*Measuring Engineering Report – Odhran Mullan*

***Introduction***

This report will consider the ways in which the software engineering process can be measured and assessed in terms of measurable data. It will also detail an overview of the computational platforms available to perform the work and the algorithmic approaches available. Finally, it will detail the ethics concerns surrounding this kind of work.

Since software development began there has been people trying to find out how to optimise the process and get the most out of their developers. This report details my opinions on how this is possible through measuring and the analytics of data.

***Data Measurement -***

There are different measures that we can observe to give an insight on the productivity of the Software Engineering process. This process needs to be monitored by managers to ensure a smooth delivery of the product. For this, managers need a clear idea of how their team is performing and how the product/process is. For this we need to be able to objectively measure output of these.

Production metrics are an attempt to measure how much work is done and determine the efficiency of software engineering teams.

*Active Days*

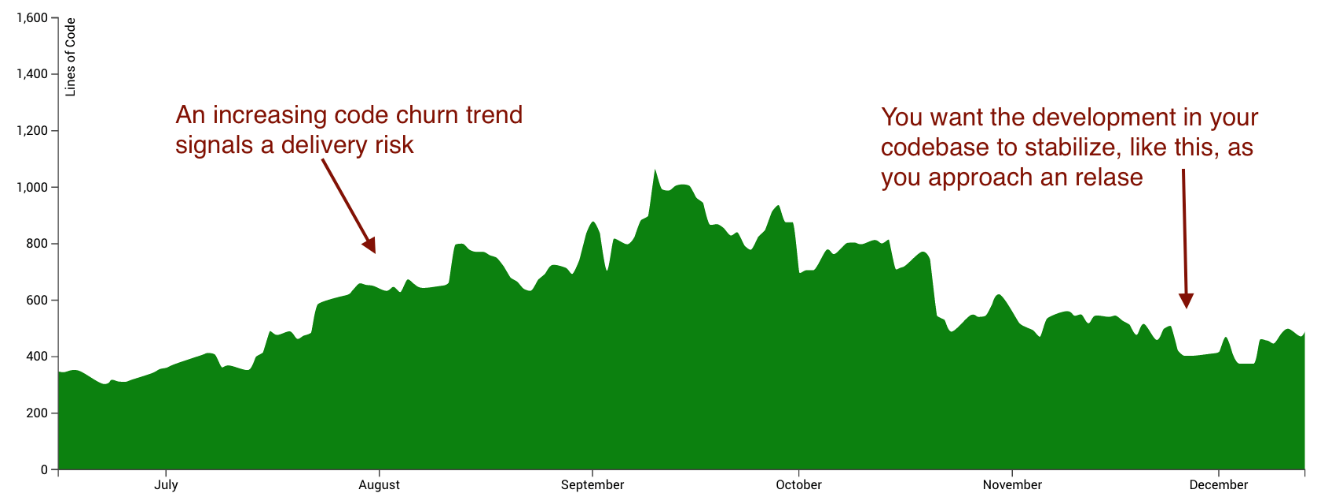
Active days is an important production metric. It is a measure of how much time a developer contributes code to a project. If a developer contributes on a project on a day then that day is considered an active day. This does not include planning or tasks not directly related to the development of the project.

*Assignment Scope*

Assignment scope is another production metric showing the amount of code that a developer can support in a year. This is used to see how many developers would be needed to be set aside for maintenance of a system.

*Code Churn*

Code churn is a metric that represents the number of lines of code that were modified, added or deleted in a specific period. Code churn is used for visualising the development process. Managers should watch out for large spikes in code churn, which can signify problems with the development process. It also lets you inspect the size and impact of the completed tasks.



*Mean Time Between Failure/ To Repair*

Mean time between failures and mean time to repair are metrics that measure how the product performs in the release environment. These metrics quantify how often the software might fair and how long it would take for the developers to fix any prospective problems. It also measures the time from a hypothetical security breach to when a patch is delivered.

