Introduction

In this report, I will investigate the ways in which the software engineering process can be measured and assessed in terms of measurable data. There are four topics that I will discuss and examine in detail.

1. How the software engineering process is measured and assessed using measurable data and how this data is collected and recorded
2. Explore some of the computational platforms available to collect and analyse this measurable data
3. Provide a summary of the algorithmic approaches available
4. Discuss the ethical concerns and effects of measuring software developers and this type of analytics

The last number of years we have seen an explosion in the use and development of technology. With this came an exponential increase in the amount of available data which has caused a significant development in the use of data analytics. Data analytics has become a critically important element of many different industries. Governments, companies small and large now have the ability to analyse processes, behaviours and data in order to improve forecasts, efficiencies and better understand how their industries, customers, employees and the voting public work. These advancements in data analytics and data collection has naturally had an enormous effect on how companies measure the software engineering process.

Measurement of Software Engineering in terms of Measurable Data

Software engineering process

Firstly, what is the software engineering process? The software engineering process is the model chosen for managing the creation of software from when it is first proposed by the customer to its completion and maintenance. The process generally consists of five steps: 1) requirement analysis, 2) software design, 3) unit testing, 4) system testing and 5) maintenance. Some examples of methods are Waterfall, Rapid application development or Agile software development. The Agile methods is increasingly popular today. Like any methodology it is useful to measure its effectiveness so as to achieve better results and better understand its performance. By measuring the software engineering process, you can not only improve your own performance as a software engineer but also help improve a team effort. The software engineering process is not easy to measure. The goals that are defined can vary from project to project and it can be very hard to predict the extent of some tasks until the task is underway.

Relevant Data

One of the first problems in tracking the productivity of software engineers is defining what data one should measure. Collecting data leads to challenges as one must ensure that the data will "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 the development team". In general, the aim is to minimise cost and time, and maximise efficiency and productivity. There are some key questions that need to be answered such as, should the data be quantitative or qualitive; Is the data self-reported or automated; What data is relevant to assess and measure a software engineers' performance?

Relevant data may include:

Data relating to the software engineers tangible project:

- Number of commits
- Number of contributions
- Frequency of contributions
- Individual performance
- Bug Fixing
- Technical debt

Non project related data:

- Repository dates
- Meetings attended
- Company joining date
- Communications with team, managers and clients
- Gender

With the completion of the data collection, comes an important component in data analytics known as taxonomy. A taxonomy provides the framework for your data, enabling you to analyse the performance of a couple of dimensions that matter most to your data. By implementing a taxonomic framework, your data will consistently be categorised by a number of classes. This classification of data can be achieved through various computation platforms such as SQL which will store data in a large database.

Data Collection

Techniques are key to collecting data. One must ensure that the technique they employ is appropriate for the data they want to collect. Techniques should be chosen based on what data is required. Researchers must be cognitive of the methods they use as some techniques can require a rapport with the developer to record accurate data Each method has drawbacks which the research must be aware of.

Manual data collection is unreliable; often being both time consuming and costly. It often results in missing data or data that is error filled which will affect the analysation process. Human error will also lead to unreliable data, so a solution is the automation of the process. One of the methods for tracking software production is the Personal Software Process (PSP) originally developed by Watts Humphrey in his book *A discipline of software engineering* (Humphrey, 1995). The PSP is a process whereby a software engineer is provided with a framework as to how they should track their work. It is concerned with four main statistics lines of code, time taken to complete a task, quality of code produced and progression against a schedule. This results in a large amount of data where automated tools such as Hackystat and PROM are useful.

Measuring Software Engineering

In order to accurately assess and measure performance we must carefully consider the three main measurable branches of analytics; those of time, size and quality. It is also important to take into account softer data such as job satisfaction when measuring software engineering practices. While discussing these three main branches I will also highlight some of the more prominent software metrics.

Time

When measuring productivity, we must consider the amount of time spent on each process of a project. Realistic deadlines or goals must be outlined for each individual stage of the project. Creating goals can instil a sense of determination and achievement in employees. Goal measurement is an effective way to measure how effective an engineering process is.

