Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Software Engineering CS3012

Measuring Software Engineering Report

Contents

Introduction

This report looks at the software engineering process itself, metrics used in assessing it and the platforms that best enable us to obtain those metrics. The ethics and algorithms discussed demonstrates the complex nature of measuring and assessing the software engineering process. With such complexity, companies must take careful consideration when choosing how they measure their processes and how they gather and evaluate those metrics. This report hopes to clearly outline the reasons for choosing certain metrics and the different ways of obtaining them while also providing an insight into the difficulty of negotiating the ethical issues and algorithmic approaches associated with assessing software engineering.

Software Engineering Process

Before looking into the measures and assessments of the software engineering process we must first have an understanding of the process itself. In the field of software engineering the journey from idea conception to a final software system is known as the software development life cycle. The life cycle model according to the Institute of Electrical and Electronic Engineers is a “framework of processes and activities concerned with the life cycle, which can be organized into stages, acting as a common reference for communication and understanding”. The software engineering process refers to all the connected activities within the development life cycle (Aydan et al., 2017). The stages of the life cycle, in its most basic form, are; planning, analysis, design, implementation, testing and integration, and lastly maintenance as shown in figure 1.

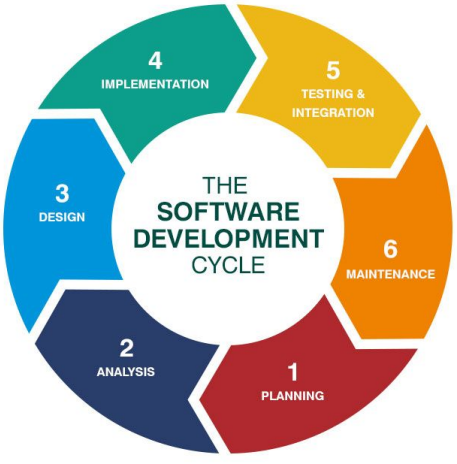
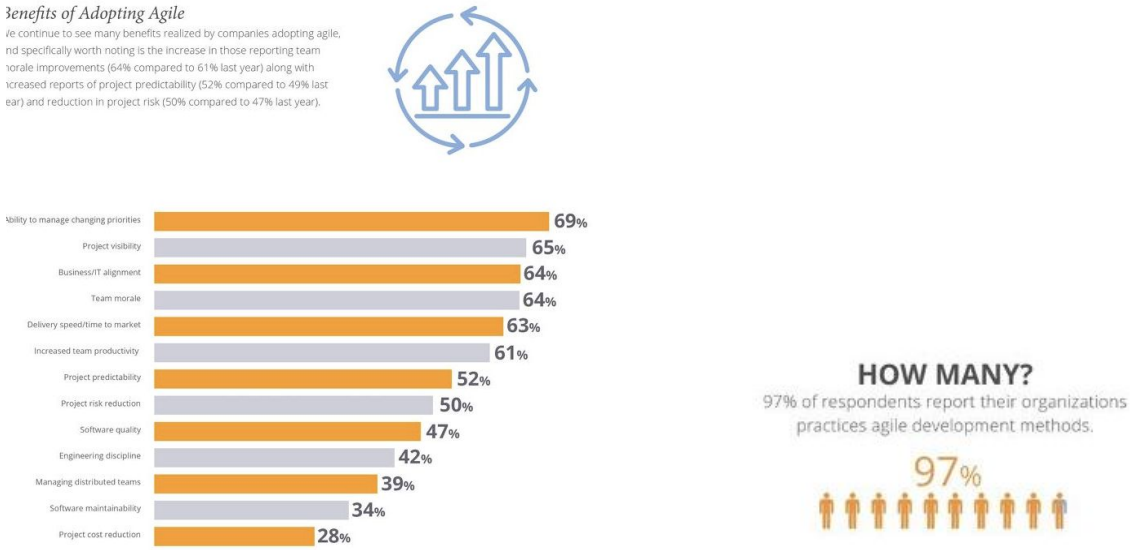


Figure 1. (Hussung, 2019)

There are many life cycle models that aim to improve the efficiency and effectiveness of software development. These models tailor the processes in various ways depending on the project. The main models include; waterfall development, spiral development, incremental development, iterative development and evolutionary development (Aydan et al., 2017). The agile development framework can be applied with a range of models but is mostly used with the evolutionary model. The agile framework is the most used and adopted methodology across every industry with 97% of organisations in VersionOne's 13th State of Agile Survey practising agile development methodologies (CollabNet VersionOne, 2019).

