**Measuring Engineering – Report**

*Deliver a report that 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 ethics concerns surrounding this kind of analytics.*

**Introduction**

Software engineering process is the model chosen to manage the creation of software from initial customer inception to the release of the finished product. The process usually involves techniques such as Analysis, Design, Coding, Testing and Maintenance. An important part of a managers role is to measure this process. My report is divided into four sections explaining what is involved in the measurement of this process :

1. the ways in which the software engineering process can be measured and assessed in terms of measurable data
2. the computational platforms available to perform this work
3. algorithmic approaches available
4. ethics concerns surrounding this kind of analytics.

**Measurable Data**

Software quality metrics are important to the success of software development, but it is important to keep them in context of specific business goals. It is critical that measurements are designed to answer business questions and not just answer questions such as “how many KLOCs are we up to now?”. Software quality metrics are important but they are irrelevant if the customer doesn’t like the product and the business loses money. Producing consistently successful products is of course related to source-code metrics but it has a lot more to do with the process and production environment. And it has everything to do with delivering value to the customer.

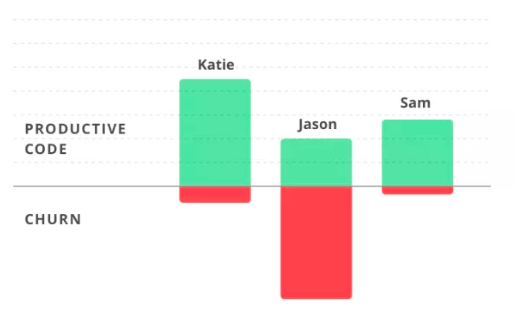
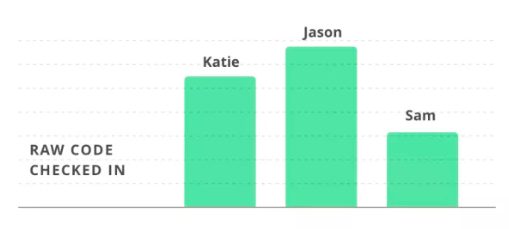
I have chosen to split the data produced by a software engineer into code and non-code. Starting with some of the metrics used to access data that falls under the code category.

1. **Lead time**

This is the time period between the start of a project’s development to the point where is delivered to the customer. As a manager the software engineers lead time history can greatly help you predict when a product will ready for a customer with a high degree of accuracy. This data can be useful if the engineer does not provide you with routine project estimates, since predictions can be made based on the historic lead times of similar projects.

1. **Churn**

Code churn is the percentage amount of a developers own code which represents an edit to their own recent work. It is measured as lines of code (LOCs) that have been altered , added and deleted over a few weeks. Churn allows software managers to control the software development process, particularly its quality. A spike in churn indicates that something is not right in the development process. For example, you may look at the volume of code produced by three of your software engineers over the past month. One of rthe engineers may look as though they have done most of the work from looking at the raw code checked in but a good software engineering manager understands that raw lines of code is an incomplete way to access productivity. This software engineer did in fact write a lot of code , but a lot of the code may have been deleted shortly after resulting in very little net change to the codebase.



1. **Impact**

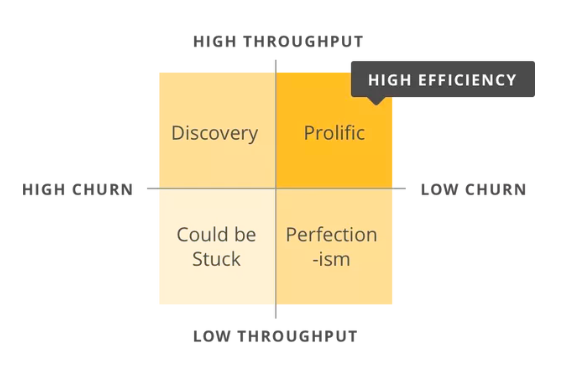
This is a measure of the effect that code changes have on the project, and a way to consider the cognitive load they place on the developer. Change sets that are more difficult to implement will therefore result in a higher impact score. The impact of a change depends on a cumulation of factors such as the amount of code changed, the severity of changes and the number of files affected by the changes. Adding 100 lines of code may seem like the developer has put in a lot of work but it could have much less impact than a change with far fewer lines affected if it includes many insertions and deletions across different files.

