Measuring Engineering

The field of software engineering was first created in 1968 due to a phenomenal rise in software work and a need for collaboration between software engineers. That was almost 50 years ago and yet the field is still considered to be in its infancy with new methods and technologies changing aspects of software engineering every year. Rapid development has brought increased knowledge of how we should be measuring work and also how best to program, maintain and update software projects.

The goal of this report, is to discuss the following :

1. What data is available and what should we be measuring, along with matching data to projects.
2. Computational platforms that have emerged to make the process of software engineering easier and more accessible
3. Algorithmic approaches to deal with all this data
4. The ethics surrounding the aforementioned analytics.

Measurable Data

“without data you're just another person with an opinion” - W. Edwards Demings

Data is undoubtedly and fundamentally the most important component when trying to measure the software engineering process. In today's world there is absolutely no shortage of available data and numerous ways in which to interpret it. That being said, it is also extremely easy to get lost in this sea of data, fail to find the right starting point and miss key points. Therefore, it is crucial that when measuring the software process, planning of data selection is carefully considered as this piece of the process is as vital as the data itself.

In this section I will consider the main areas that should be assessed in the software engineering process. I will look at how data can be structured and the different types of measurable data that affect the software engineer noting what they can do and how they can help us further the software engineering process.

Data analysis requires a plan, which for the software engineer must include these fundamentals:

- Productivity levels
- Impact/Effectiveness,
- Consistency
- Communication.

Productivity:

Productivity should be considered under headings that allow the analysis and monitoring of output. The objective is the production of working projects in a fast timeframe. To determine this it is important to find out how much work developers do and how much of their time is wasted idling instead of implementing. Examples to look at would be lead time and churn.

Impact/effectiveness:

The effectiveness of an engineer in implementing solutions can be measured by the changes to code made by the developer. It can also be measured by the tests they create and the debugging changes they implement.

Consistency:

Consistency of a developer is crucial in the development process as it engenders confidence in their reliability and dependability. It is important to know that developers can ‘produce the goods’ time and again in a consistent timeframe. This consistency is a measure of their skill and indicates potential to adopt other projects. A lot of this can be measured with commit dates, active days and comparing finished projects to project plans.

Communication:

Good communication skills are arguably one of the developers most important assets because communication is vital in team dynamics and is highly valued by employers.

Without predefining these headings on what we want our developer to be it makes us essentially blind when looking at the actual data . Measuring for example a “perfect” developer may not give the best results for software engineering practice. Yes it will show us how that specific developer is fantastic but it may skew information if it is not compared with other factors. For example if that particular developer is constantly meeting deadlines but has extremely poor communication skills, we may take this as a sought out quality when we all really know we shouldn't

Types of measurable data

- Source lines of code
- Number of commits
- Lead time
- Churning
- Bug fixing
- Communication (ie. slack)

Source lines of code (SLOC)

This is a software metric that is used to measure the size of a program by counting the lines of source code. This simple concept will give a rough idea of how much work is being done by the developer. It has also been split into two categories; Physical and logical. Physical being the numbered amount of lines while logical attempts to condense data into statements. This can be problematic as it can differ from language to language, for example, in c every time a semicolon is used it is counted.

Physical SLOC are far easier to measure and therefore more efficient in measuring work done, but often are not reflective of condensed code. To make use of this data effectively it would be wise to use it along side commits and the impact of the developer on the code. KISS (keep it simple stupid) is the advice du jour that has been advocated by my lecturers since commencing my degree and makes sense in this scenario..

Commit Count

This refers to any commit or change made to a project by the developer. The number of commits is a very good indicator of who is doing work and how much is being done. Although committing regularly does not define a software engineer as excellent it is still very good practice and a habit to be recommended. Should anything go wrong with any systems your work is always safe if you commit your projects regularly.

