in turn results in a higher need for coordination and collaboration in the workplace. Consequentially, it is very important for a company to be able to track and measure the productivity of its workers.

3. DATA MEASUREMENT

Software engineering is an unusual industry in that it can be difficult to objectively determine how well an engineer is performing. In most industries there are tangible aspects of a business one can use to measure performance, such as profit and time taken. In a business in manufacturing or retail one can measure products and sales respectively.

In contrast, there is no one clear and consistent metric to measure the productivity of an engineer. We can use metrics like time taken and profit, as in other industries, but these will not show us the whole picture of an engineer's productivity. Additionally, measuring software based on the sheer volume of code written, the numbers of errors incurred or bugs fixes is not effective. Engineers could write simple code that does not solve complex problems to avoid errors or write redundant code to increase their word count. We need other measures to get a clearer picture.

Kan (2003) said it was important "collected data can provide useful information for project, process, and quality management and, at the same time, that the data collection process will not be a burden on development teams", meaning it should not use up resources such as time and money.

Considering this, I will now examine some of the most used metrics to collect data on the productivity of software engineers and the quality of their work.

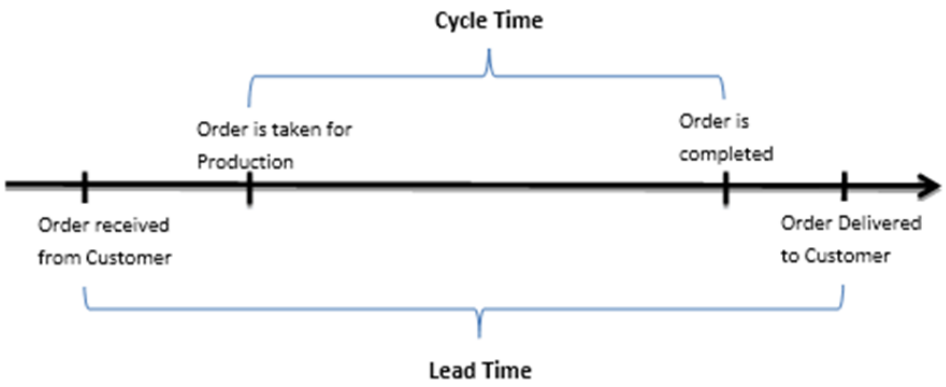
LEAD TIME AND CYCLE TIME

Lead time and cycle time are "agile metrics", meaning they help a software team monitor how productive a team is across phases of developing software. (Plutora, 2020)

Lead time is defined as the latency between the initiation and completion of a process. In the case of software engineering, this can refer to the time elapsed between the conception of an idea to its delivery. It's important to note lead time includes the time before work begins on the project.

Lead time is useful because it gives an engineer an idea of what stages of development take the longest, and steps can be taken to improve efficiency at those stages. It can also be used to predict future delivery times, allowing for analysis on customer satisfaction and the cost of a project.

Cycle time is like lead time, the difference being cycle time refers to the time elapsed between the beginning of work on a project and its completion. The difference can be seen in the accompanying diagram.



TECHNICAL DEBT

Technical debt refers to the concept where additional development work emerges when short-term solutions are implemented to solve an issue instead of the best overall solution. A developer will have to rewrite the code at some point, possibly leading to delays. This metaphor was coined by Ward Cunningham in 1992:

"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with refactoring. The danger occurs when the debt is not repaid. Every minute spent on code that is not quite right for the programming task of the moment counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unfactored3 implementation, object-oriented or otherwise."

