MEASURING SOFTWARE ENGINEERING: A REPORT

Cian Moynagh

18323887

Table of Contents

Abstract 3

Introduction 3

Identifying Measurable Data 3

- Scrum Metrics 4
- Lean Metrics 5
- Kanban Metrics 6
- Other Common Agile Metrics 7

Available Computational Platforms to Perform this Work 9

Available Algorithmic Approaches to Perform this Work 10

Ethical Concerns Surrounding these types of Analytics 12

Conclusion 13

Bibliography 14

Abstract

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

Introduction

Improving productivity within any system is paramount to the efficiency and success of that system, with the software engineering process being no exception to that matter. Collecting metrics on various elements within the software engineering process can be invaluable to teams seeking to improve the quality of their output. However, due to the lack of physical, and often times tangible outputs produced in the software engineering process, it is an immensely complex profession to analyse when compared to other engineering and non-engineering disciplines. Often times redundant and counter-productive measurements are analysed resulting in equally redundant and counter-productive conclusions, which aid in no way to the achieving of a software engineering team's goals and objectives.

A popular example of such a redundancy is the outdated measurement of productivity via lines of code. Workers gauged by this metric are more likely to approach their work with the intention of adding more lines of code than is necessary in order to fulfill this metric, when the very act of doing so is entirely anti-proactive to the true objectives of the project. As summarised by Bill Gates; "Measuring programming progress by lines of code is like measuring aircraft building progress by weight." Although a specific example, this problem remains true for a huge variety of metrics within the measurement and analysis of the software engineering process.

Identifying Measurable Data: Agile Software Engineering Metrics

As described above, the optimisation of productivity relies on the collection and analysis of valid data which is relevant to team objectives. This realisation has led to the development of suitable metrics known as Agile Software Engineering Metrics. These metrics can be ontologically categorised in three groups: *Scrum Metrics, Lean Metrics,* and *Kanban Metrics*.

Scrum Metrics

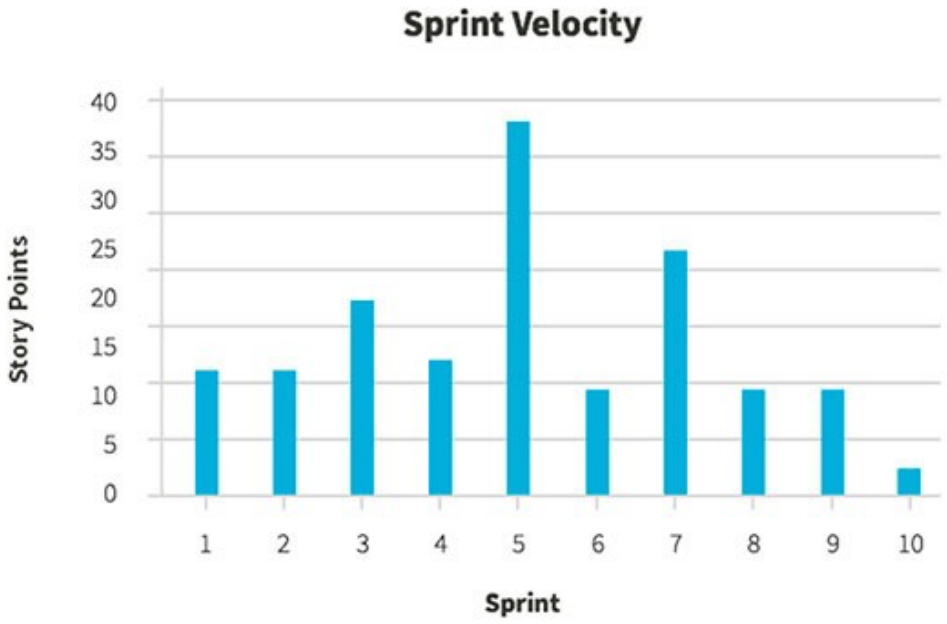
Scrum metrics are those measurements which predict the delivery of working software to clients. Popular types include *Sprint Goal Success, Team Velocity,* and *Sprint Burndown.*

Sprint Goal Success

A sprint is a short, time constrained period – usually a week long - when a team works to complete some set work. The sprint goal success is a very simple way of staying on top of that set work. The process consists of three main stages; selecting a sprint goal, running the sprint, assessing if the sprint goal has been met or not. By defining these goals and measuring the success of their implementations, a qualitative measurement of the team's efficiency can be deduced.