1. **Active days**

An active day is a day in which an engineer contributed code to the project, which includes specific tasks such as writing and reviewing code. An engineer’s unique skillset involves building and solving difficult conceptual problems, therefore contributing code is one of the most important things an engineer can do. There are other tasks such as meetings, planning, and chasing down specs which are inevitable. These non-engineering activities will take up at least one day each week. Looking at this type of data allows a manager to consider the cost of interruptions, such as how a group meeting in the middle of the week can affect overall productivity.

1. **Efficiency**

Efficiency measures the percentage of an engineer’s work that is productive, this generally involves balancing coding input against the code’s longevity. Efficiency is a measure independent of the volume of code written. The higher the efficiency rate, the longer the code is providing value to the business. Whereas a high churn rate reduces this value.

Different engineers work will result in different efficiency rates. An engineer that is trialling a new solution may have to try a lot of paths in the discovery phase, this will result in a low efficiency rate. The most creative engineers contribute many small commits, with a modest churn rate, resulting in a high efficiency rate. Efficiency rate is useful for a manager to understand in what area the engineer is best suited.

It is also important to look at the ‘non-code’ measurable data when measuring software engineering process. This refers to the human aspect of the software engineering which have an impact on the performance of software developers.

The paper ‘Network Effects on Worker Productivity’ by Matthew J. Lindquist and Jan Sauermann explores the topic that can exert economically significant effects on their peers.

They analyzed data from a multi-national mobile network operator to see how co-worker productivity impacts on worker productivity via network effects. It was found that a 10% increase in average co-worker productivity produces a 1.7% increase in a worker’s own productivity. They put this productivity spillover down to conformist behavior in the workplace.

Another report written was **‘**Measuring Happiness Using Wearable Technology from the Hitachi Review. This paper looks at the impact happiness has on worker productivity.

It has been found that people who are happy have 37% higher work productivity and 300% higher creativity. It has also been reported that “companies with a large number of happy people have higher earnings per share”. From this it is evident that the degree of happiness of a software engineer directly impacts the level of productivity and creativity in their work.

Problem is that happiness cannot be quantified. This group are looking at quantifying happiness using wearable technology. This may be possible in the not so distant future however for now we must still look at happiness as something not strictly quantifiable.

These are two examples of non-code related factors which could greatly influence the performance of a software engineer. Other non-code factors which should be considered when assessing the performance of a software engineer are their soft skills. Soft skills refer to personal attributes that enable someone to interact efficiently and naturally with other people. Examples of these are email response time, meeting attendance and communication among developers. Managers when considering employees for promotions generally look for candidates with both hard and soft skills. A software engineer’s hard skills are measurable by the code data that was previously mentioned. A software engineer’s soft skills refer to their interpersonal skills and are much harder to define and measure. It is important that there is a combination of hard and soft data used to measure a software engineer’s performance.

**Where to Compute ?**

After a manager has chosen the software engineering metrics they would like to use and have collected the relevant data, the next step is deciding what to do with this data. The paper ‘Searching under the streetlight for useful software analytics’ by Philip M. Johnson uses the metaphor of street lights to help explain this topic.



**Personal Software Process (PSP)**

The PSP was considered a significant development in the measurement of software engineering. The concept was first introduced by Watts Humphrey in the 1990’s. Humphrey was the one of the first of many who went out in search of a better understanding of the impact that personal characteristics and behaviours of software engineers have on producing higher quality software. Humphrey believed that by following a structured development process and constantly tracking work and progress, a developer could have a better understanding and allow them to improve their output. A key component in the Personal Software Process was self-assessment and management. It would be more feasible for developers to meet deadlines, manage the quality and reduce the number of defects in their work having set personal goals and time commitments prior to starting their project. The PSP model requires manual data collection and analysis. This required a substantial amount of time to be spent recording defect logs, code checklists, project plans and time estimates. In practice the PSP model proves too time consuming for developers to efficiently put into practice. Referring to the stated metaphor the original version of the PSP is compared to “lighting a candle” rather than looking under a streetlight because of the custom , situation specific tendency nature of PSP. The ability of a candle being moved to navigate through darkness, PSP encourages users to use analytics best suited to their needs. PSP is not without problems and that is what led to other improved processes of measurement such as the LEAP toolkit.