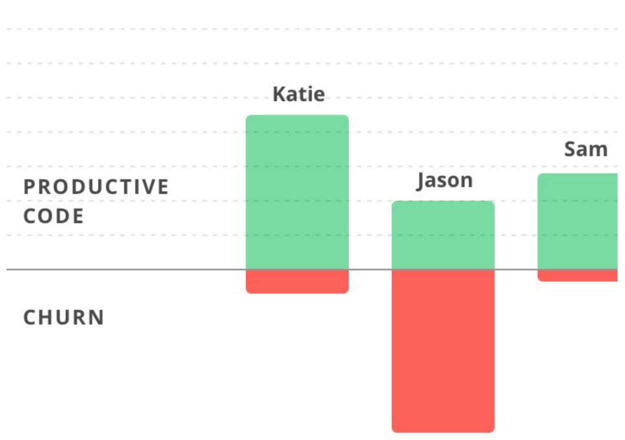
Lead Time

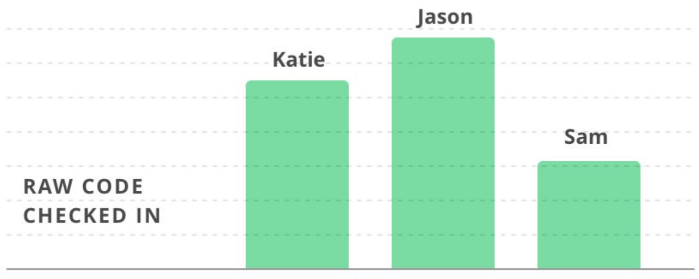
Lead time put simply is how long it takes developers to go from idea to product. Although this can be difficult to track some of the time it is worthwhile because the more it is observed the better the estimation time of future project completion.

Code Churn

Is the percentage of work done by the developer taking into account the edits and deletions that may have taken place. It is typically measured as lines of code (LOC).

Think of it as writing a postcard and then tearing it up and writing it again, and then again. Yes, you technically wrote three postcards, but in the end only one was shipped so we’re really talking about one postcard worth of ‘accomplishment’ from all that effort. The same is true with code. Examples below.





As you can see The amount of code written in this case by each member was no indication of how much productive code they contributed which is what makes churn so powerful to analyse. Churn rate can also help to identify problems with individual developers. For example, a sudden increase in churn rate may indicate that a developer is experiencing difficulty in solving a particular problem or is repeatedly polishing a feature that’s ready for release. A high churn rate may also mean that a developer is under-engaged. Other causes of high churn include an indecisive product team that has the developer running in circles.

Bug Fixing

This metric involves the measuring of how much time an engineer spends fixing bugs. Software is seldom 100% perfect and the number of bugs varies from project to project. The time involved in bug fixing includes both the time it takes to troubleshoot the problem and the time it takes to fix it. According to Vlad Givetrts, head of Engineering for Clara Lending, if dealing with bugs takes more than 20% of a software engineers engineering time, there is a definite problem with either quality of code or the architecture system in place.

Level of Communication

Put simply this refers to how well a developer is working with their team. Monitoring their team chat platforms, such as the application slack, to see how often the team are conferring about problems and making commits to the project. Measurement in this area is growing and is being used in big office spaces to make them feel small and boost productivity while also generating conversational data to achieve a better indication of the project's direction and progress. Some companies such as Humanyze have even gone as far as to include a microphone in the employee's badge to track data and store it on their data base.

It is important to note that a lot of big companies are also searching for various qualitative features in new developers that often have no real way of being measured. For example Google looks for intelligence and experience in developers along with something they refer to as “Googliness”. This quality they claim is difficult to define but very easy to spot. Essentially they need someone who has a personality that is a good fit for the company along with fantastic skills as a developer and communicator.

Computational Platforms

Once it has been decided where and what needs to be monitored it is essential to have a computational platform to gather and display the information.

Examples of computational platforms.

- Git
- PsP
- Hackystat
- Codacy

Git:

It would be an oversight to write this report and not mention the platform students have become very well acquainted with. Everything submitted into git is recorded, processed and automatically scaled for us into graphs an example of which is Github. Github records all significant actions and stores it on their log “github archive”. They provide the facilities to create graphs which present information such as push and pull activity as well as recording what type of information is being committed in relation to work done. It is a team facilitating platform which does an excellent job in combining work so that everyone is working off of the same code.

PSP

The Personal Software Process (PSP) is a software development process that is created to help software engineers better understand and improve their performance. It does this by keeping track of their predicted and the actual development of their source code.
Watts Humphrey created this platform to apply the underlying principles of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer. The idea behind this platform is to give software engineers the process skills necessary to work on a team software process (TSP) team

Hackystat

Hackystat is an open source framework that automatically collects data and analyses the project data and software engineers. The aim of the Hackystat platform is to provide a mechanism that is extendable that can thoroughly reduce the overheads associated with collection of a wide variety of software engineering data. Alongside that, the platform contains a toolkit of analyses that can make a useful report. Hackystat is widely used in application areas. This can include classroom pedagogy, software engineering of high performance computing systems .

Codacy

A technology start-up founded in 2014, Codacy offers an automated code review platform. Codacy software can be installed on premises or accessed in the cloud. It is used by developers to check the quality of code, and implement code quality standards. Their product offers metrics such as the Churn, complexity, duplication and number of lines of code. It also calculates code coverage, offers a project tracking and evaluation system, and gives developers feedback on security concerns and code style violations.

Codacy analysis is flexible, and metrics can be customized to compliment certain projects. Codacy co-founder Jamie Jorge recently claimed "With Codacy, we estimate that we help developers optimise around 30 per cent of their code review time". On a more tangible basis, this corresponds to delivering software two weeks ahead of schedule.