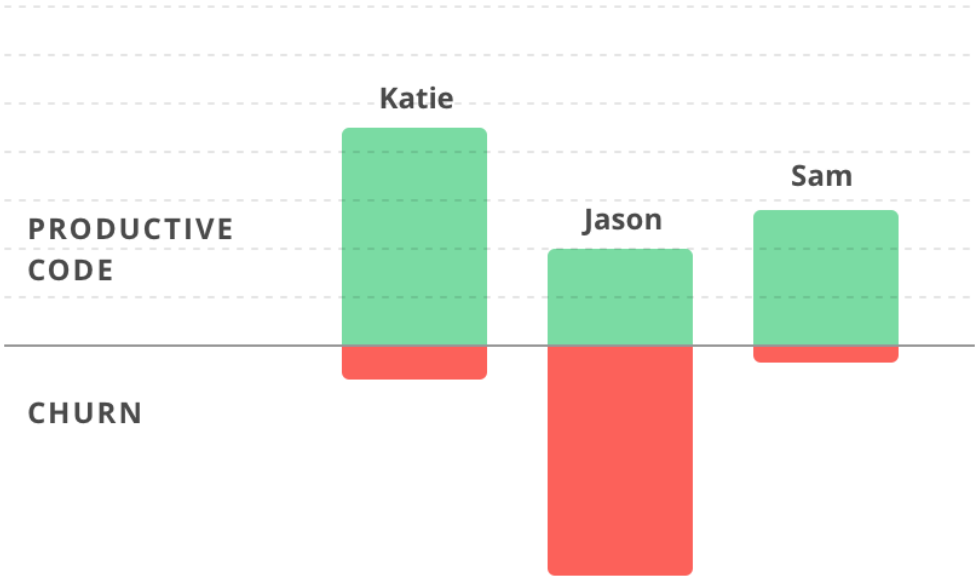
Just like financial debt, technical debt can be detrimental to a business in the long run if it is allowed to accumulate. Not addressing debts soon could lead to more debt building up due to the difficulty in editing the code. It is inevitable that some technical debt will appear in a project, but efforts can be made to reduce it.

CODE CHURN

Code churn refers to code that is rewritten or edited shortly after being written. It represents an engineer reworking a section of the code they have written. While a "normal" amount of code churn is inevitable and expected, spikes in code churn by an engineer on a feature can indicate there is a problem, as the engineer is spending a lot of time on the feature while making little progress. This problem could be isolated to the feature, relevant to the whole project, or to the engineer working on the feature. It could be a symptom of an engineer having an off-day or being a perfectionist, or of an indecisive client changing their mind often. In any case, it is good for a manager to have an idea of what "normal" code churn looks like so problems can be promptly identified and solved.

High code churn rates are expected at the beginning of a project as the project finds its feet and becomes more stable. It is expected that the rate of churn would decrease as a project continues towards completion.

If we say the below diagram refers to the code submitted by 3 engineers in a day, we can see that while Jason has produced more code overall than Katie and Sam, he has produced less productive code than either of his co-workers. Most of his code has been chode churn. (Pluralsight, 2020)



SOURCE LINES OF CODE

Source lines of code, or SLOC, is a metric used to measure the size and complexity of a project. It is calculated by simply counting the lines of code in the project. It is often used to predict the effort and time required to develop a program and is also used to estimate the productivity of the project. There are 2 types, physical and logical. Physical SLOC is the complete sum of lines in a project, comment lines included, whereas logical SLOC attempts to measure the number of statements in the code. Naturally, physical SLOC is easier to calculate than logical SLOC.

SLOC is largely seen as a metric that is difficult to judge, as it will vary from organisation to organisation based on the practice of each. For example, one organisation might require a certain format that will lead to the SLOC to be quite long. Additionally, promoting a large SLOC promotes the idea of complex and confusing code, which is not ideal. It can lead to code being hard to understand by those who read it. Companies should instead be promoting neat and concise code.

LIMITATIONS OF METRICS

As with all figures and statistics, it can be very easy for a manage to look at a metric and jump to a conclusion. Therefore, it is very important for the manager to communicate with their team and investigate out the cause for a negative-tending metric. A seemingly alarming result could be an easily explained and uncontrollable lapse in measuring the process. As always, its important to take measurements in context.