*Number of Commits*

The number of commits that a developer has contributed to a project is also a production metric that is widely used. However, this can also be a misleading metric to use. The number of commits does not separate a good developer from a bad developer. A developer may contribute few large high-quality commits which is objectively better than many small low-quality commits. On the other hand, it paints a good picture on how active developers are and can be used for analytics showing when developers are most active and if they contribute in clusters or continually.

*Source Lines of Code*

Source lines of code is a metric used to measure the size of a software project. It is used to estimate the productivity of development once the software is produced. It is commonly used to predict how long it would take to develop a project and how much effort/manpower would be required. The two major measures of source lines of code, physical and logical. The physical measure of source lines of code is a count of lines in the source code including comments. While the logical measure of the source lines of code measures the number of statements. It is considered a poor measure of productivity as it could vary due to the coding standards or programming language also good code is often compact and the managers should be promoting this.

*Impact*

Impact is a productivity metric used to measure the effect of a code change or addition or deletion in a project. An alteration to the code that affects multiple files has more impact than an alteration that only affects one or two files.

*Measurement Process*

The measurement process includes five sets of activities. The first activity is Formulation which is performing measurement and developing appropriate metrics. The second activity is collection which is collecting the data. Next there is analysis where the metrics are actually calculated. The process moves onto the interpretation of metrics where the results are interpreted to draw out meaning. Finally, there is feedback where the results of the measurement process are relayed to the team from the product metrics.

***Computational Platforms Available –***

Using computational tools to actually collect these metrics detailed above is absolutely paramount. It is unrealistic for a project manager to expect his developers to create their own tools to collect measurable data. There are many different tools available for this exact purpose and below some of these tools will be outlined.

*Personal Software Process*

The personal software process is designed to help developers into managing their practices better. It shows developers how to plan and track projects, how to establish goals, how to use a rigid defined process and how to record their performance. The main principle behind the personal software process is producing high quality systems and because of this every engineer working on the project must produce top-quality work.

The principles of the personal software process are as following, each developer is different and for them to become more active, their work needs to be planned and this should be based on their personal data. To improve their performance, developers should use regular and well-defined processes. For developers to produce top-quality projects, they should be made to feel personally responsible for the quality of what they are creating. It is easier and less expensive to trace and fix defects sooner rather than later. It is easier to prevent errors rather than searching for them at a later date. Finally, the easiest, fastest and least expensive way to do any task is by doing it correctly.

*GitPrime*

A tool that is used by many for computing the data and metrics that have been gathered is GitPrime. GitPrime uses data from GitHub, GitLab or any other Git based code repository, with this data it helps software developers to move faster, optimize their work habits and improve their developing with concrete data.

The platform is known for its power to display a complete overview of the whole software development process. It visualises the activities of the team for the manager in real-time. GitPrime bridges communication gaps and provides solid facts to the managers to present to their teams. It removes subjectivity in the development process as problems and issues can be presented directly to the team.

*Code Climate*

Code Climate is a static quality analysis tool, it was created by Bryan Helmkamp. The tool scans the codebase looking for quality issues in the code such as duplication of code or any code smells. It is one of the most popular computational platforms available and supports most of the most popular languages, for example JavaScript, Python, Ruby, Java and PHP. Some of the features that code climate provides are, automated git updates which means every time you commit to the repository the analysis tools run on the new code. Code climate provides accurate up to date data enabling developers to see feeds showing when and how code changes. It provides code sharing for the entire development team which increases cross project visibility. A security dashboard is created by Code climate and it shows all of the projects weaknesses and how to address them, it also shows when any of the weaknesses first appeared. Code climate matches code quality against areas of high churn showing hotspots where particular attention is needed. It also shows the test coverage of the whole project and where tests need to be added.