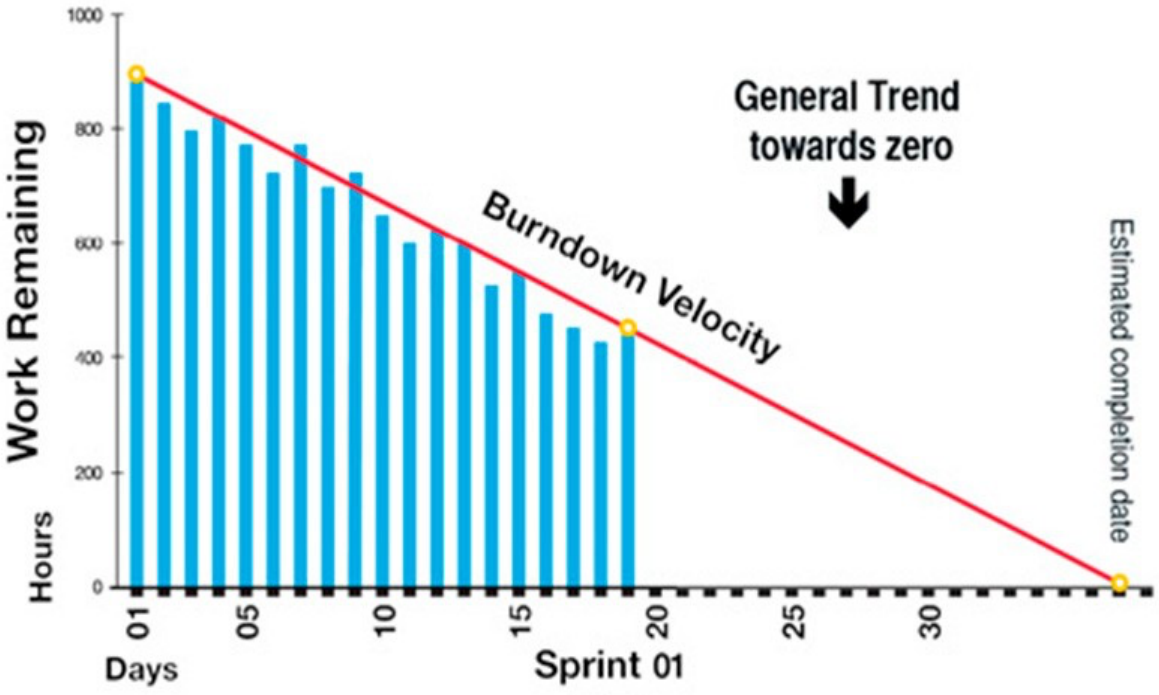
Team/Sprint Velocity

This metric is key for measuring how much work has been done in previous sprints and provides a basis for more accurately predicting how much work the team will be able to complete in future sprints. It is a measurement of how many individual items of work (e.g. story points) were accomplished on average by the team in all previous sprints. This metric can only be used as an internal method of measurement, rather than one that can be applied to compare different software teams, as the deliverables from one team to the next are likely to be very different in nature and in definition.



Sprint Burndown

A sprint burndown chart is the classic and most recognisable representation of progress within a sprint. It can be understood at a glance and shows whether a team is on schedule to successfully complete a sprint or not. The number of days/hours remaining till the end of the sprint are displayed on the X axis and the amount of work remaining on the Y axis.