**LEAP**

The leap toolkit attempts to address the problems encountered in the PSP by automating and normalizing data analysis. The toolkit does still require the developer to manually enter most of the data, the toolkit then automates subsequent PSP analysis and also provides some analysis not provided in PSP such as various forms of regression. Leap data is portable. It creates a repository of personal process data that developers can keep with them as they move from different projects and organizations. In reference to the metaphor, the leap toolkit replaces the PSP candle with a campfire. Providing higher level tool support metaphorically increases the light by improving data quality and decreasing the manual analysis required. Although Leaps introduction of automation makes certain analytics easier to collect it also makes others increasingly difficult. The paper states that after several years the leap toolkit that they agreed with Humphrey, that the PSP approach could never be fully automated. The question was posed ‘What kinds of useful software analytics could we obtain if both collection and analysis were “free”? Answering this question brought about a decade-long research project called Hackystat.

**Hackystat**

Hackystat goes against the conventional wisdom of defining high level goals first and then figuring out what data collection analysis is necessary to achieve them, in fact Hackystat does the opposite. They aimed to develop ways of collecting process and product data whilst minimizing overhead costs for developers, then decide the high-level engineering goals that could be supported by analyses on this data.

Hackystat “implements a service-oriented architecture”. Sensors attached to the development tools gather process and product data and send it to a server. This is useful as other services can then query and then build higher-level analyses based of data collected by Hackystat.

There are four main design features of Hackystat:

1. Client-and-server-side data collection: Instrumentation was developed for client-side tools like editors, build tools, and test tools. Server-side tools were also developed such as configuration management repositories and build servers.
2. Unobtrusive data collection: Using Hackystat client-side tools locally caches data gathered while an engineer works offline. It then sends the data to the Hackystat data repository when the engineer reconnects
3. Fine grained data collection: Hackystat allows for data to be collected on a minute-by-minute or second-by second basis. For example, Hackystat supports a measure called buffer transition - collecting a data instance anytime an engineer changes the active buffer from one file to another. Hackystat tracks an engineer as they edit a method, develops a test case for a method or invokes the test.
4. Both personal and group-based development: It enables engineers to define projects and group work. Hackystat can then keep record interplay among engineers such as when different engineers edit the same file.

**Codacy**

Codacy is another alternative platform. “We’re on a mission to help developers ship better code, faster”. Codacy is a technology start-up that was founded in 2014 by Jaime, the CEO. It is a platform that was created to save engineers thousands of hours of time reviewing code and monitoring code quality. Their aim was to allow engineers to focus on the parts of their work that could change the world while Codacy make the process of creating high quality software easy.

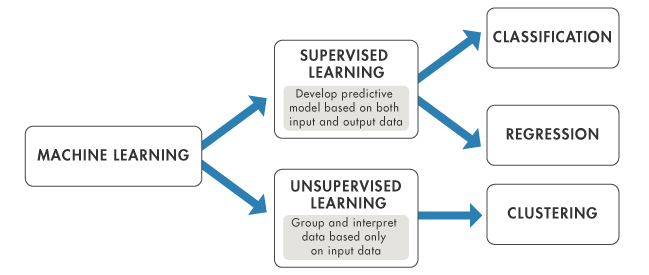


**What Algorithms?**

Data analytics is an essential part of measuring the software engineering process. The question is what algorithms are available and which ones are relevant to us?

**Machine Learning**

Machine learning or computational intelligence refer to the process of giving computer systems the ability to “learn” from data without being explicitly programmed to do so. There are different learning styles in machine learning algorithms, supervised or unsupervised.

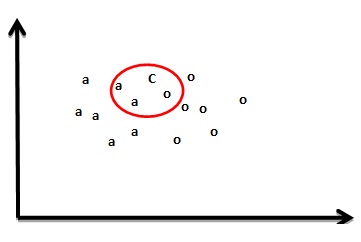


**Supervised learning**

Supervised learning algorithms are those which use training data, which maps known inputs onto known outputs. A model is prepared through a training process in which it is required to make predictions and is corrected when those predictions are wrong. The training process continues until the model achieves the desired level of accuracy on the training data. Example problems are classification and regression. Example of an algorithm is Logistic Regression.