*Collaborator*

Collaborator is another code review tool. It is built for teams who are working on projects where the quality of the code is of utmost importance. It was created by the company Smartbear. It is one of the most comprehensive review tools one the market. Some of the key features of the Collaborator software is that all users can dee code changes, identity defects and the can make comments on any line. It can be easily integrated into some of the most popular IDE’s including Visual Studio and Eclipse. One of Collaborators unique features is that you can create custom review templates. In these custom templates you can set your own fields, checklists and participant groups. These custom reports are especially useful as they can be tailored for the firms need and can be used to drive improvement of the company’s processes. Finally, Collaborator facilitate peer document reviews in the same package so teams can easily make sure everyone is up to speed on requirements or design changes for example.

*Gerrit*

Opposed to the other tools that we have discussed above, Gerrit is a free web-based code review tool, unlike the others which are expensive paid pieces of software. It was originally created by Google when a code review system for Android was requested. Gerritt is a commit review tool where developers amend individual commits in response to feedback that the tool gives. The developers review the suggested changes and then can accept or reject the changes depending on how they feel about them. It tracks each version of every commit which allows developers to see all differences between versions. Developers can use pull individual commits and push certain groups as commits as part of a patch. The tool can be integrated into Git and can provide repository management. The tool is especially useful for dissecting a small number of detailed segments of code and suggesting suitable solutions for these.

***Algorithmic Approaches –***

Another approach to measuring the performance of a developer is by using algorithms. These are a set of rules to be followed in calculations or in problem solving. Of course, in computer science we can represent these rules in code. Below I will discuss some algorithms that could be used to measure the performance of developers.

*K Nearest Neighbours Algorithm*

A possible way of classifying the software development process would be by using the K nearest neighbours’ algorithm. In this algorithm the data must be classified first before it can be used. Data that has already been classified can also be used. For example, classifying developers as active or inactive, or classifying code as problematic or acceptable. The algorithm takes part of the data as a test set after it has been classified and then it analyses the data within. To analyse the data the algorithm finds the distance of each data point which has known origin and then compares it to the data points that have an unknown origin. Moving on to the classification of the points of unknown origin, the algorithm finds the classification of the nearest k points, then the data point is assigned to the group with the most points in the group of k. Then this process is repeated many times and in each iteration the value of k is changed.

The algorithm can estimate the thought process of a manager examining his team and deciding whether a developer is up to scratch or now. However, the algorithm has some flaws, such as if the person who is training the algorithm cannot judge the difference between a poor developer and an acceptable engineer. Then the algorithm will likely have a flawed outcome mirroring the opinion of the classifier. Also, if the data is not classified on production it can be prone to the inherent biases of the classifier.

*Decision Tree Analysis*

Decision tree analysis uses a tree-like graph of decisions and all their possible outcomes including all the small chance outcomes. This type of analysis facilitates a structured approach to reaching rational conclusions. The goal of decision tree analysis is to create a model that predicts the value of a target variable based on several input values. This can be applied to analysing developers by putting in different data metrics such as commits, lines of code or active days and training the algorithm. It could analyse metrics such as these and give a result on how effective a developer they are.

*Neural Networks*

Using neural networks could be another algorithmic approach to classifying the software development process. Artificial neural networks are complex algorithms designed to try and mimic natural neural networks like the ones in our brains. These networks are mostly used currently for data analysis and classification, so this fits perfectly into classifying the software development process. Neural networks can change with how they experience the data. They allow for data clustering, which is the grouping data by characteristics, which is very useful for classifying the software development process. Neural networks can improve on the possibility of human error in algorithms such as the k nearest neighbours as it doesn’t need to be trained by a person.

*Reinforcement Learning*

Reinforcement learning is an algorithmic trial and error approach to building computational intelligence. It is a computational approach to learning through actions, receiving feedback and improving its processes as a result of this feedback. The algorithm is not instructed on what actions to take, it is given a result and must find the actions that return this result by trial and error. The behaviour can be learned once off or more commonly it continues improving itself as time goes on. The global optimum is where the algorithm would need to get to if the behaviour with the perfect result is to be observed.