The model and it's methodologies can impact the measurements organisations focus on when assessing their software engineering process. Due to the fact that agile methods are so widely practised this report will focus on measurements used in agile software development teams within organisations across any industry. The benefits of using an agile framework align with the reasons a business or team measure their processes. These benefits include some of the following (CollabNet VersionOne, 2019); ability to manage changing priorities, team morale, delivery speed/ time to market, increased team productivity, project predictability, project risk reduction, software quality, software maintainability and project cost reduction.

Figure 2. (CollabNet VersionOne, 2019)



1. Measurable Data

1.1 Why we measure

There are many ways in which an engineer or an organisation can measure their software engineering process. Brooks in 1987 wrote about there being no silver bullet when it comes to improving the software development process (Brooks, 1987). It is a combination of many aspects that lead to larger improvements throughout the development of a software system. Physicist William Thompson is attributed with saying “if you cannot measure it you cannot improve it” (Zapatopi.net, 2019). With the same logic, we can see that aspects of the software engineering process must be measured if we are to improve them.

1.2 What we measure

Scrum frameworks are the most widely used agile methodologies, accounting for 72% of agile methods used in organisations (CollabNet VersionOne, 2019). The scrum itself relies on a team breaking down the software system into epics, artifacts, features and user stories. Each user story is given a value based on its complexity and effort required. The team estimates this value based on their understanding of what is involved in developing the user story. This value is called the story points. These story points are used to keep track of progress throughout a team’s sprint. After a few sprints, teams will know how many story points they can complete within a sprint. Using the story points as a comparative metric can give teams an indication of when they may have underperformed or overperformed during a sprint. This metric is used by team leads and scrum masters to measure a team’s performance during sprints. (Scrumguides.org, 2019)

Scrum teams generally track the following metrics (Sealights, 2019):

- Escaped defects - defects that were experienced by user in production
- Defect density - bugs found per lines of code
- Team velocity - the user stories completed by the team per sprint

- Sprint burndown - shows if velocity per day is on track to complete all user stories that were brought into the sprint
- Scope change - the amount of storied added to the project during the sprint

The metrics above relate are common measures scrum teams can use to evaluate themselves. In a more general agile form of the software engineering process, organisations use a vast array of metrics. The list below is a selection of metrics, excluding ones aforementioned, that demonstrates the different ways organisations, managers and teams can also assess their software engineering processes (Simplilearn.com, 2019) (Anaxi, 2019).

- Review efficiency - the number of defects caught in the review stage out of the total defects caught
- Testing efficiency - defects found in acceptance out of the total number of testing defects
- Defect removal efficiency - defects caught by the customer out of the total number of defects
- Residual defect density - defects found by the customer out of the total number of defects plus the number of defects found by the customer
- Commit or pull request risks - the amount of lines changed, surface area of the change and a comparison to previous changes can indicate the riskiness of a commit. This allows teams to track average commit risk which teams may decide to decrease by committing more frequently with simpler code changes.
- Number of bugs and bug severity - keep track of bugs and give them a severity rating. Microsoft teams have a bug cap of 5 bugs per team member. If they go over their bug cap during a sprint then the team cannot start any new user stories or items until they have brought their bug count below the bug cap (Docs.microsoft.com, 2019).

