**Measuring Engineering – A Report**

**6th of December 2017**

*“To 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”*

**Measureable Data**

An initial problem encountered in measuring and assessing the productivity of a software engineer’s performance, is defining the metrics in which should be used, to measure the performance. By assessing previous data we can attempt to predict future outcomes with the goal of improving that performance. It should be established early on whether the metrics are to be quantitative, qualitative, automated or manually inputted. There are numerous different types of metrics which can be assessed;

**Lines of Code**

The metric of lines of code refers to counting the lines of code that a software engineer produces. This is not a good metric to measure, as more lines of code does not necessarily mean a better code. Codes should aim to be accessible, efficient and as short as possible while aiming towards a high quality.

**Commit Count**

A commit count shows how many times a user will have uploaded new code to a repository i.e. GitHub. This can be beneficial as it can show how often a software engineer is working and developing the project. However, a developer could commit many times a day, giving the illusion of working well when those commits may only have small changes. While this metric may encourage good practice, it is not reflective of a good performance by a developer.

**Test Coverage**

Test coverage measures the amount of testing performed by a set of tests and is usually portrayed as a percentage i.e. JUnit testing in Eclipse. The problem with test coverage is that 100% coverage does not mean 100% tested. Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other doesn’t.

**Agile process metrics**

For agile process metrics, the basic metrics are lead time, cycle time, team velocity, and open/close rates. These metrics aid planning and inform decisions about process improvement and time management. Agile process metrics they don’t measure success or value added and they have nothing to do with the objective quality of the software. However, these metrics should be measured as they give an insight into where essential processes need attention.

* **Lead time**

Lead time can be defined as the amount of time that elapses between when a process starts and its completion. This metric is beneficial to assess, if monitored effectively it can be easily highlighted where the software engineering process takes the most time, with an aim to improve the lead time.

* **Cycle Time**

Cycle time refers to how long it would take to make a change to a software system and deliver that change into production.

* **Team Velocity**

Team velocity refers to how many “units” of software the team typically completes in an iteration (a.k.a. “sprint”). This number should only be used to plan iterations. Treating velocity as a success measurement is inappropriate, and making a specific velocity into a goal distorts its value for estimation and planning.

* **Open/close rates**

How many production issues are reported and closed within a specific time period. The general trend matters more than the specific numbers.

The importance of using metrics to assess and measure the software engineer process is clear. However, it is vital that the metrics outlined are beneficial to the overall goal of the project, and measure aspects which contribute to the overall efficiency and effectiveness of the project.

**Computational Platforms Available**

A number of computational platforms have been made available in order to aid and assist in collecting data on the metrics used to measure and asses the software engineering process. Until the mid to late 1990’s most software engineering metrics were designed for the use of companies, rather than to be utilized by the software engineers themselves. Three computational platforms I will talk about are PSP, Hackystat as outlined by the article written by P.M. Johnson et al. “*Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined”*, and finally Web-CAT & EclEmma*.*