Lean Metrics

Lean Metrics are those measurements which focus on the continuous flow of value from a team to their clients, and continuously identify any wasteful activity which can be eliminated from the process. Two key metrics within this mode of measurement are *Lead Time* and *Cycle Time.*

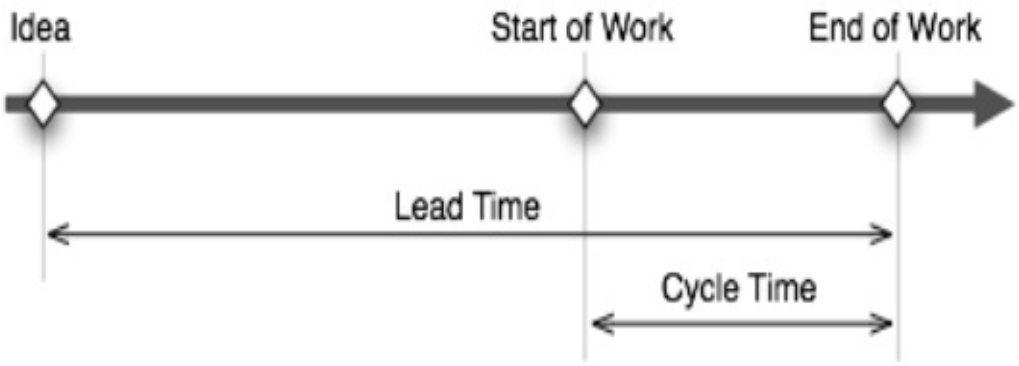
Lead Time

Lead time is one of the most important metrics for understanding the time taken for any work item to move through its production system. It is a measurement of the total time from when the work is originally required, to when that work is completed and delivered to the client. A shorter lead time is sought after in the software engineering process as it results in a more efficient development pipeline, so long as the quality of the output is not compromised by the shortening of this time. That is why measuring lead times can make teams more efficient, and

accurate with their predictions on delivery times. It is important to note that the lead time is not a measurement of the time from when the work starts to when the work is completed and delivered, but rather from time at which the work is first requested to the time of completion and delivery.

Cycle Time

Cycle time is a measurement of how quickly a team can process a piece of work at an individual stage, usually from the start of the work to the completion of the work, although in other disciplines cycle time can be applied to the design stage, marketing stage, and so on. A shorter cycle time is desirable for obvious reasons, and this metric is simple enough that clear red flags can be raised if the time is exceeding predetermined expectations. A cycle time should usually not exceed the time between sprints, and if it is doing so the work that is being committed to is not being completed on time.