Those metrics can provide ways to measure software quality and the processes of testing and fixing bugs within the software system that is being developed. They can also help in measuring maintenance which relates to fixing issues that have arisen after deployment, i.e. defects found by the customer. Time and cost can also be factored in to measurements to give more context into the efficiency or lack of efficiency in their organisation's software development life cycle. The metrics below factor in time along with direct software metrics (Stackify, 2019) (Anaxi, 2019) (GitPrime, 2019):

- Lead time - the time it takes to develop and deploy software
- Cycle time - the time it takes to implement changes of a software system and deploy
- Active days - days in which developers committed code to the project, mainly gives insight into costs of interruptions
- Assignment scope - amount of code an engineer can support in a specified period of time
- Code churn - the lines of code a developer edits, adds or deletes during a project. Figure 3 which demonstrates how code churn can fluctuate during the life cycle. A high churn rate is considered to reduce efficiency. Figure 4 demonstrates the relationship between churn rate levels and code efficiency.

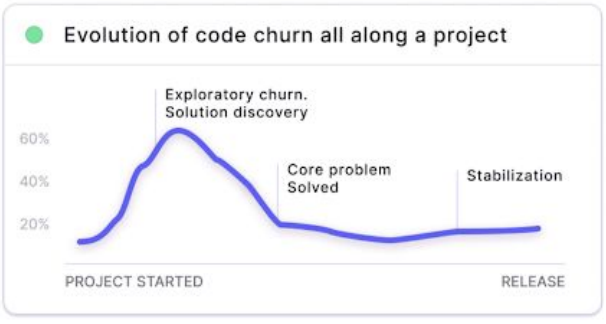


Figure 3. (Anaxi, 2019)

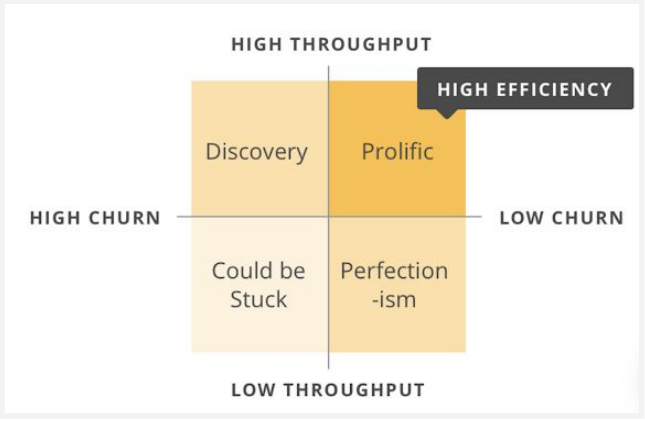


Figure 4. (GitPrime, 2019)

- Work in progress (WIP) - total number of tickets or unresolved parts of the project that the team have started but not finished. This metric prevents burnout and increases efficiency as engineers work on one thing at a time which is the ideal WIP per engineer.
- Mean time to repair - measures the time from the security issue to when its solution is put in production.
- Deployment frequency - tracks how often a team deploys to each environment and most importantly production. This emphasises the focus on smaller deployments which are generally easier to both test and release with reduced risk.
- Pull requests-related velocity - often the average pull requests merged per week which should show the throughput of the team. This metric can often show bottlenecks within the process if a teams velocity falls below the average pull requests merged.

It is clear that there are many ways to measure and assess the process. The metrics discussed already have touched on software quality, efficiency and to some extent the productivity of the software engineering process that organisations use to help them make decisions. These decisions are enhanced by the use of data but there are some rules that engineers or managers should follow when assessing these metrics. The rules are (Stackify, 2019) (Anaxi, 2019):

- Software metrics should be easy to understand, compute and be objective
- Software metrics should be relevant to business priorities or goals and cost-effective to obtain
- Software metrics should be consistent and reliable
- Software metrics should be comparative to facilitatef tracking trends and not just numbers

2. Development Analysis Platforms

2.1 What is a development analysis platform

They are systems that facilitate the management and measurement of the software engineering process. These platforms have enabled improvements in software quality, efficiency and productivity throughout every aspect of the software development lifecycle. The following development analysis tools will be discussed in this report:

1. Personal Software Process
2. GitPrime
3. HackyStat
4. Code Climate
5. Others

2.1.1 Personal Software Process

PSP is focused on manual input but is considered an influential guide and framework that was introduced in the 1990s. The framework provides engineers with a structured development process to help them measure themselves and make the engineering process more efficient and quality focused (Humphrey, 2007).