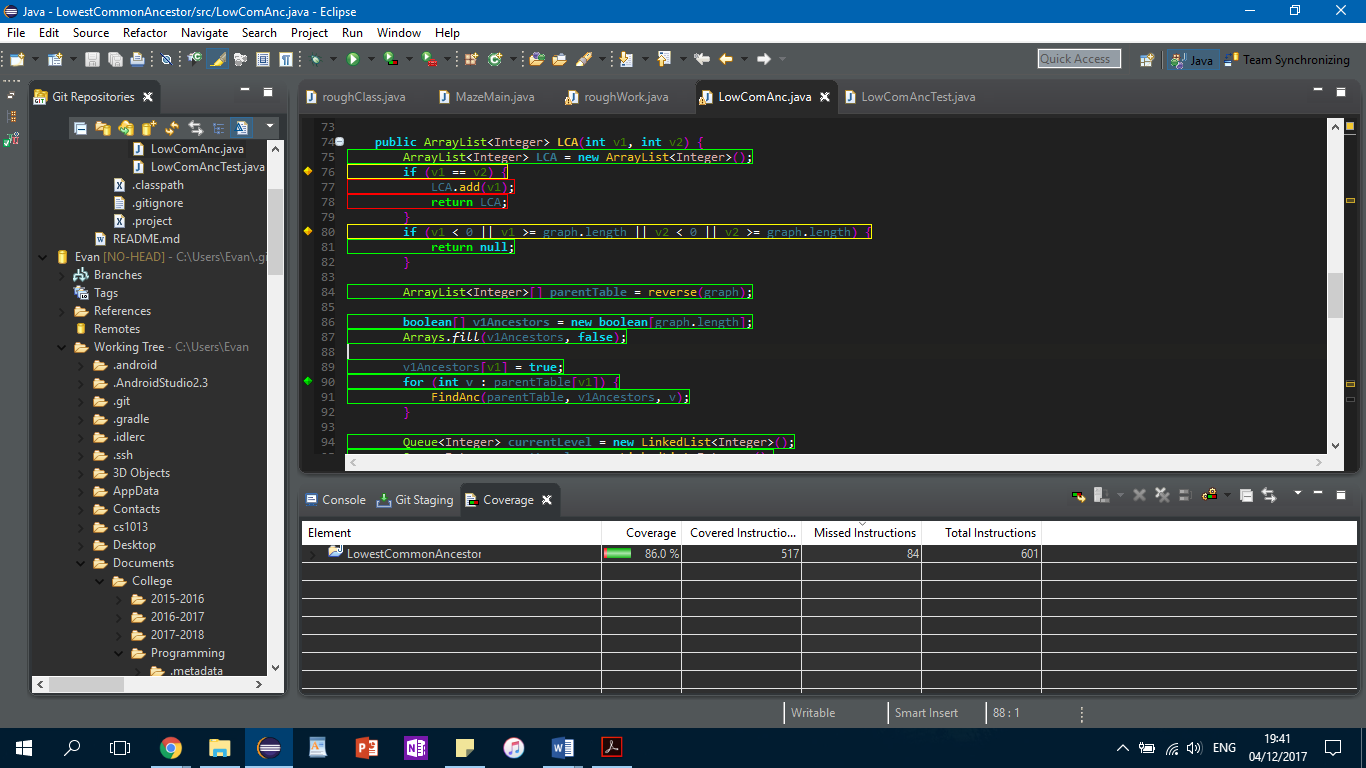
**Personal Software Process (PSP)**

The personal software process is a structured software development process that is intended to enable software engineers to understand, and improve their performance by tracking their predicted and actual development of code. PSP was created by Watts Humphrey in 1995. The PSP was designed for use with any programming language or design methodology and it can be used for most aspects of software work such as writing requirements, running tests and repairing defects. PSP was an initial success as it was deduced by researches from the Software Engineering Institute that the “PSP improved the students estimation accuracy and product quality”, (P.M. Johnson). Although PSP’s initial positive induction, it was found that the PSP was time consuming and although it improved the quality of programs, many students stopped using PSP as they found the “overhead of metrics collection and analysis to be excessive”, and the found there was a need for students to context switch to process recording.

**Hackystat**

Hackystat provides the user with a range of sensors to collect data from tools such as Eclipse, JUnit and Visual Studio. Hackystat completely automates both collection and analysis of metric data. It is a progression from PSP as it addresses the problems with overheads and context switching that students found were a deterrent from using PSP. In comparison to PSP, Hackystat offers some drawbacks. These are that it does not support certain activities that the user is provided through the use of PSP, such as planning and estimation.

**WebCat & EclEmma**

Web-CAT is a plug-in based web application for automatically grading and programming assignments. It allows the user the choice of programming in numerous different languages. Web-CAT is a submission server and it is gives automatic grading for students using written tests. Web-CAT is a beneficial tool for the software engineer that tests the code coverage of java code. The user will input code into Web-CAT and it will return a percentage and highlight how much code coverage the user’s tests have. EclEmma is an Eclipse integration and is based on the JaCoCo code coverage library. It works the same as Web-CAT providing the user with a percentage representing code coverage of a user’s test.

EclEmma Eclipse Plugin

The use of Web-CAT & EclEmma as a computational platform has some flaws. These flaws are that the process of using code coverage as a metric to assess software engineering is not always beneficial and accurate. The problem with test coverage is that 100% coverage does not mean 100% tested. Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other doesn’t.

**Algorithms**

Software firms are becoming more advanced and capable every day. Computational intelligence (CI) is an offshoot of artificial intelligence (AI) and can be used to measure and analyse the software engineering process. In CI the emphasis is placed on heuristic algorithms such as fuzzy systems, neural networks and evolutionary computation. Heuristic algorithms are a method of solving a problem faster and more efficiently than classic methods. The expression CI refers to the ability of a computer to learn a specific task from data or experimental observation. Generally, CI is a set of nature-inspired computational methodologies and approaches to address complex real-world problems to which mathematical or traditional modelling can be inadequate. Many real life problems cannot be simply translated into binary language in order for computers to process it. CI can therefore provide solutions to such problems.

**Heuristic Algorithms**

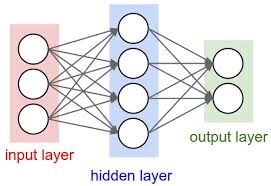
* **Fuzzy Logic**

The idea of fuzzy logic was first advanced by Dr Lotfi Zadeh of the University of California at Berkeley in the 1960s. Computers traditionally operate using binary logic where variables can only take the values 0 and 1. However, natural language cannot be easily translated into the absolute terms of 0 and 1. Therefore, fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1. Fuzzy logic seems closer to the way our brain works.

* **Evolutionary Computation**

In Computer Science, Evolutionary computation is a general name for a group of problem-solving techniques whose principles are based on the theory of biological evolution, such as genetic inheritance and natural selection. These techniques are applied to a variety of problems, from practical industry applications such as analytics and prediction algorithms to leading-edge scientific research such as protein folding.

* **Neural Networks**

Neural networks are computing systems inspired by the biological neural networks that make up the human brain. Neural networks are made up of learning algorithms and progressively improve their performance, the more data it analyses the more accurate it becomes. They can excel at fault tolerance compared to traditional networks. Neural networks are typically a group of nodes organised in layers. One layer will process the input, another will control the signals and one will control the outputs. Some examples where neural networks have been used in significant commercial applications since 2000 include handwriting recognition for cheque processing, speech-to-text transcription, and facial recognition.