Detailed records need to be kept in order for assessment to take place. Due to the fact that time records are solely reliant on the input of the software engineer, there can be resulting drawbacks. E.g. the engineer may overstate the task's complexity in order to always finish ahead of schedule. Therefore, time logs should be reviewed in conjunction with the size/difficulty of the task.

In Lowe's article, '9 metrics that can make a difference to today's software development teams', he highlights the importance of planning by further splitting time into four separate metrics.

1. *Lead time*: The time it takes to go from an idea to delivered software. Useful to see if they complete tasks periodically or in bursts, also if they start early or wait until the end to get their work done
2. *Cycle time*: The measurement of the time taken in order to make a change to your software and implement that change into production. Useful to see how quickly issues are being addressed during your development. Projects with high cycle time will often lead to a high build-up of bugs and errors.
3. *Team velocity*: The expected and measured time taken on a given task, usually on agile teams. An estimation on how long each task of the project will take is done before development starts and the actual time taken is then compared with this to gauge the development speed. This can be good in order to maintain momentum on a project so that it gets completed within an acceptable time frame.

4. *Open/Close Rates*: The measurement of how many issues are reported and then closed within a specified amount of time during production. Less about the data and more about the trend it conveys. If there is a trend of many reports but few closes, then there is a big issue in development.

Size

We cannot say that one engineer is better than another because he completed a project faster, if we do not know the size of the respective projects. In the same way we cannot say that one engineer is better because he has more lines of code, if we do not know how much of the code is actually efficient. This brings into perspective the critical link between time and size. This poses complexities as there are no physical dimensions for software. Lines of code is the principal measure of size.

Lines of Code: This is the measure of the amount of lines a specific developer has written. This is easy to measure and automate but has many issues. As it is only counting the amount of code written and not its performance it can be widely inaccurate as it will count white spaces, comments and dead code etc.

Quality

Our final and most important measurable performance metric is code quality. This relates to anything that could detract from the program's ability in meeting user needs. Below are a number of key metrics for measuring code quality.

1. *Defect density*: refers to the number of defects per new and changed LOC in a software engineer's program. In an ideal world, it would never fail, however, this is extremely unlikely. It is important to look at the defect density rather than number of defects as larger code is naturally going to have more defects.
2. *Code coverage*: the measure of how much of the program is run when it is being tested. A poor code coverage usually means a lack of proper testing which can mean a higher chance of running into bugs or errors later on in development.
3. *Application crash rate*: refers to the number of times an application fails divided by the number of times it is used. The recovery time is also important.
4. *Code coupling*: the measurement of the degree of which different classes are related to one another. Usually the higher degree of coupling means the harder it will be to implement changes later in the development cycle without affecting the relation it has with the other parts of the code base which can lead to errors.
5. *Review rate*: is the rate at which engineers find all or most of their programs defects.
6. *Code churn*: This is the measure of the amount of code that has been rewritten within a specific time period. This means if a developer has a high churn rate they are consistently going back and reworking parts of their project which is supposedly finished, this can be a good indicator that they are having issue with the given task and may need help.

Softer Data

It is my opinion that to successfully measure a software engineers' performance, softer data must also be examined, however this is often harder to assess.

Employee Well Being: Naturally happy and satisfied employees will work harder than miserable ones and contribute more to a healthy environment. Employees well-being is closely linked to a positive healthy working environment which should also better communication between teams. The characteristics of happy employees include autonomy, mastery and a sense of purpose.

Computational Platforms

For companies to analyse their data, they require analytics processes. These analytic processes are labour intensive and costly and will often require the purchase of different kinds of hardware.

Analytics as a service (Aaas) is the provision of analytics software and operations through web delivered technologies and could bypass the company having to develop analytical processes of their own. Aaas also have the ability to reduce compliance risks as workers can access data sets through an online portal, without having to duplicate it. Aaas results in reduced costs and compliance risks while increasing user productivity.