2.1.2 GitPrime

GitPrime is capable of analysing every aspect of code, reviews and collaboration within the software engineering process to help organisations and developers measure their performance (Gitprime.com, 2019). GitPrime integrates with GitHub, BitBucket, GitLab, Azure DevOps, Git, Jira, AWS, Google Cloud and Assembla (Financesonline.com, 2019). It gives managers the autonomy and accessibility to analyse any metrics during the development life cycle.

2.1.3 Hackystat

Hackystat is an “open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data” (Hackystat, 2019). Hackystat enables users to attach sensors on their development tools which are sent back to Hackystat’s web storage called the Hackystat SensorBase. This storage can be accessed by other web services to query the data to provide visualisations or insights. Integrates with GitHub, Jira, Slack and various task management tools (Financesonline.com, 2019). Hackystat is no longer under active development but is still widely used by academics, organisations and developers.

2.1.4 Code Climate

Code Climate is a collaborative tool for software development teams that helps improve software quality. Code climate can provide line-by-line test coverage reports, assess technical debt and check the style of the code in each pull request ensuring code is clear, maintainable and consistent (GitHub, 2019). Code climate also released a product called Velocity which provides real-time data and custom reports from a developer’s or a team’s GitHub repository to allow them to measure their software engineering process. Velocity’s capability is similar in many ways to the integrations and capabilities provided by GitPrime (Codeclimate.com, 2019).

2.1.5 Other platforms

The potential top ten competitors to GitPrime besides code climate are; GitLab, WayDev, Checkmarx, Smartbear, GitHub, Veracode, BitBucket, RhodeCode and Perforce (Owler, 2019). Each provide similar functionality in terms of metrics and automated assessments along with dashboard features and visualizations. Other competitors to GitPrime include Codacy, Sonarqube and Gitalytics (Waydev, 2019). Source {d} is another open source platform with strong machine learning tools for measuring software development codebases, teams and processes.

3. Algorithmic Approaches

3.1 Importance of Machine Learning

The primary algorithmic approach that can provide insights and assess software engineering process trends, through the metrics being tracked, is machine learning. Machine learning is a key factor in providing automation in the measurement of processes in industry today. Machine learning is a strand of artificial intelligence that can allow systems to automatically learn and continually improve on their own without being explicitly programmed (Deloitte, 2019). Supervised machine learning involves training models with labeled data and then training it further on unlabeled data during inference. The models, once robust enough, can learn on their own. A regression model can predict continuous values while a classification model can predict discrete values (Google Developers, 2019). Unsupervised machine learning does not rely on giving the models an output to learn from, using only input values to train the models. Figure 5 shows which of the main machine learning algorithms belong to supervised and unsupervised and the sub-categories within these strands.