An example of an algorithm that works off the reinforcement learning ideas is the Monte Carlo methods. The Monte Carlo method performs risk analysis by building models of possible results by substituting a range of values for any factor that is uncertain. It then calculates results over and over, each time using a different set of random values from the probability functions. Depending upon the number of uncertainties and the ranges specified for them, a Monte Carlo simulation could involve thousands or tens of thousands of recalculations before it is complete. Monte Carlo simulation produces distributions of possible outcome values. This could be used in the software engineering industry for finding the risk of hiring a developer, or by putting the developer’s metrics into the simulation and training it to spit out the probability of them being a good asset to the team.

***Ethical Concerns –***

As computers are used more and more in society, one of the most important issues in modern society is the ethics behind large computer systems and companies. The use of user data within these companies’ products is also of paramount relevance at the minute. Software developers as the creators of these products that reign over our lives have great responsibility on being ethical or irreparable harm could be caused.

One of the greatest steps forward in the field of ethics in software development was the introduction of the General Data Protection Regulation or GDPR. This came into effect in the summer of 2018. These set of regulations impose stricter laws on organisations that hold personal data. For example, no data can be used by the organisation without explicit, informed consent by the owner of the data. Datasets also cannot be used to identify a subject. Any processor of personal data must clearly declare if they are engaging in data collection, they must declare the lawful basis of the data collection and state how long the data is being retained for or if it is being shared with any third party. So, taking GDPR into account to collect your employee’s info to analyse you would have to have explicit permission from all employees. Also, employers collecting too much data on their employees or monitoring them too often could be a breach of their rights to privacy.

Employers should be careful to not monitor elements of their employee’s life that are not relevant to work. Employers should only monitor what is relevant to the software development process. Even though the company is paying the employees, it doesn’t mean that they have the right to collect data on elements of the employee’s private lives. Humans are imperfect creatures and software developers are human, so no amount of monitoring and tracking will unlock 100% productivity, there is no such thing as the optimum worker. Everyone has their strengths and weaknesses.

An example of a severe breach of ethics in the software engineering industry was with Facebook and Cambridge Analytica. This company had been harvesting Facebook user data without their consent and was estimated to have over 5,000 data points on every user in the US. They then used this data to create profiles on users and push biased political ads towards them. This was a clear interference in the elections, as the firm was hired by Donald Trump’s campaign it was suspected that their biased political advertising helped him win the election. Facebook also have a great deal of responsibility in this as they unethically do not fact check political advertisements. As a huge company they obviously have the resources to do this because they fact check any other advertisements.

***Conclusion –***

In conclusion the software engineering process is a complex process that can be assessed using many metrics that are easily found. These metrics can be analysed using algorithms or by industry packages. Finally, the possible ethical conflicts of software engineering and monitoring employees have been addressed.

***References***

https://www.tutorialspoint.com/software\_quality\_management/software\_quality\_management\_metrics.htm

http://ecomputernotes.com/software-engineering/software-measurement

<https://explainagile.com/agile/personal-software-process/>

https://stackify.com/track-software-metrics/

https://www.gitprime.com/platform/code/

<https://www.softwaretestinghelp.com/code-review-tools/>

<https://medium.com/automation-with-batista/improve-the-code-quality-of-your-automated-tests-with-code-climate-72367bfce46b>

https://www.geeksforgeeks.org/software-engineering-classification-software/

<https://ecommons.cornell.edu/bitstream/handle/1813/31637/BU-1065-MA.pdf;jsessionid=306EA9AEB67D3E729EA07AD8D13D3466?sequence=1>

<https://www.palisade.com/risk/monte_carlo_simulation.asp>

<http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf>

<http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf>

https://www.researchgate.net/profile/Jan\_Sauermann2/publication/285356496\_Network\_Effects\_on\_Worker\_Productivity/links/565d91c508ae4988a7bc7397.pdf