4. COMPUTATIONAL PLATFORMS

Once data has been gathered it must be analysed to shed light on the software engineering process. The quicker a company performs reviews its code and improves its process, the more competitive the company becomes. This has led to several computational platforms being released over the years to collect and analyse software engineering metrics, like those discussed above. I will now examine some of the more prominent platforms.

PERSONAL SOFTWARE PROCESS (PSP)

The concept of PSP was introduced by Watt Humphrey in his innovative book *A Discipline for Software Engineering.* He aimed to improve the performance of engineers by improving their time and project management skills. In the book, PSP is a series of spreadsheets and forms requiring manual data collection and analysis. There is a total of 12 forms a developer must complete, which amount to 500+ values that must be manually calculated. While unwieldy and arduous work, Humphrey supported the manual nature of the process. He saw it as personal, as the developer must use their own judgement when performing the analysis. He also saw it as flexible, allowing developers to edit the forms as needed for their individual company. However, this personal aspect is a double-edged sword; it is very vulnerable to human error. PSP was a pioneer in its field and led to the development of many other platforms. (Johnson, 2013)

LEAP TOOLKIT

PSP's shortcomings led to the creation of the Leap toolkit. It improved on PSP's data quality issues by automating data analysis. It still required manual data entry, which allowed some amount of human error to bleed in. But automating data analysis made the process much more lightweight, cutting time spent on the process. It also included some additional analytic methods over those provided by PSP, such as various forms of regression. It was also portable, creating a repository of personal process data that a developer can keep as they move between projects and organizations.

However, over time, it was realised that the PSP process could not be fully automated as attempted in the Leap toolkit, it would always require manual entry and would therefore be prone to bias. It was also realised this process, like PSP, required too much input for too little output. These realisations led to the development of alternative tools, such as Hackystat. (Johnson 2013)

HACKYSTAT

Hackystat is an open source framework for collecting, analysing, and interpreting software process and product data (Hackystat, n.d.). It works by attaching software 'sensors' to development tools which sends data to the Hackystat server. Hackystat deviates from the conventional thinking of defining high-level goals and then figuring out how to perform data

collection and analyse to achieve them. The developers instead developed ways to collect software process and product data unobtrusively, and then determined what high-level goals could be supported by this data. Hackystat collects data from both client- and server-side, allowing for more extensive analysis. It also collects data subtly and automatically, with developers unaware what data is being collected and when. It also allows for data to be collected very finely; data can be collected on a minute-to minute or second-to-second basis. It is also easily integrated with Git.

However, some developers expressed discomfort with some of Hackystat's features. Many were uncomfortable with the widespread availability of their work and others were uncomfortable with the idea of their data being collected without their knowledge. While Hackystat succeeded where Leap failed in terms of automatically collecting data and not interrupting developers while doing so, but fell in face of engineers' dissatisfaction. (Johnson, 2013)

CODACY

Codacy was founded by Portuguese programmer Jaime Jorge in 2012. It was founded with the simple mission statement to "help developers ship better code, faster". Codacy allows developers to visualise their coding patterns and improve their practices and process. Some of the analytics it covers includes technical debt, churn and test coverage. It also allows developers to customize their analysis through multiple pre-made configurations or a custom one. Codacy is known for its clear interface and support for 30+ languages including Java, JavaScript, Python, C, Ruby. It also supports GitHub integration. (Codacy, n.d)

OTHER PLATFORMS

There are a multitude of alternative data collection and analysis platforms available to the modern software engineer, including and not limited to: Code Climate, Codebeat, SonarQube, Waydev and GitPrime. The sheer number of platforms is indicative of how important measuring software engineering is today.

5. ALGORITHMIC APPROACHES

In this section I will examine some of the algorithmic approaches a software developer can take to measure and assess their work in place of a platform.