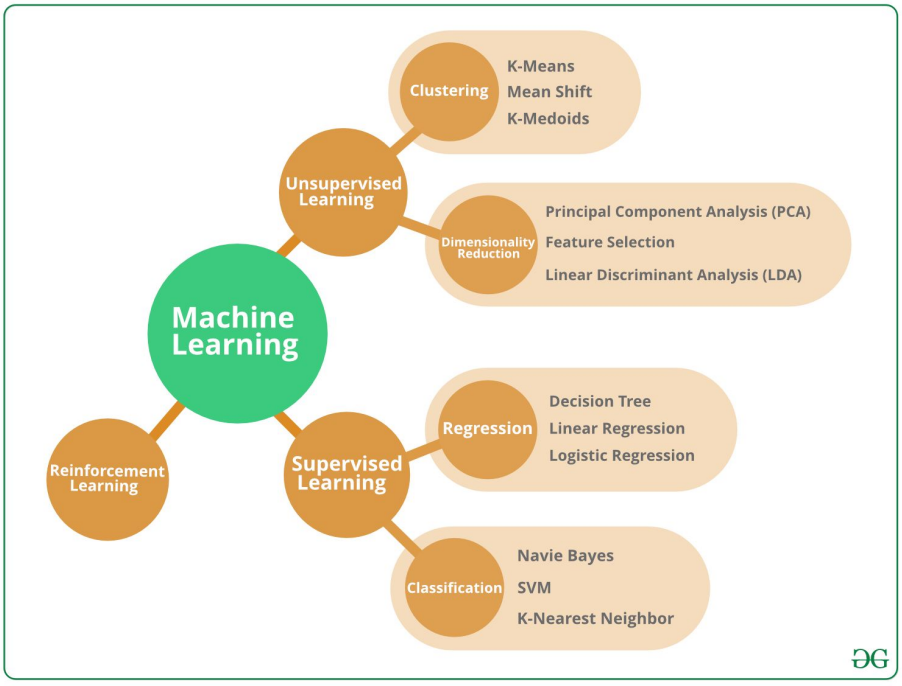


Figure 5. (GeeksforGeeks, 2019)

In the backend of the software development analysis tools there are algorithms running regression, classification, clustering and dimensionality reduction models to provide valuable metrics to software teams. Some algorithms are used to help predict and identify potential pain points and bottlenecks while others are providing the simple metrics that managers need to directly assess their software engineering processes.

The following are a few specific algorithms that can be used in helping gain better measurements in code complexity, code quality and coding time predictions.

3.2 Code Complexity: Cyclomatic Complexity (McCabe, 1996)

Introduced in the 1970s by Thomas McCabe, the cyclomatic complexity provides software engineers with an algorithm to help measure the complexity of a program and it's source code. To calculate the complexity an engineer uses the amount of path changes there are within the source code. For example, if there are no control flow statements (conditions) in the code i.e. one single path through the program then the complexity would be 1. In more complex programs, McCabe uses the control flowpath graph to count the nodes and control flow statements, also taking into account the edges on the graph and the connections between certain nodes. This algorithmic approach can be used to correlate with number of defects, useful in test code coverage and keeping track of the 'structuredness' and limiting the complexity of an application. Figure 6 is an example of control flowpath graph with edges, nodes and control statements.

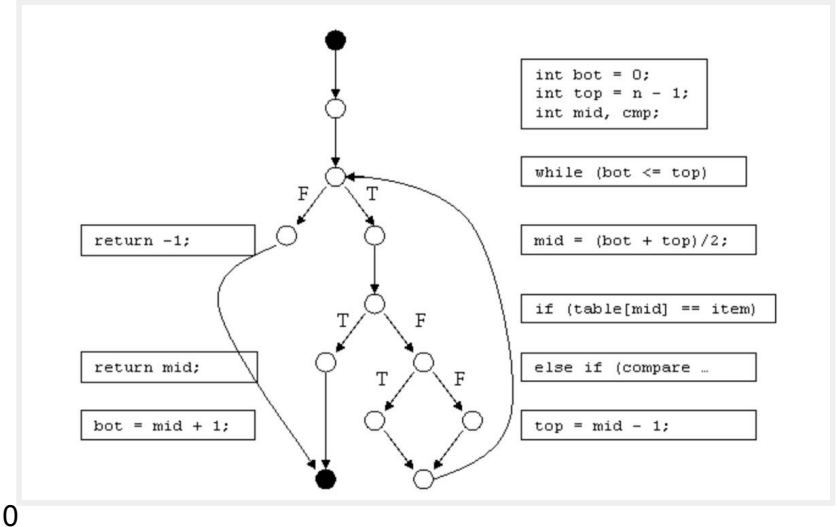


Figure 6. (Medium, 2017)

3.3 Code Quality: Nonlinear Quantile Regression Models (Semmle, 2019)

This is an example of a machine learning algorithm being applied to provide a potential measurement for code quality in the software engineering process. In the Semmle paper, they use lgtm which can take in alerts from checking source code. These alerts detect problems such as run-time bugs and technical debt. The regression model takes in the number and types of alerts in the source code and uses them to provide a value for code quality. The alerts are categorized based on the severity of the problem. These are used as inputs for the model which can then predict the alert-distribution for a system with a given size. Size is measured in lines-of-code. The final code quality score is weighted depending on the types and occurrences of certain alerts. This quality score is also given a corresponding quality grade. Figure 7 below taken from Semmle's paper on Measuring Software Development Productivity: A Machine Learning Approach, demonstrates an overview of the model in production.

