Name: David Scolard
Student Number: 16322277

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

Measuring Engineering

Objective – “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 **ethical concerns** surrounding this kind of analytics.”

Introduction

The measuring of software engineering is almost as old as software engineering itself. Employers who may not always have understood the technical aspects of a project, sought out ways in which they could analyse and compare the productivity of their engineers. In this piece we will examine the different approaches taken in regards to this analyses and the accuracy of its yield. We shall also delve into the ethics surrounding the topic as engineers find themselves ever more increasingly under the critical observation and examination of their employers.

The measuring of engineering can take many forms, from simple analysis of the number of written lines of code an engineer produces, to much more complicated algorithms which can take into account a wide range of variables. We will look more at these algorithms later in this piece. The managerial decisions which result from these analytics is the source of much debate. Would it be fair to say Bill isn't as valuable an employee as Mary because he wrote 75% as much code as she did? It's well within the realm of possibility that Bill spent half his day helping his co-worker Sarah solve a problem, where is this taken into account? This is why we must be careful with such analytics and our resulting decisions.

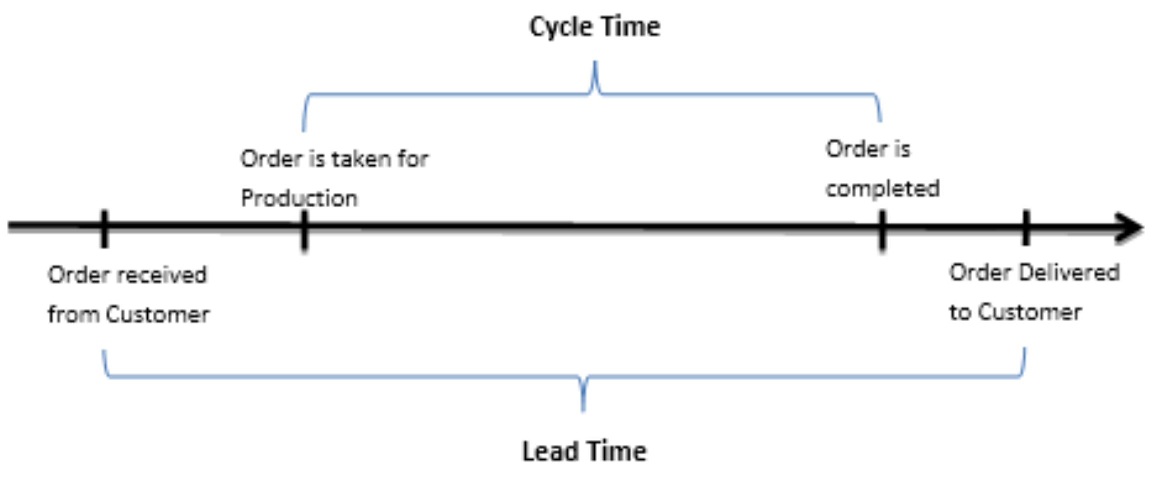
At what point are employers becoming too invasive. It's all well and good to measure how many lines of code an engineer produces, however is it ok to insert a pressure pad in their chair to measure how mow many minutes they spend at their desk per day? Or perhaps make them to scan themselves in and out of the bathroom so we can measure the duration of their defecation? These suggestions may seem like a crude joke in 2018 however the future is always much stranger than we can imagine.

Measurable Data

The first instances of measuring engineering and productivity date back to the late 1960's. Back then the predominate method of measurement was simply Lines of Code or "LOC" for short. They would often refer to KLOC's as well as a thousand lines of code. Companies would measure a programmers number of defects per KLOC. Using these two easily measurable data points they would examine an engineer's productivity level. This system has many obvious drawbacks. For example an experienced programmer can produce the same result using many less lines of code than a novice programmer, the novice is no more productive just because he typed out 100 lines of inefficient code. With the introduction of higher level programming languages in the 1970's the LOC method of measurement became obsolete. The added complexity and functionality of these languages called for a re-thinking of how we could measure productivity. Ten lines of C code can achieve a whole lot more than ten lines of assembly code and so they are hardly comparable.