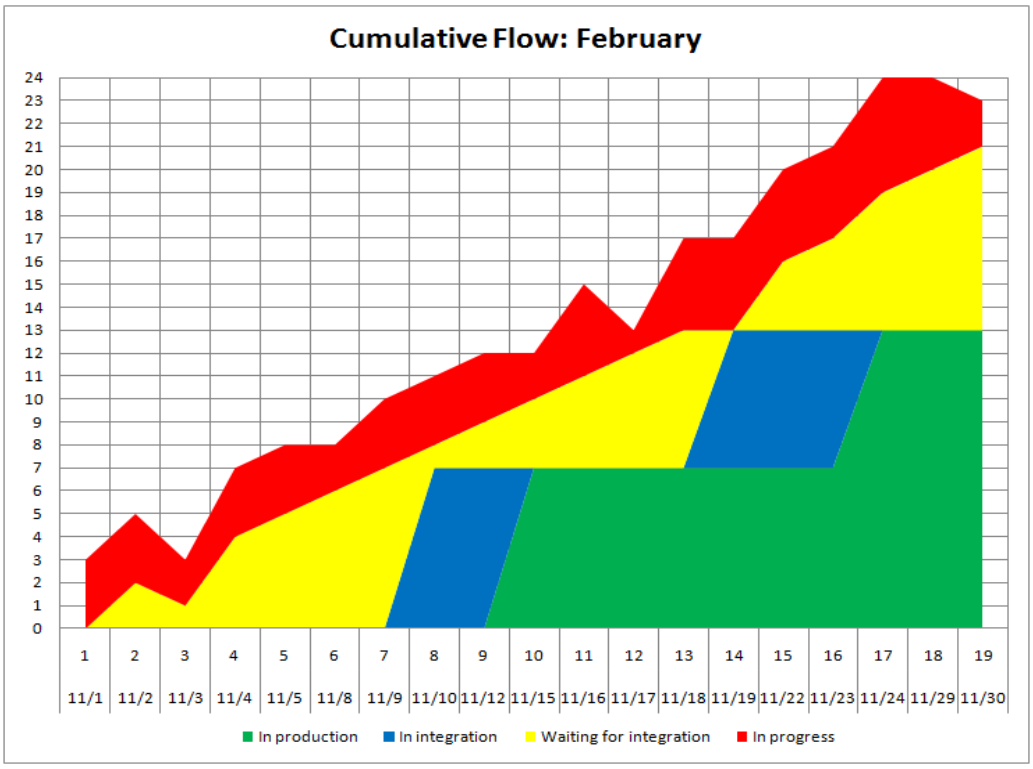
Kanban Metrics

Kanban metrics are a system of measurements which focus the prioritisation and organisation of the workflow. Perhaps the most utilised form of Kanban metric is the *Cumulative Flow Diagram.*

Cumulative Flow Diagram

The cumulative flow diagram is a visually simplistic way of modelling and describing the number of tasks underway at any one stage in the development process. It is usually updated daily and allows members of a team to visualise weather a project is currently overwhelmed with a disproportionately large number of tasks or functioning below desirable capacity. The various colour blocks represent the different tasks and the height of each block represents the amount

of overall effort being expended on that task. The value of the cumulative flow diagram comes from its simplicity and comprehensibility.



Other Common Agile Metrics

Unit Testing/Code Coverage

With this metric, unit testing of functions, branches and conditions within the code is expressed as a percentage, providing useful insight into how well developed and tested the software is. It is a metric which can be run automatically every time the code is built, making it a convenient and non-time-consuming measurement. A low code coverage from unit testing is almost guaranteed to deliver inefficient and dissatisfactory code.

All files

84% Statements 21/25 **100%** Branches 6/6 **100%** Functions 13/13 **84%** Lines 21/25

File	Statements		Branches		Functions		Lines	
src	33.33%	2/6	100%	0/0	100%	1/1	33.33%	2/6
src/component/counter	100%	5/5	100%	0/0	100%	5/5	100%	5/5
src/component/header	100%	2/2	100%	0/0	100%	1/1	100%	2/2
src/component/headline	100%	5/5	100%	2/2	100%	2/2	100%	5/5
src/component/login	100%	7/7	100%	4/4	100%	4/4	100%	7/7

Team Morale

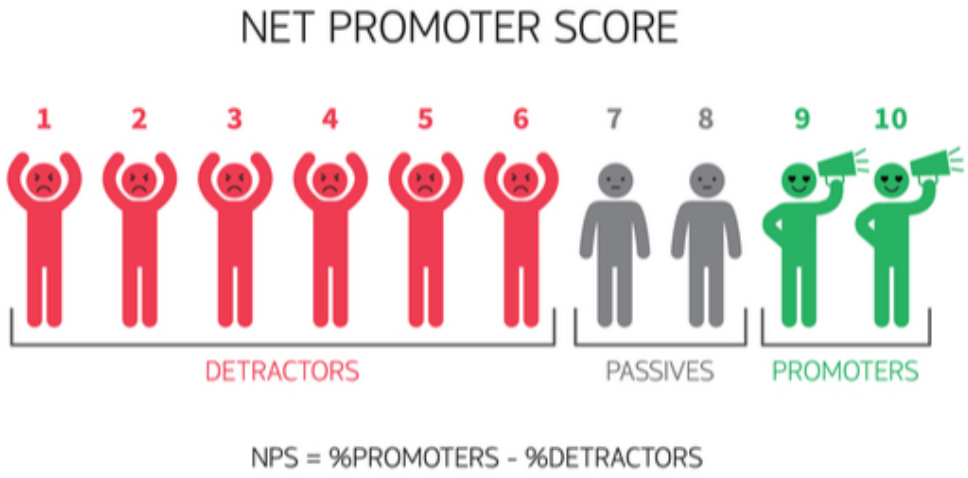
Although a tricky facet to measure, this metric is useful for evaluating the happiness and motivation of a team based off their commitment to and satisfaction with the work they are doing. Varwijs has proposed some framework around the measuring of this, consisting of 4 statements team members should answer with a rating of 1 – 5. These statements are:

- I feel fit and strong in my team
- I am proud of the work that I do for my team
- I am enthusiastic about the work that I do for my team
- I find the work that I do for my team meaningful and purposeful

Positive answers to these statements indicate a good morale within a team, which tends to be beneficial for both the developers and consumers, as well as a good indication of the well-being of a team to management.

Net Promoter Score

This metric is a formula based on user satisfaction and expressed as score between 0 – 100. The formula is deducted by subtracting the percentage of detractors (those who scored the software between 1 – 6) from the percentage of promoters (those who scored it 9 or 10). The users score is based off their likelihood to recommend the software to a friend or colleague, making the NPS a telling metric as to the likelihood of the software being a success. Unlike many other metrics mentioned above, it has a unique value through its consumer-centric nature.



Post-deployment Defects

This metric analyses the number of bugs discovered within a software system after it has been delivered to clients. The lower the number the better for obvious reasons, and a higher number indicates a lack of competence somewhere within a team.

Available Computational Platforms to Perform this Work

Due to the complexity of measuring the software engineering process, competent computational and algorithmic platforms are an essential part of executing this measurement. There is an ever-increasing number of systems with which to do this work, a few of which are detailed below.

Personal Software Process

First developed by Watts Humphry in the late 1990's, PSP was one of the first methodologies for the computation of software engineering performance. It is a structured software development process which allows engineers to manually record some simple metrics in order to better understand their performance and thereby improve upon it. The basis of PSP lies in the continued effort towards incremental improvement of process efficiency, and the list of steps taken to do this is as follows;

- Define the quality of the goal
- Measure the product quality
- Understand the production process
- Adjust process accordingly
- Implement adjusted process
- Measure the result
- Compare result to goal
- Repeat with incremental improvements

GitPrime

Founded in 2015 GitPrime has quickly become one of the foremost tools for measuring the software engineering process. With over 100,000 users worldwide the goal of GitPrime is to maximise software engineers' efficiency and productivity by providing relevant data to a project. Data points such as sprint-cycle analysis, percentage of new code added with any one commit, time spent adjusting old code, and so on, are compiled by various algorithms and condensed into useful statistics from which team leaders and members can draw conclusions, analyse what works, and address what doesn't. On top of this GitPrime assists in drawing

attention to parts of a project which might require more attention from the whole team, for example a large commit of new code which if not scrutinised adequately will likely cause setbacks further down the line.

Available Algorithmic Approaches to Perform this Work

Cyclomatic Complexity

Cyclomatic complexity is an incredibly useful metric used to indicate the complexity of a software system. Developed in 1976 by Thomas McCabe, this algorithm has lasted the test of time. The algorithm quantitively measures the number of independent paths within the software's code, and from the resulting degree deduces the program's complexity. A simple example of how the algorithm works is to think of a program containing only one IF statement. As the only possible outcomes of this program would be TRUE or FALSE, the cyclomatic complexity of this system would be 2. This algorithm can be used on specific functions and parts of the program, or on the system as a whole, and the higher the complexity is, the more difficult and expansive testing will be. A more detailed description of the algorithm is described below.

$$CYC = E - N + P$$

Where;

- *CYC = Cyclomatic Complexity*
- *E = number of edges (transfers of control)*
- *N = number of nodes (group of statements containing a single transfer of control)*
- *P = number of disconnects within the flow graph (i.e. subroutines)*