Machine learning is based on pattern recognition and the theory that machines can learn from data without being explicitly programmed to perform a specific task. It is an application of artificial intelligence and is ubiquitous online today, even if the average person doesn't realise it.

For example, machine learning is used to filter your Twitter, Instagram or YouTube feed, based on the content you have interacted with in the past or on what topics are popular on a given day. The algorithm evaluates various tweets, posts or videos and scores them based on various methods, then displays them to viewers based on those scores rather than by most retweets, most likes or chronologically.

There are 3 main types of machine learning: supervised learning, unsupervised learning and reinforcement learning.

SUPERVISED LEARNING

Supervised learning refers to when you have input variables x and an output variable Y and you use an algorithm to map the function between your input and output, i.e. Y=f(x). The goal of this process is to have the algorithm approximate the function so well that it can accurately predict a new output variable Y for any new input data x.

They are called supervised methods because a "training" data set is used to "teach" the algorithm. The algorithm learns by comparing its predicted output to the true intended output and modifies itself according to its mistakes, the continues to make predictions using further pieces of data. The training process is ended once the algorithm has reached a satisfactory level of accuracy. (Brownlee, 2016)

Supervised models can be further divided into 2 types: classification and regression.

- **Classification:** where the output variable is categorical, e.g. red / blue or disease / no disease
- **Regression:** where the output variable is a real continues value, measured in e.g. dollars or weight

Examples of supervised learning algorithms include K-nearest neighbours, decision tree analysis, regression and linear discriminant regression (Brownlee, 2016). 2 of these are explained in greater detail below.

EXAMPLE: K-NEAREST NEIGHBOURS

The K-nearest neighbours algorithm is a non-parametric method of assigning nodes of data to a group i.e. the method makes no assumption on the spread of data within a class. The algorithm works by looking at the K points of known origin closest to a point of unknown origin (the input data we want to make a prediction about). The algorithm then classifies this unknown point to the group that has the most points from the designated K points. (Houlding, n.d.)

EXAMPLE: DECISION TREE ANALYSIS

This method uses a graph or model of decisions and possible outcomes, arranged in the shape of a tree. The probability of a possible outcome is considered, as is its utility and costs. The algorithm iteratively splits the data set into 2 or more groups, then splits those groups into more groups etc, until all the data is grouped based on varying factors. (Chauhan, 2020)

UNSUPERVISED LEARNING

Unsupervised learning refers to when you have input variables but no corresponding output variables. The goal of this kind of learning is to study how algorithms can create a function to describe the underlying structure in the data.

These algorithms are called unsupervised learning because there is no “teacher” in the form of a training data set, the algorithms are left to their own devices. The algorithm doesn’t accurately predict the “correct” output, it instead explores the data and describes the patterns within.

Unsupervised models can be further divided into 2 types: clustering and association.

- **Clustering:** where you discover the inherent groupings in the data, e.g. grouping animals based on features (e.g. number of legs) or customers based on purchasing patterns
- **Association:** where you discover the rules that describe a portion of your data, e.g. those people who buy X also tend to buy Y.

Examples of unsupervised learning algorithms include K-means clustering, principle component analysis and the Apriori algorithm (Brownlee, 2016). 2 of these are explained in greater detail below.

EXAMPLE: K-MEANS CLUSTERING

The aim of this method is to divide data into K distinct groups so that observations between groups are different while those within each group are similar. Doing this identifies a structure within the data set.

The algorithm works by determining the number of groups or clusters K, then randomly designating a data point to each cluster (the initial cluster center). Each other data point is then assigned to the cluster whose center it is closest to. Calculations are run to assign the new cluster center within each cluster, then each point is re-assigned to the cluster it is closest to. This process is repeated until it can not be improved any further. (Houlding, n.d.)

EXAMPLE: PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal Component Analysis is an algorithm that re-expresses data with many variables into another form using linear combinations of the original variables i.e. a dimension reduction technique. Doing this makes the data easier to explore and visualize, but can also lead to inaccuracies when calculating data. It can also be used to identify associations among variables. (Houlding, n.d.)