The emergence of GQM(Goal-Question Metric) in the 70's is accredited to Victor Basili and colleagues of the University of Maryland. This method of quantifying productivity focuses on a goal based system, Basili argued that a metric lacking objective goals was destined to fail and the only true way to measure productivity was to define measurement goals for all employees to work toward. GQM was generally met with high appraise. GQM only measures those metrics which are relevant to the specific goals, this provided clarity in objectives for engineers and achieved greater productivity output than previous models. Although in academic circles GQM was noted to be more successful than the original LOC method, it took some time before industry practices adopted the new measurement. Although LOC was a crude method, it was easily implemented and analysed. One major problem for GQM was the lack of detailed guidelines in the industrial environment on how to set up and maintain a GQM system. Another hurdle was the fact that the selection of goals was a subjective thing, an individual could skew results by selecting easier, or tougher metrics.

Companies have mostly moved away from these traditional metrics. Now we increasingly come across terms such 'Lead-Time' and 'Cycle-Time'. Lead-time measures the time it takes a team to go from accepting a project to delivering the finished goods. Using a team's median lead-time one can make a very good *estimation of delivery date on a new project. This type of metric is essential for companies managing tens or even hundreds of teams. Cycle-time is much like lead time, except cycle time measures the actual effective completion rate per project. Essentially lead-time is what people see from the outside, whereas cycle-time would be known by the team members who may accept multiple projects simultaneously or perhaps work off a backlog of projects. These metrics can be critical for teams dealing with eager clients who need an estimated delivery date. The team can simply compare the project to their lead time's for projects of a similar size. If 95 percent of projects took less than 6 months then they would feel comfortable providing that as a provisional delivery date for the project. Alternatively within a large company lead times of different teams working on similar sized projects can be analysed by managerial staff to get a clear picture of which of their teams are most productive. Lead Time vs Cycle Time:*


Cycle Time
Order is taken for Production
Order is completed
Order received from Customer
Order Delivered to Customer
Lead Time

Credit: https://www.whatissixsigma.net/cycle-time/

Other metrics such as defect density and defect removal efficiency(DRE) track bugs in our code. Defect density is measured as the number of defects per KLOC. 'DRE' is the measurement of how quickly your team respond to, and repair bugs which emerge in the code. One can calculate the defect removal efficiency rate with the following formula:

Number of defects found in development = A,

Number of defects found after development = B,

*Rate = (A/(A+B))*100*

This formula can give you a very good estimation of a team's ability in predicting and testing for possible bugs within their code before release. The higher percentage the rate the better, low rates indicate short sightedness and lack of thorough testing in development. On the topic of testing in development the obvious metric here is code coverage. Without fail engineers should be achieving 100% code coverage through their testing before completing and implementing a project. This is essential for catching any unforeseen bugs.

Computational Platforms

There are many platforms which can aide in the monitoring of development. These tools are available for everyone including researchers, practitioners and educators. These metrics can be of great affect for long range research initiatives such as collective intelligence which requires collective efforts and competition of many individuals to gather data for decision making. Practitioners can avail of the information on these platforms to use for quality assurance, project planning and resource management. Educators similarly make use of these platforms within software engineering courses to introduce students to the concept of software measurement.

'Hackystat' is an example of such a platform. Users of Hackystat are able to invasively track and store raw data about their development process on Hackystat's web service called Hackystat SensorBase. Through SensorBase the user is able to generate visualizations of their raw data to be easily analysed and queried.