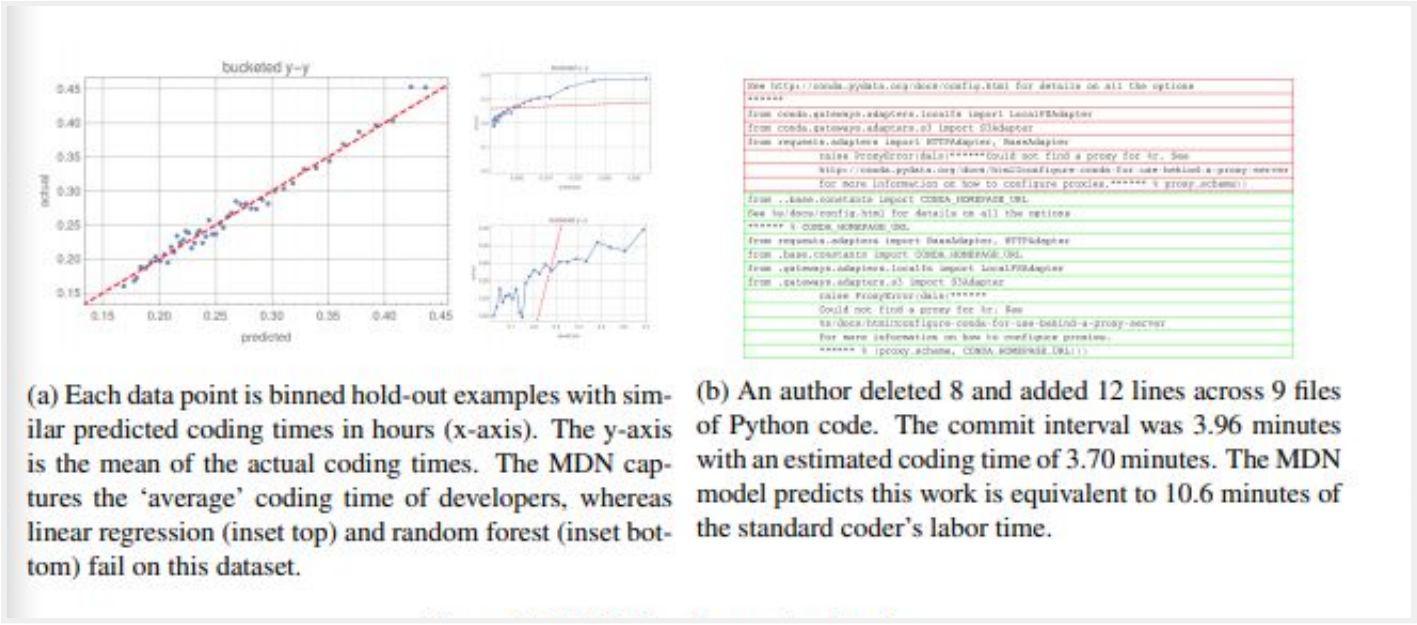
Figure 7. (Semmle, 2019)

Figure 4: A code quality comparison chart from `lgtm.com`. The x-axis is LOC and the y-axis is quality. Each bubble is an open-source Java project. The highlighted project, `apache/flume`, has 107 alerts, a quality score of 0.22 and final grade `D`.

3.4 Coding Time Prediction: Neural Hidden Markov Model (Semmle, 2019)

This model can make predictions on the time between two successive commits. The model takes in timestamped commits. The models that are trained using gradient descent of the coding times then create dataset that associate code changes with the estimate of how long it will take to implement those changes. A deep mixture density network allows the model to reduce the “noise” associated with code change and code time predictions and also accounts for levels of uncertainty. The coding time prediction algorithm detailed by Helie, Wright and Ziegler requires a deep mixture density network for optimal performance. Figure 8 demonstrates the capability of the model.