A wide range of companies and organisations in the software industry today hold their code base on repositories such as **GitHub**. With over 2.1 million repositories hosted, GitHub provides some useful analysis and metrics into the development lifecycle of a project and also the team working on it. A GitHub commit can tell us a lot about a given developer and allows us to draw some particularly useful metrics from the source code they are contributing to a project.

GitHub commits can be used to give a brief overview of the output of a given developer within a team and to gather metrics such as lines of code added or deleted. These commit statistics however do not provide enough detail to draw much useful conclusions from this output with regards to how much of an impact these contributions have had towards a project. One insight these commit metrics can provide is how important a given developer may be within an organisation due to their contributions to the code base. A developer who has seen the majority of the code pass through from the beginning of a project be it from their own contributions or by reviewing pull requests will have a much larger value than a newly recruited developer, even at a senior level. If any bugs were to arise it is quite likely that this developer may be have knowledge of how to resolve the issue even it was not their own personal contribution.

Automated tools that plug straight into developers IDE's and also the repositories housing a projects code base can be quite useful for analysing metrics within an organisation and also for the purpose of quality assurance (QA). An example of such a platform is the likes of **TestRail**, a product which integrates directly into both GitHub which holds source code and Jira which hosts relevant development tasks and Agile development boards. TestRail does not gather source code productivity metrics such as individual productivity with regards to lines of code produced. Instead it serves the purpose of allowing teams to monitor their testing metrics, providing detailed user-friendly interfaces which allow both developers and project managers to closely monitor the testing quality, test results, code coverage and other useful metrics relating to the individual tasks they are working on. This is extremely useful from a QA perspective as it allows project managers to quickly ensure that individual tasks have been tested accordingly and that a desired percentage of code coverage has been reached. By using these metrics both project managers and developers can significantly reduce the amount of bugs that may arise within a given project thus increasing productivity in the long term by reducing the need to spend time fixing bugs on old issues that may arise down the line in the future. (Test Case Management - TestRail, n.d.)

Tools such as **Jira**, which was developed by Atlassian, are widely used throughout the software development industry today especially within organisations operating on Agile methodology. Like most modern implementations, Jira has moved away from pure data collection and analysis and instead has integrated many features that are useful in modern software development, be that bug tracking, issue tracking or project management into one location. So instead of data being collected in the background and then the analysis highlighting certain areas that need improvement it has now been directly integrated into our workflow.

An individual creates their own account in which they can work on various projects which are ran by their team leaders. Every person has a clear outlined task with goals that they are expected to reach within a specific time frame. Anyone on a team can view their peers work, being able to see when they committed, if they are behind on work, if there are bugs that need to be fixed etc. Many of the various software metrics mentioned in in the first section of this report can now be tracked and displayed automatically and can be easily viewed. As stated previously ethics for this practice are still debated to this day but it is clear from the shift from background collection to integration into our daily workflow that this practice is here to stay for the foreseeable future.

Algorithmic Approaches

Machine learning is a data analysis method that automates the development of analytical models. It is a branch of artificial intelligence that is based on the idea that systems can learn from data, identify patterns and make decisions with little to no human intervention.

Supervised machine learning: For this process of machine learning we ‘train’ the algorithm to predict some well-defined output based on the input data. In this process we give the algorithm a test data set and is asked to map it to some desired output. The larger the data set the more accurate the model will be.

One of such algorithms is the **K nearest neighbour algorithm**, which is a pattern-recognising algorithm used for finding regression and classification. Regression is the idea of giving a set of data and finding the best relationship that represents this set of data. While classification is being given a known relationship, the goal is to identify the specific class that the data belongs to.

This is, in essence, a machine learning algorithm which uses classification based on finding the most similar data points in the training data, and making an educated guess based on their classifications.

Unsupervised machine learning: very much like supervised learning in the way it is tasked to create the most precise model to complete the given task but instead it is not given any desired output to be mapped to. This is used to uncover any hidden structures, but it is rather difficult to implement and as such is not used nearly as much as supervised learning. It may be used in the initial phase of supervised learning as the internal structure of the data can provide additional information on how to better reproduce outputs.