K-Nearest Neighbors

K-Nearest Neighbors (Knn) is a supervised machine learning algorithm simply looks at the k closest points of known origin to the point of unknown origin. The point is then classified as belonging to the group which contains most of these k points. This can be useful in software metrics by using previous data of when errors have occurred KNN can make a prediction as to when an error will occur in the future based of similar past examples.

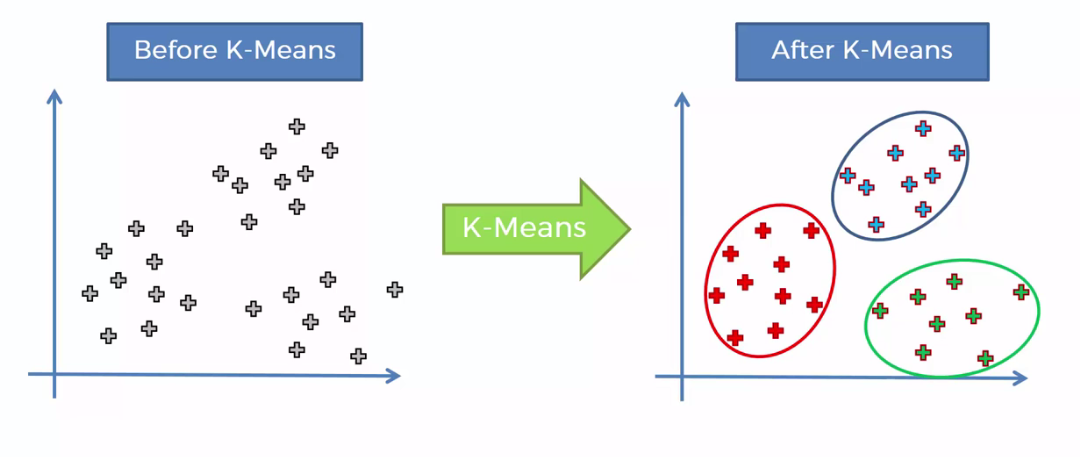


**Unsupervised learning**

An unsupervised learning algorithm uses input data that is not labeled and does not have a known result. A model is prepared by deducing structures present in the input data. It may be through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity. It is easier to obtain this kind of unlabeled data and is a more realistic method to use in research. Example problems are clustering and dimension reduction. K-means is an example of an algorithm.

K-means clustering

K-means clustering is a type of unsupervised learning algorithm which is used when you have data without defined categories or groups. This algorithm attempts to find groups in the data, with the number of groups represented by the variable K. This can be useful in software metrics, as you could be collecting huge amounts of data. This algorithm would allow you to find similarities in the data points helping you to group them and then classify them.



**Ethics**

My report has covered in detail the topic of the measurable data used to assess the software engineer’s performance as well as how and where this data can be analysed. The last part to consider is the ethics involved.

Ethics involves ‘the moral principles that govern a person’s behaviour or the conducting of an activity’. What are the moral principles that govern the collection and analysis of software engineering data?

It is important to first look at the ethics of data collection in general. In today’s world data is everywhere, it is found in almost every area of our lives. We see it in banks, small businesses to social media. Any service that requires some sort of personal information being collected and used must consider the ethics of what that data is being used for.

An example of this would be the Cambridge Analytica story. Facebook exposed personal data of almost 87 million Facebook users to a researcher in Cambridge Analytica the political consulting firm that was involved in the Trump campaign. The app created by this researcher allowed him to gather information from Facebook about users their friends without their knowledge. This scandal brought the topic of data protection to the attention of the general population and now more than ever people are questioning how much they can trust Facebook as well as other social media platforms with their personal information.

General Data Protection Regulation (GDPR) is a new set of rules designed to give EU citizens more control over their personal data and how it may be used. Unfortunately, data breaches are inevitable. Information gets lost or stolen or released into the wrong hands as was the case with Cambridge Analytica. With GDPR, not only will organizations have to ensure that personal data is gathered legally and under strict conditions, but those who collect and manage it will be obliged to protect it from misuse and exploitation, as well as to respect the rights of data owners. GDPR is a step in the right direction in terms of prioritising the protection of personal data.