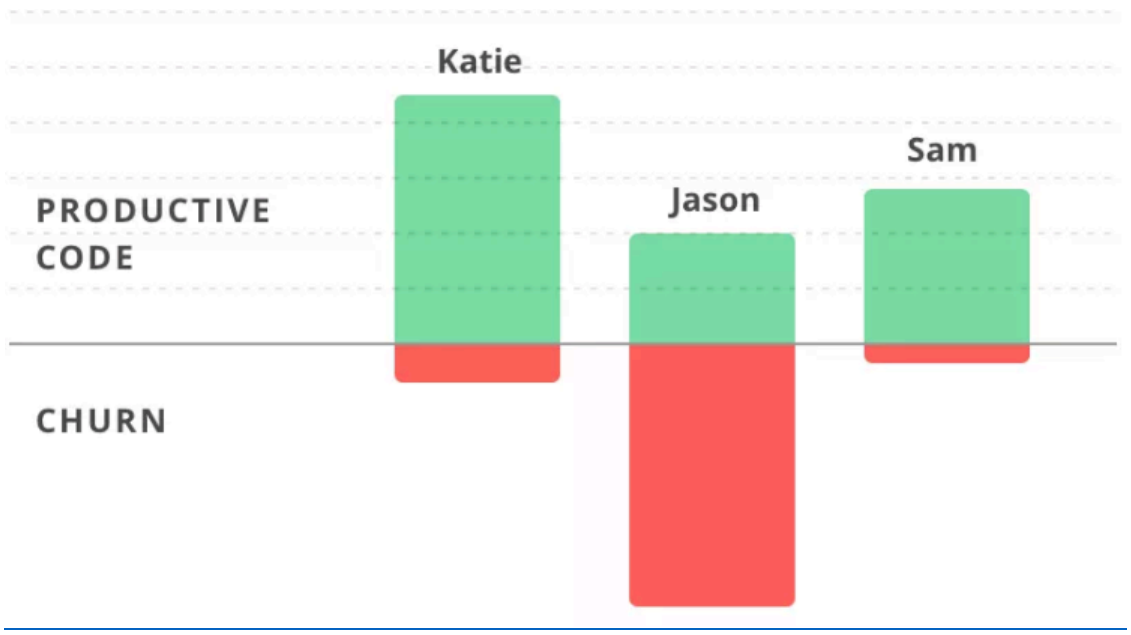
PSP – The Personal Software Process is a framework designed for software engineers to work within. This framework has set methods which instruct engineers on how they should plan, measure and manage their work. PSP helps engineers in the tracking of their predicted and actual development of code as well as reduce the number of defects in their work. PSP is normally used within a TSP team environment which is designed to help teams with similar challenges. Using PSP process methods helps TSP teams produce improved high quality software. The quality of the software output is measure in terms of defects, the main objective of PSP is to reduce the number of defects on release. PSP is certified by the SEI at Carnegie Mellon University.

Algorithmic Approaches

The software development metrics we recover are often fed into complex algorithmic systems in order to generate a deeper understanding than is possible by human examination.

Algorithms must tackle the task of predicting the ultimate quality of the end product. One way of doing this is by calculating the "Code Churn". Code churn measures the rate of change in your code, this includes lines added, lines deleted and lines of code modified over a time period. These metrics can be analysed to give some valuable insight into the development process. For example if the project is nearing its release date and you begin to see a spike in code churn, then alarm bells should be ringing in your head. A project should not become volatile towards the end, in contrast it should become more stable and code churn should steadily decrease. Code churn is a terrific tool for visualising and assessing the development process in real time

Code churn can also be used to assess individuals. High levels of code churn are normally a bad sign for any programmer. A team leader is never going to be happy to find you spent the majority if your time writing code which you subsequently deleted and replaced. To analgise this we can compare it to somebody writing out a letter to a friend, they decide they don't like the wording so they rip up the letter and rewrite it. They have written two letters but ultimately their friend only receives and cares about one letter. Its inefficient and wastes time, for this reason an individual's 'productive code' is much more important than their LOC number.
Here I found a good example diagram of churn productivity :