Cluster analysis is an unsupervised machine learning algorithm which aims to distinguish group structure within a data set. The objective with cluster analysis is to cluster the observations in such a way so that there is little dissimilarity within groups, but large dissimilarity between groups. A popular clustering algorithm used is a k-means cluster analysis.

It is my belief that machine learning will play a huge role in automation in the coming years and will likely see them have a large influence in the developer's workflow in the future.

Ethics

Whilst some of the methods discussed above can be extremely useful for an organisation with a software development branch they also bring with them some serious ethical concerns and implications.

There is a large ethic obstruction to the mass collection of software engineering data, which companies must do their utmost best to not infringe upon.

Having developers under constant monitor within a workplace also introduces some ethical concerns such as a quite serious invasion of privacy. Whilst it may be useful to have an automatic quantitative software development measuring tool built in to developers IDE's to track coding metrics, organisations must be careful not to take this too far and start infringing on people's privacy. Proceeding further than this and attempting to gather metrics from a developers daily life within the workplace such as how long their breaks take, how frequently they use the bathroom, who they talk to etc. may introduce an extremely hostile and somewhat dictatorship-like environment that will serve a negative impact on the productivity of developers rather than increasing their productivity.

Employers must take care to only monitor aspects that are relevant to the work the employee might do and not from their personal life. Employees need to know and consent to the extent of the data being collected. People are entitled to location privacy; an employer shouldn't be tracking the location of its employees without a good reason. Using location-based tracking systems affect the security and privacy of user as other apps can access your location thus harvesting your data and being able to process this data as they see fit.

The way in which this data is stored would have to be GDPR compliant if dealing with the data of EU citizens. Organizations in breach of GDPR can be fined up to 4% of annual global turnover or €20 Million (whichever is larger). This is the maximum penalty that can be imposed for the most serious infringements which includes; that they do not have sufficient customer consent to process data. It should be noted that both controllers and processors are subject to these rules, which means that “clouds” are not exempt from the implementation of GDPR. This is a powerful deterrent for companies who would otherwise harvest our data without our utmost consent and every company is now forced to implement these consumer protection laws.

Conclusion

The ability to track productivity is still in its infancy, yet many individuals have strong arguments for or against it. In my opinion, if used correctly, it is a great way for an individual programmer to decide where he could become more efficient, and it could lead to the possibility of self-improvement on the developer side. I believe that there is no single all-in-one application which can harvest and display all the data detailing a software engineers productivity, which means that every organisation must decide to use the appropriate algorithms on an ad-hoc basis, as different people and different companies might view different metrics in higher regard to others. A basis of trial and error would be useful to finding the right platform to analyse and measure the engineering process, but in the end settling for one (or more) is a necessity.

References

1. Atlassian. 2019. Atlassian / Software Development and Collaboration Tools.
    a. [online] Available at: https://www.atlassian.com

2. Code Climate.com (2019). Codeclimate. [online] Available at: http://codeClimate.com

3. Humprey, W.S. (2000,November). The Personal Software Process. Pittsburgh, PA: Carnegie Mellon Software Engineering Institute. Retrieved from The Personal Software Process

4. Humprey,W. (1995). A discipline for software engineering. Reading, Mass.: Addison-
    a. Wesley

5. Humprey,W. (1995). A discipline for software engineering. Reading, Mass.: Addison-
6. Wesley

7. Jira Software - Features. (n.d.). Retrieved from Atlassian.com: https://www.atlassian.com/software/jira/features

8. Lethbridge,T.,Sim,S. and Singer,J. (2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. Empirical Software Engineering, 10(3), pp.331-341.

9. Lowe, Stephen A. (n.d.). 9 metrics that can make a difference to today's software development teams. Retrieved from: https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams

10. Roie, S. (2017, July 25). Analytics as a Service. Retrieved from Birst: https://www.borst.com/business-insights/analytics-as-a-service/

11. Rouse, M. (2013, March). Data Sovereignty. https://whatis.techtarget.com/definition/data-sovereignty

12. Test Case Management - TestRail. (n.d.). Retrieved from https://www.gurock.com/testrail

13. Siddique, N. and Adeli, H. (2013). Computational intelligence. John Wiley & Sons.