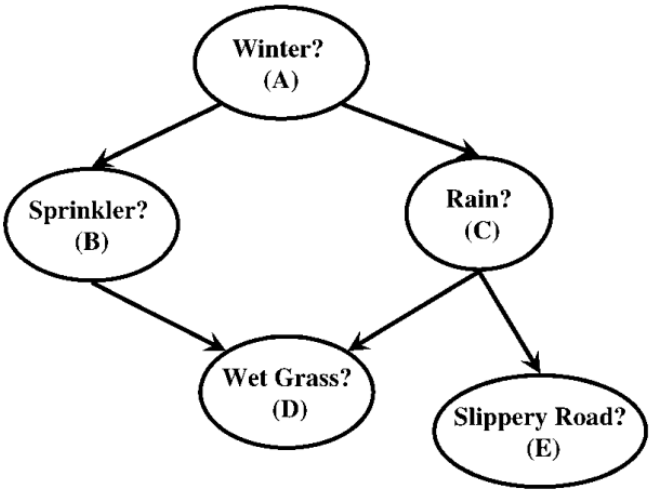
Bayesian Belief Networks

Bayesian networks provide a useful tool to visualize the probabilistic model for a domain, review relationships between variables and provide reason to causal probabilities within given scenarios. The algorithmic platforms which deduce these networks are based off Bayes' Theorem, and are comprised of two fundamental components; a qualitative component in the form of a directed acyclic graph (DAG), and a quantitative component in the form of conditional probabilities or probability tables. The benefit of these networks is not only their ability represent an exponentially sized probability distribution in a compactified and factored

manner, but also to provide this information to inference algorithms which can answer queries about these probability distributions without having to construct them explicitly.

Bayesian networks can be constructed in a multitude of ways, depending on the available information and the application at hand. Traditionally, these networks can be constructed by domain experts through the synthesizing of data from high level specifications. Alternatively, machine learning algorithms can be implemented to help fast track the process somewhat. Software such as OpenBUGS and SPSS Modeller are frequented by software teams nowadays to calculate these networks. Systematic testing and continued collection of a systems data is vital in order to successfully implement this model.

The below graph shows a Bayesian network over five propositional variables. The accompanying table depicts each node within the network and their associated probabilities given its parents.



A	Θ_A
true	0.6
false	0.4

A	B	$\Theta_{B\|A}$
true	true	0.2
true	false	0.8
false	true	0.75
false	false	0.25

A	C	$\Theta_{C\|A}$
true	true	0.8
true	false	0.2
false	true	0.1
false	false	0.9

B	C	D	$\Theta_{D\|B,C}$
true	true	true	0.95
true	true	false	0.05
true	false	true	0.9
true	false	false	0.1
false	true	true	0.8
false	true	false	0.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	0.7
true	false	0.3
false	true	0
false	false	1

Ethical Concerns Surrounding these types of Analytics

The ethical and moral concerns surrounding the monitoring of individuals within the work environment has been a subject of much debate for many years. There is a line between a business overstepping their boundaries and infringing on their employees right to privacy, and practicing reasonable monitoring with the purpose of ensuring the wellbeing of both their employees and clientele. This line, however, is a fine one, and one which is in many ways subjective, differing in strictness from one person to another. In essence, there is no universally agreed balance struck between the gathering of insightful employee data and info, and the impeding of those employees' personal rights to privacy. There is however some level of common practice employed by businesses in an effort to operate at this much sought-after healthy middle.

The most obvious malpractice when it comes to the monitoring of employees is when it is done so in secret. The monitoring of individuals without their knowledge or consent is by-and-large regarded an unethical practice, and in most places (including Ireland) an illegal one. Although any employer will have an interest in protecting their business, reputation and resources, achieving this through software or CCTV surveillance which doesn't adhere to the General Data Protection Regulation (GDPR) and the Irish Data Protection Act of 2018 can result in serious legal action for the perpetrator. Most companies in the software engineering discipline and elsewhere, will provide clarity around the types and degree of monitoring being implemented by management through monitoring policy and consent forms which explain in detail the data that will be collected, what it will be used for, and who can access it.