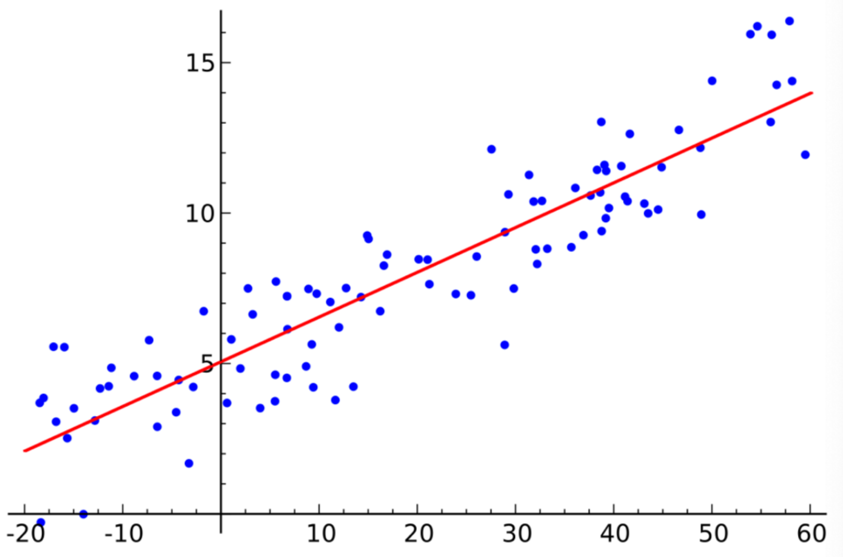


Credit: https://blog.gitprime.com/why-code-churn-matters/

As we can see above Jason has written the most code out of anybody here. This is of little consequence however as Sam has contributed more 'productive code' to their project, therefore Sam is clearly a more valuable team member than Jason. Code churn is clearly an extremely valuable metric for anyone wishing to measure the software engineering process in a meaningful way.

The application of machine learning and AI can also yield great results. Machine learning allows the program to grow and learn from the data over time, analyse trends, and predict outcomes to effectively influence business decisions. Machine learning given the appropriate data can construct a complete image of a developers skills. Using past data it can predict anything from how long it might take a given developer to complete a task, to the quality of their end product and the number of defects it may contain. Complex techniques such as regression and supervised clustering are deployed in machine learning. Clustering attempts to analyse data by grouping data points. The clustering algorithms will assign each data point into a specific group with points of similar properties. It is a main task of exploratory data mining and a common technique for statistical data analysis.

Regression is used for forecasting and finding out cause and effect relationships between variables. Simple linear regression is a simple example, it takes a number of independent data points and draws a linear relationship between them which best summarises the pattern. Linear Regression:



Linear Regression

Ethical Concerns

The future of measuring engineering is simultaneously fascinating and worrisome. With a greater understanding of the engineering process we can put into action methodologies which can greatly improve the productivity off engineers in the workplace. This is undoubtedly beneficial, however on the other hand as our employers gain more and more access to information illuminating every action of ones working day we must step back and assess the social and humanitarian implications of this.

The gathering of empirical metrics on employees is dehumanising by nature. The last few years have seen a notable increase in the layman's awareness of data collection and the lack of privacy they are afforded online. This has become a giant worry for people across the world, whistle-blowers such as Edward Snowden have brought forth the solid evidence proving what many had already suspected. Nothing is private anymore, your phone calls, texts messages, what you buy on amazon, what you like on Facebook, even your GPS location is being collected. Recent European privacy laws such as the General Data Protection Regulation (GDPR), have helped with transparency forcing companies to be clear and upfront *with users about what is going to be collected before they are allowed to use any service.*

So we have to ask what is the difference between these multinational companies gathering our data and our employers doing the same thing. The laws for ethical data collection can become a lot more complicated when it comes to an employer/employee relationship. There are well understood and tangible productivity benefits, which equate to financial benefits, that come from this data collection. Any refusing to allow their data be collected might not find themselves getting a pay rise or a promotion any time soon. Therefore people are more likely to keep quiet. There is also the case to be had that while you're at work you're an acting part of the company, anything you do and say represents the company and is part of their entity as a whole. As an employee of the company you should want it to progress and be successful and if the data collection is a necessary part of that then you should do your part and allow it to be collected. In my personal opinion the data collection is completely warranted as long as it has direct and clear business relevance.

Conclusion

We can see that within the software industry the measuring of engineering will play a significant and vital role for companies as they try to squeeze as much as they can from their employees. It has come a long way from the days of simply measuring Lines of Code and our work is under ever more scrutiny. The financial implications of this are undoubtedly positive but the welfare aspect is yet to be seen which way it will go. Companies such as Hackystat are set to do well in the current climate. As more and more companies adopt robust methods of measuring engineering, others will have to do the same to remain competitive. Undoubtedly when I finish college in 2020 I will enter a workplace where my every act is logged and collected for analysis, we will have to wait and see how I feel about that.

Reference List

- https://hackystat.github.io/
- https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283
- https://blog.gitprime.com/why-code-churn-matters/
- https://www.simplilearn.com/data-science-vs-data-analytics-vs-machine-learning-article
- http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf
- https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a
- https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68
- Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.
- https://www.wired.com/story/europes-new-privacy-law-will-change-the-web-and-more/