There are similarities to be drawn between the protection of personal data and the protection of software engineer’s data. Computational platforms like the ones I discussed above take copious amounts of data from the software engineers without them even realising. Some software engineers see this as invasive and an abuse of power. Managers have the power to use data for whatever they like, giving engineers no control over their own data.

I have previously mentioned the paper “Searching under the Streetlight for Useful Software Analytics” in my report, Philip M. Johnson of the University of Hawaii. The paper claims that the easier an analytic is to collect and the less controversial it is to use, the more limited its usefulness and generality. For example, collecting the data in a configuration management repository is easy, and the repository’s public nature means that developers generally don’t object to analysis of this data. However, the resulting analytics are constrained by the very narrow slice of development activity captured. Conversely, the hackystat technology can yield insightful, high-impact analytics but there are also certain social or political problems associated with it. The team of researchers at University of Hawaii discovered a number of these problems. Firstly, some developers viewed the hackystat technology as a bug. They would not install software which collected data regarding their work without telling them. Secondly, the fine-grained data collection was found to sometimes cause discord within a development group. The transparency of the hackystat data means that developers know exactly what everyone in the group is and is not doing. This can cause conflict. Finally, developers are not comfortable with the level of fine-grained data about their work which was being provided to management. Some developers consider the hackystat technology to be invasive.

The choice of metrics is important and should be measured as they can provide managers with useful information which they can then use to get the most out of their employees. However, a clear line must be drawn between data that is work related and personal data. More frequently companies are inappropriately analysing private personal information about their employees.

A New York Times article highlighted the level to which companies today use their customer’s personal data to increase their sales. “Almost every major retailer, from grocery chains to investment banks to the U.S. Postal Service, has a ‘predictive analytics’ department devoted to understanding not just consumers’ shopping habits but also their personal habits, so as to more efficiently market to them.” It is a worrying thought to consider how much companies can analyze people simply based on what, when and where they purchase goods. The more the data analytics industry grows and expands the more such practices will become common in our society.

**Conclusion**

To conclude, the measurement of software engineering as a process involves many different considerations. Questions must be asked such as what data should be used , where should data be stored? How to assess this data ? Is the collection and processing of this data ethical? All of these questions have been addressed in the report.

The aim of this report as a whole was to discuss and analyze the process of measuring software engineering. I have done so under the headings of measurable data, computational platforms, algorithmic approaches and ethics. I believe that they give a well- rounded view of the field. As I have mentioned throughout, it is an area surrounded by a huge number of questions. Such questions blur the lines between ethical and unethical, between right and wrong. The development of this field is one that has undergone rapid expansion in the past decade. ‘Artificial Intelligence’ and ‘Machine Learning’ are subjects that are becoming increasingly common today. However, it appears that few genuinely understand the true meaning of them and the potential they have to alter our world irreversibly. And, even though the fundamentals of such change are for the better, there is inevitably an element of harm involved. It is sometimes hard to comprehend the extent to which humans can be tracked and analyzed in today’s world.

In my opinion, the responsibility to make sure that this power is used for a greater good is up to us as a generation.

**Bibliography**

1. [**https://techbeacon.com/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals**](https://techbeacon.com/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals)
2. [**https://blog.gitprime.com/why-code-churn-matters/**](https://blog.gitprime.com/why-code-churn-matters/)
3. [**https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/**](https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/)
4. <https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf>
5. http://www.hitachi.com/rev/pdf/2015/r2015\_08\_116.pdf
6. <http://www.citeulike.org/group/3370/article/12458067>
7. https://www.codacy.com/about
8. <https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/>
9. <https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides7.pdf>
10. <https://www.scss.tcd.ie/Brett.Houlding/ST3011_files/ST3011slides9.pdf>
11. <http://csciwww.etsu.edu/gotterbarn/artge2.htm>
12. <https://www.linkedin.com/pulse/ethical-concerns-software-engineering-world-rarely-discuss-chacko/>
13. <https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram>
14. <https://www.nytimes.com/2017/11/19/opinion/facebook-regulation-incentive.html>
15. <https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html?pagewanted=1&_r=1&hp>
16. <https://eugdpr.org/>