**Relevance of Computational Intelligence to Measuring Software Engineering**

CI can be used in the future to measure the performance of software engineering, as there are clear benefits to it. Comparing software engineers under set metrics is not always beneficial. Fuzzy logic and neural networks can be utilised to tailor the metrics specific to each software engineer. Models such as neural networks progressively improve their performance i.e. the more data it analyses the more accurate it becomes. Therefore, CI can be used to analyse inputted data that is too much for the human mind to comprehend. As a result large numbers of software engineers could have their performance, under tailored metrics, analysed. Through the use of CI the quality at which software engineering are assessed and measured would improve. This should lead to better decision making and increased efficiency when assessing a software engineer’s performance.

**Ethics**

It is clear that there is a comparative advantage of assessing and measuring the performance of software engineers. It allows companies the ability to track how efficient and beneficial a software engineer can be to a given project. However, as we can see from the data that can be measured, there is no clear cut way for a company to measure performance effectively. Some metrics may been seen as beneficial and effective in the comparison of two software engineers initially but these metrics often have flaws. There is an ethical concern with this assessment and analysis; where does the measure and assessment of a software engineer’s performance lead to that engineer’s personal privacy coming into question? Should an employer constantly be able to review and scrutinize an employee’s performance? Should an employee be constantly under pressure to outperform their peers as their performance is compared against another employee’s performance?

**What to Avoid?**

From an ethics point of view there are certain measurement metrics which I feel need to be avoided. Two of the main metrics I feel that need to be avoided are measuring lines of code, and commit count. Measuring a software engineer through the process of measuring the lines of codes used is unethical and is a poor way to portray a developer’s performance. Comparing programs by the number of lines of code would be similar to comparing essays by the number of pages. One essay may only be 4 pages long but be concise, well written and to the point. Another may be 10 pages but full of waffle.

Commit count is another measurement metric that I feel should be avoided. The commit process to GitHub can be used to check whether a developer is constantly updating and working on their code. The measurement of commit count leaves a glaring issue if used as a measurement metric. The reason why it is a poor measurement metric is as follows; Developer one may commit regularly giving the assumption that they are constantly advancing their project. However, developer one may only be making small changes that don’t have a large contribution to the overall completion of the problem. Developer two may commit not as often, but their commits may be more fruitful and may represent large advances in their code. The problem with counting commit frequency is that developer one could be seen to be progressing their project at a faster rate than developer two, which may not be the case.

**Are Some Analytics Intrusive & Unethical?**

When a company set outs its performance metrics it should be wary that the privacy of their employees does not come into question. Most software engineers will accept that their progress is being analysed, often for their own benefit, however, transparency from the company is key. Some monitoring analytics may be legal but can be intrusive and unethical. An example of where a monitoring technique is legal but unethical is one where legal employee computer monitoring could inadvertently show employees’ sensitive data i.e. sexual orientation, medical problems and political views. A company may monitor its employee’s internet use. This monitoring could lead to information about the web sites an employee has visited leading to that employees’ medical history being unveiled. To avoid both the crossing of ethical and legal lines, the policy of not browsing the web for personal purposes could be negotiated with the staff. If this is negotiated, the company must be transparent to the employee about what it will be monitoring, as to avoid inadvertently obtaining private information.

**Bibliography**

1. Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology - Gregor Grambow and Roy Oberhauser
2. http://istqbexamcertification.com/what-is-test-coverage-in-software-testing-its-advantages-and-disadvantages/
3. <http://searchnetworking.techtarget.com/definition/neural-network>
4. <http://ukci.cs.manchester.ac.uk/intro.html>
5. <http://whatis.techtarget.com/definition/fuzzy-logic>
6. http://www.businessdictionary.com/definition/lead-time.html
7. <http://www.cs.umd.edu/~mvz/mswe609/book/chapter3.pdf>
8. <https://arxiv.org/ftp/arxiv/papers/1301/1301.5964.pdf>
9. <https://en.wikipedia.org/wiki/Artificial_neural_network>
10. <https://en.wikipedia.org/wiki/Computational_intelligence>
11. <https://en.wikipedia.org/wiki/Evolutionary_computation>
12. <https://en.wikipedia.org/wiki/Fuzzy_logic>
13. <https://en.wikipedia.org/wiki/Heuristic_(computer_science)>
14. <https://kanbantool.com/kanban-library/analytics-and-metrics/explaining-cumulative-flow-diagrams>
15. <https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams>
16. <https://visualstudiomagazine.com/articles/2013/05/01/neural-network-feed-forward.aspx>
17. <https://www.investopedia.com/terms/l/leadtime.asp>
18. <https://www.qasymphony.com/blog/64-test-metrics/>
19. <https://www.worktime.com/is-it-legal-and-ethical-to-use-employee-computer-monitoring-software/>
20. P.M. Johnson et al., “Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined,” Proc. 25th Int’l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.