REINFORCEMENT LEARNING

Reinforcement involves exposing an algorithm (in this case, usually referred to as an agent) to a game-like interactive environment and having it learn through trial and error. The agent learns to achieve a goal in an uncertain environment. As it explores the environment, it takes actions, which are either rewarded or punished, depending on the programmer wishes. The agent is not instructed on which actions to take, it must decide them itself.

The agent's goal is to maximise the total reward. Through trial and error, the agent can eventually achieve its goal. It starts with seemingly random choices, but by the end uses sophisticated methods to achieve its goal.

A drawback of reinforcement learning is, the more complex the problem, the more memory it requires. Examples of reinforcement leaning include the Markov Decision Process and Monte Carlo methods.

EXAMPLE: MARKOV DECISION PROCESS

Markov Decision Processes are mathematical frameworks which describe an environment in reinforcement learning. A process consists of 4 components: states, actions, effects of actions and the immediate value of actions. The algorithm begins in a particular state and is presented with its immediate possible actions and their consequences. It then chooses an action, which defines the next state it is in and the next possible set of actions are presented to it. The algorithm learns from any poor decision it makes and does not repeat them. This process is repeated until an optimal series of actions is found.

6. ETHICAL CONCERNS

Ethics are "the discipline dealing with what is good and bad and with mortal duty and obligation" (Merriam-Webster, n.d.). Software is ubiquitous today, appearing in almost all aspects of life, including entertainment, business, education and health. As society's reliance on software continues to grow, so does the impact engineers have on the world. They have the power to improve the lives of many or cause great harm. It may seem dramatic to claim software developers have this power, but it is the truth. By designing the software millions if not billions of people use every day, software engineers have the power to collect immense amount of data and can direct society without our knowledge. Strict laws and guidelines have been put in place to prevent people from taking advantage of this.

While the data described in the data measurement section of this report that can be used to measure an engineer's performance are ethically acceptable, it is unacceptable to use other data about an engineer to make business decisions. For example, an engineer can be fired due to poor performance. But he can't be fired for reasons revealed by data gathered unlawfully, for example, information about an engineer's private life. It is imperative that data collection remain above-board and used only with the express consent of the affected persons.

Legal concerns about the use of personal data has become increasing crucial in recent years. General Data Protection Regulations, or GDPR, were introduced in the EU in 2018 to improve the protection of the privacy of EU citizens. Under these laws, personal data can only be used by a person if they give permission, and this permission can be withdrawn at any time. These regulations impose stricter laws on all organizations that hold personal data, including software development companies. Under GDPR, organisations which hope to measure their software engineering practices must do so in a legal fashion, or face being fined €20 million or 4% annual turnover, which ever is the larger sum.

A recent example of breaches in ethics when it comes to data gathering is the high-profile Facebook-Cambridge Analytica scandal in 2018. Cambridge Analytica, a British political consulting firm, harvested the data of millions of Facebook users and sold it to American political campaigns without the Facebook users' consent, to try and manipulate votes. Cases like this are detrimental to the credibility and reputation of the software development industry. As Facebook has been seemingly minimally affected by this scandal, remaining one of the biggest and ubiquitous social platforms and potentially illegally selling data to this day, it is clear there is a pressing need for stricter laws surrounding dana analysis.

Admittedly, a one-size-fits-all code of practice may be difficult to produce, as the field of software engineering is dominated and furthered by diverse thinking. However, I believe the ethical approach to data measurement needs to be a core subject in studying the profession of software engineering.

7. CONCLUSION

In summary, measuring and analysing software engineering is a complicated process, suiting a complicated profession. There are many questions surrounding this process. This report has outlined the ways software engineering was and is measured namely examining the metrics, digital tools and algorithms used to do so. The ethics of such practices was also discussed.