Figure 8. (Semmle, 2019)



(a) Each data point is binned hold-out examples with similar predicted coding times in hours (x-axis). The y-axis is the mean of the actual coding times. The MDN captures the ‘average’ coding time of developers, whereas linear regression (inset top) and random forest (inset bottom) fail on this dataset.

(b) An author deleted 8 and added 12 lines across 9 files of Python code. The commit interval was 3.96 minutes with an estimated coding time of 3.70 minutes. The MDN model predicts this work is equivalent to 10.6 minutes of the standard coder’s labor time.

4. Ethical Analysis

To analyse whether a business or a person is being ethical we need to have a clear understanding of what it means to be ethical. According to the Cambridge dictionary, ethical behaviour relates to our beliefs of what is morally right and wrong. As morals refer to a person's own understanding of good and bad behaviour, ethical analysis can be a very complex area to navigate.

4.1 Data Regulation

The easiest way to distinguish between right and wrong is following regulations. GDPR has granted people more rights when it comes to the use and accessibility of their personal information. To this extent, employees must consent to the use of their personal data. This means employees have a direct influence on how their data is processed and ensures the "processors" keep employees informed. The new regulations have reduced ethical concerns when it comes to employee performance measurements across every industry and process as there is much more transparency in the usage of an employee's personal data. (Dickinson-wright.com, 2019)

4.2 Data Collection and Usage (Harvard Business Review, 2019)

Other dimensions that affect ethical concerns are the collection and usage of data that follow regulation while at the same time are viewed as too intrusive or morally wrong. These concerns are echoed in arguments against machine learning and big data as there can be human bias in models even with the most robust and carefully trained models. At the heart of this argument is fairness. The metrics used to compare individuals can often lead employees to feel the metrics that compare their work against their colleagues are unfair. In software engineering this is due to the different responsibilities that software engineers have on teams, especially scrum teams and in devOps where members have many competencies to carry out multiple functions within a single team. The two main concerns are the amount of data being collected while you work and how the metrics might be used without a fair

consideration for an employees full workload. Machine learning algorithms and numbers on their own cannot give the context that is necessary in providing a fair assessment of an individual's performance. Managing the concerns surrounding the issue of fairness, as always, becomes a balancing act and must be approached with a view to accepting the trade-offs that come with such complexity.

Conclusion

This report has touched on metrics, platforms, algorithms and possible ethical concerns that all require attention and consideration when measuring the software engineering process. It is clear that businesses should strive to create robust and transparent methods of measuring the software engineering process in a way that facilitates improvements in terms of quality or efficiency of work across any stage of the software development life cycle.

Bibliography

[1] Aydan, U., Yilmaz, M., Clarke, P. and O'Connor, R. (2017). Teaching ISO/IEC 12207 software lifecycle processes: A serious game approach. *Computer Standards & Interfaces*, 54, pp.129-138.

[2] Hussung, T. (2019). *What is the Software Development Cycle?*. [online] Husson University. Available at: https://online.husson.edu/software-development-cycle/ [Accessed 6 Nov. 2019].

[3] CollabNet VersionOne. (2019). *14th Annual State of Agile Survey*. [online] Available at: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508 [Accessed 6 Nov. 2019].

[4] Brooks (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20(4), pp.10-19.

[5] Zapatopi.net. (2019). *Lord Kelvin Quotations*. [online] Available at: https://zapatopi.net/kelvin/quotes/ [Accessed 6 Nov. 2019].

[6] Scrumguides.org. (2019). *Home | Scrum Guides*. [online] Available at: https://scrumguides.org/ [Accessed 7 Nov. 2019].

[7] Sealights. (2019). *11 Scrum Metrics and Their Value to Scrum Teams*. [online] Available at: https://www.sealights.io/software-development-metrics/11-scrum-metrics-and-their-value-to-scrum-teams/ [Accessed 6 Nov. 2019].

[8] Simplilearn.com. (2019). *Project and Process Metrics Classifying the process Metric Measurement*. [online] Available at: https://www.simplilearn.com/project-and-process-metrics-article [Accessed 6 Nov. 2019].