Amazon, a company who frequently face criticism for the treatment of their employees, are a good example of where a business can overstep their boundaries, and the monitoring becomes more of an infringement than it does a justified protocol. In September of 2020 a report by Reuters detailed how the multi-trillion dollar business uses surveillance technology on it's workers to stymie unionisation efforts. The report claims that through Amazon's use of navigation software, keystroke monitoring, item scanners, thermal cameras and location tracking wristbands, the company stations it's employees in such locations as to undermine any potential unionisation efforts that would deem the company less monetarily profitable. Regardless of the true intentions of Amazon, which could be debated, the extent of their surveillance technology no doubt raises some ethical eyebrows in it's 1984-esque prevalence.

Due to the aforementioned complexity of the software engineering profession, the way in which productivity is measured when monitoring employees is a tricky task, and one which

needs upmost transparency to minimise worker discomfort. Quantitative measures alone can render unfair depictions of the work being done, as narrowing the software development process down to a few numbers is an incredibly limited and robotic view of the process as a whole. How well an employee fits in with the company culture, adds to team morale and comes up with new and innovative ideas should all be included in the monitoring process in order to paint a more complete picture. Because of this the agile metrics mentioned above are so vital to the ethical practice of analytical monitoring in this profession.

Conclusion

There is clearly no shortage of platforms and algorithms with which to measure and assess the software engineering process. Although an unavoidably complex task, when successfully undertaken the benefits of being able to accurately compactify the complexities of a software development project into data that can be used to make sound and educated decisions are invaluable to a team seeking to maximise proficiency.

As important as these software engineering measurements are, it is equally important for management to realise that there are still limits to what these measurements can tell us on their own. A human aspect to the measuring of this discipline is an essential part of analysing the data, noting points of potential added value, and decision making going forward. There are certain aspects of any software engineering project that cannot be quantified in their entirety, and the platforms and algorithms we use to simplify these things are only tools to help the team and management, not mechanisms to find objective and inflexible truths.

Moreover, striking a balance between gaining quality insights through the pursuit of these metrics and not infringing on an individual's rights to privacy is paramount to building a culture of trust and productivity within a team. Any overstepping of this boundary will likely instil employee discomfort and a level of disconcertedness which will far outweigh the value of any data point.

Bibliography

Fenton, N. and Neil, M. (1999). Software metrics: successes, failures and new directions. Journal of Systems and Software, 47(2-3), pp.149-157.

Sealights. (2019). 10 Powerful Agile Performance Metrics - and 1 Missing Metric. [online] Available at: https://www.sealights.io/software-development-metrics/10-powerful-agile-metrics-and-1-missing-metric/

Anon, (2019). [online] Available at: https://www.atlassian.com/agile/project-management/metrics

phoenixNAP Blog. (2019). 15 KPIs & Metrics that Make the Case for DevOps. [online] Available at: https://phoenixnap.com/blog/devops-metrics-kpis

Intellectsoft Blog. (2019). 13 Agile Metrics to Improve Productivity & Software Quality. [online] Available at: https://www.intellectsoft.net/blog/agile-metrics/

Perforce Software. (2019). What Is Cyclomatic Complexity? | Perforce Software. [online] Available at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity

Intellectsoft Blog. (2019). 13 Agile Metrics to Improve Productivity & Software Quality. [online] Available at: https://www.intellectsoft.net/blog/agile-metrics/

GeeksforGeeks. (2019). Software Engineering | Halstead's Software Metrics - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics

Saedsayad.com. (2019). Bayesian Belief Network. [online] Available at: https://www.saedsayad.com/docs/Bayesian_Belief_Network.pdf

Ptgmedia.pearsoncmg.com. (2019). [online] Available at: http://ptgmedia.pearsoncmg.com/images/0131424602/samplechapter/0131424602_ch05.pdf

GitPrime Help Center. (2019). What Is GitPrime Exactly?. [online] Available at: https://help.gitprime.com/general/what-is-gitprime-exactly

AIHR Analytics. (2019). People Analytics: Ethical Considerations | AIHR Analytics. [online] Available at: https://www.analyticsinhr.com/blog/people-analytics-ethical-considerations/

Handbook of Knowledge Representation (2008) Edited by F. van Harmelen, V. Lifschitz and B. Porter