BIBLIOGRAPHY

Agilealliance.org. 2020. [online] Available at: <https://www.agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.pdf> [Accessed 27 November 2020].

Bhatt, S., 2020. 5 Things You Need To Know About Reinforcement Learning - Kdnuggets. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html> [Accessed 27 November 2020].

Brownlee, J., 2020. Supervised And Unsupervised Machine Learning Algorithms. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/> [Accessed 27 November 2020].

Brownlee, J., 2020. Supervised And Unsupervised Machine Learning Algorithms. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/> [Accessed 27 November 2020].

Chauhan, N., 2020. Decision Tree Algorithm, Explained - Kdnuggets. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html> [Accessed 27 November 2020].

Citizens Information. (2018, September 5). Overview of the General Data Protection Regulation (GDPR). Retrieved from Citizens Information: http://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_of_general_data_protection_regulation.html [Accessed 27 November 2020].

Clubhouse.io. 2020. [online] Available at: <https://clubhouse.io/blog/lead-time-what-is-it-and-why-should-you-care/> [Accessed 27 November 2020].

Codacy.com. 2020. About. [online] Available at: <https://www.codacy.com/about> [Accessed 27 November 2020].

Codacy.com. 2020. Codacy | The Fastest Static Analysis Tool From Setup To First Analysis. [online] Available at: <https://www.codacy.com/product> [Accessed 27 November 2020].

Computer.org. 2020. First Software Engineer | IEEE Computer Society. [online] Available at: <https://www.computer.org/publications/tech-news/events/what-to-know-about-the-scientist-who-invented-the-term-software-engineering> [Accessed 27 November 2020].

Expert.ai. 2020. What Is Machine Learning? A Definition - Expert System. [online] Available at: <https://www.expert.ai/blog/machine-learning-definition/> [Accessed 27 November 2020].

Hackystat, n.d. A framework for analysis of software development process and product data. [Online] Available at: https://hackystat.github.io/ [Accessed 27 November 2020].

Homepages.cs.ncl.ac.uk. 2020. NATO Software Engineering Conference 1968. [online] Available at: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/> [Accessed 27 November 2020].

Houlding, B., 2019. STU3011 Notes. [Online] Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides3.pdf [Accessed 27 November 2020].

Houlding, B., 2019. STU3011 Notes. [Online] Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides9.pdf [Accessed 27 November 2020].

Houlding, B., 2019. STU33011 Notes. [Online] Available at: https://www.scss.tcd.ie/~arwhite/Teaching/STU33011/STU33011-slides7.pdf [Accessed 27 November 2020].

Johnson, P. M., 2013. Searching Under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.

Lowe, S. A., 2018. 9 metrics that can make a difference to today's software development teams. [Online] Available at: <https://techbeacon.com/app-dev-testing/9-metrics-can-make-differencetodays-software-development-teams> [Accessed 27 November 2020].

Medium. 2020. A Complete Guide To Principal Component Analysis — PCA In Machine Learning. [online] Available at: <https://towardsdatascience.com/a-complete-guide-to-principal-component-analysis-pca-in-machine-learning-664f34fc3e5a> [Accessed 27 November 2020].

Nytimes.com. 2020. Cambridge Analytica And Facebook: The Scandal And The Fallout So Far (Published 2018). [online] Available at: <https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html> [Accessed 27 November 2020].

Osiński, B. and Budek, K., 2020. What Is Reinforcement Learning? The Complete Guide - Deepsense.Ai. [online] deepsense.ai. Available at: <https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/> [Accessed 27 November 2020].

Pluralsight.com. 2020. Guide | Understanding Code Churn. [online] Available at: <https://www.pluralsight.com/blog/tutorials/code-churn> [Accessed 27 November 2020].

Plutora. 2020. Agile Metrics: The 15 That Actually Matter For Success - Plutora.Com. [online] Available at: <https://www.plutora.com/blog/agile-metrics> [Accessed 27 November 2020].