[9] Docs.microsoft.com. (2019). *Agile principles in practice - Azure DevOps*. [online] Available at: https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/agile-principles-in-practice [Accessed 7 Nov. 2019].

[10] Stackify. (2019). *What are Software Metrics? Examples & Best Practices*. [online] Available at: https://stackify.com/track-software-metrics/ [Accessed 7 Nov. 2019].

[11] Anaxi. (2019). *Software Engineering Metrics: An Advanced Guide*. [online] Available at: https://anaxi.com/software-engineering-metrics-an-advanced-guide/ [Accessed 7 Nov. 2019].

[12] Humphrey, W. (2007). Software process improvement—A personal view: How it started and where it is going. *Software Process: Improvement and Practice*, 12(3), pp.223-227.

[13] GitPrime. (2019). *5 Developer Metrics Every Software Manager Should Care About | GitPrime Blog*. [online] Available at: https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/ [Accessed 7 Nov. 2019].

[14] Financesonline.com. (2019). *GitPrime vs Haystack 2019 Comparison | FinancesOnline*. [online] Available at: https://comparisons.financesonline.com/gitprime-vs-haystack [Accessed 7 Nov. 2019].

[15] Hackystat. (2019). *Hackystat*. [online] Available at: https://hackystat.github.io/ [Accessed 7 Nov. 2019].

[16] GitHub. (2019). *Code Climate - GitHub Marketplace*. [online] Available at: https://github.com/marketplace/code-climate [Accessed 8 Nov. 2019].

[17] Codeclimate.com. (2019). *Understand Each Step of Software Development Life Cycle | Velocity*. [online] Available at: https://codeclimate.com/velocity/understand-diagnose/ [Accessed 8 Nov. 2019].

[18] Owler. (2019). *GitPrime*. [online] Available at: https://www.owler.com/company/gitprime [Accessed 7 Nov. 2019].

[19] Gitprime.com. (2019). [online] Available at: https://www.gitprime.com/platform/code/ [Accessed 7 Nov. 2019].

[20] Waydev. (2019). *5 Gitprime Competitors. What Development Analytics Tools Are Leading the Way? | Waydev #1 Git Analytics Platform for Engineering Productivity*. [online] Available at: https://waydev.co/5-gitprime-competitors-what-development-analytics-tools-are-leading-the-way/ [Accessed 8 Nov. 2019].

[21] Deloitte. (2019). [online] Available at: https://www2.deloitte.com/content/dam/Deloitte/global/Images/infographics/technologymediatelecommunications/gx-deloitte-tmt-2018-intense-machine-learning-report.pdf [Accessed 8 Nov. 2019].

[22] Google Developers. (2019). *Framing: Key ML Terminology | Machine Learning Crash Course*. [online] Available at: https://developers.google.com/machine-learning/crash-course/framing/ml-terminology [Accessed 8 Nov. 2019].

[23] GeeksforGeeks. (2019). *Top 10 Algorithms every Machine Learning Engineer should know - GeeksforGeeks*. [online] Available at:

https://www.geeksforgeeks.org/top-10-algorithms-every-machine-learning-engineer-should-know/ [Accessed 8 Nov. 2019].

[24] McCabe, T. (1996). Cyclomatic complexity and the year 2000. *IEEE Software*, 13(3), pp.115-117.

[25] Semmle.com. (2019). [online] Available at: https://semmle.com/assets/papers/measuring-software-development.pdf [Accessed 8 Nov. 2019].

[26] Dickinson-wright.com. (2019). *The GDPR Covers Employee/HR Data and It's Tricky, Tricky (Tricky) Tricky: What HR Needs to Know | Insights | Dickinson Wright*. [online] Available at: https://www.dickinson-wright.com/news-alerts/the-gdpr-covers-employee-hr-data-and-tricky [Accessed 9 Nov. 2019].

[27] Harvard Business Review. (2019). *How Machine Learning Pushes Us to Define Fairness*. [online] Available at: https://hbr.org/2019/11/how-machine-learning-pushes-us-to-define-fairness [Accessed 9 Nov. 2019].