Firms like Codacy – who offer Analytics as a Service - are the future of measuring and improving software engineering. As the amount of data generated and collected continues to increase, it is more efficient and realistic for firms to pay for this service than to exhaust their own resources doing these computationally intensive tasks.

The algorithmic approaches available

In view of the development of these new platforms, you may wonder how it is possible to keep improving year on year de to measure. The answer is Algorithms. All around the world people are writing algorithms to make once tedious manual processes more automatic or in a lot of cases completely automated making the machine do the work for us (Artificial intelligence).

Heuristic Algorithms

Fuzzy logic:

First introduced in the 1960s fuzzy logic was a pioneer in our understanding of logic. Computers traditionally operate on a binary system using either 0 or 1. This doesn't always translate well in the real world as a lot of decisions made by Humans and animals are fuzzy, rather than being clearly defined in terms of a simple yes or no. Fuzzy logic allows for many logic values in which the truth values could range from anywhere between 0 and 1. This definitely reflects human's brains better than a binary option.

Neural Networks:

Neural networks are computing systems inspired by the biological neural networks that make up the human brain. The basis of a neural network is a lot of learning algorithms that progressively improve the more they are used and the more data they collect. They excel at fault tolerance compared to traditional networks. Neural networks are set up as a group of nodes organised in layers. One layer for input, one for control signals and one for 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.

The Uses of Computational Intelligence

This is defined as an alternative to Artificial Intelligence in which the emphasis is placed on heuristic algorithms.Unlike Artificial Intelligence, Computational Intelligence does not aim to implement humanity into machines. Although it focuses on computers following a logical decision making process like we do, it does not attempt to give machines emotional intelligence.

Comparing software engineers under set metrics is not always beneficial. The use of CI heuristic methods would yield better results in the long term. Neural network models progressively improve their performance i.e. the more data they analyse the more accurate they become, allowing CI to analyse vast amounts of inputted data that would be too much for the human mind to manage. In light of this it seems logical to employ CI in the assessment and measurement of software engineering. It should lead to better decision making and increased efficiency when assessing a software engineer's performance.

<u>Ethics</u>

With any new discovery there will be new ethical questions to answer. In terms of measuring the software engineering process and the developer two major ethical concerns come to mind, Privacy and Security.

Privacy:

Privacy is at the forefront of the ethical debate at present. Questions are arising amongst developers about the extent of monitoring and when it should stop? What are the benefits and drawbacks associated with monitoring? Is monitoring everything a developer does really necessary or is it just adding needless pressure, requiring a developer to be on best behaviour 100% of the time.

Feedback from HackyStat (previously discussed) gives a great example of the privacy concerns of the developer, who objected to the automated nature of the data collection, and raised concerns about how much data was being collected. This highlighted the amount of data that was being collected and the amount that was being submitted. The term “Hacky-stalk” was coined by one developer to describe the software.

Companies have also taken to tracking tabs and logs of all work computers, which is creating a similar ripple of unhappiness amongst workers. This has been acknowledged by big companies such as Google who have now adopted a more lax approach to mitigate against unrest in the workforce.

Security:

Security is arguably the biggest concern in today’s global love affair with the online world. Leaked data is a real concern and a strong possibility with the sheer amount of data that is being logged by developers.

Ethically, having all your information leaked by your employer is a huge invasion of your privacy. In recent times it has become clear that it doesn't matter how big the company you work for is, anyone can become a leak target. This was evident in the Yahoo account leak, which involved more than three billion employee and user accounts. In spite of the fact that Yahoo is a massive online business with significant employee numbers it was unable to prevent leaks highlighting how insecure the net currently is. Data gathering may be valuable in determining what makes a great software engineer however it has the potential to create huge security and ethical concerns for everyone involved.

Conclusion

To conclude, the measuring of software engineering has come a long way and many new and innovative ways of measuring and improving these processes have been developed. With this ocean of information and technologies to monitor it, determining where we will be in the future is difficult to say.

Will ethical concerns halt progress or catapult research forward providing us with the means to make the perfect software engineer? It’s difficult to make that call right now and Ironically I think that more research is required which necessarily involves ethical data gathering, storage, analysis and interpretation for us to find out.

References/Bibliography

http://engineering.kapost.com/2015/08/you-can-and-should-measure-software-engineering-performance/

https://en.wikipedia.org/wiki/Software_engineering

https://en.wikipedia.org/wiki/History_of_software_engineering

https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/

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

https://en.wikipedia.org/wiki/Source_lines_of_code

https://medium.com/letters-from-slash-hyphen/how-ibm-is-using-slack-to-persuade-software-engineers-to-socialize-72731a642319

http://snap.stanford.edu/class/cs224w-2013/projects2013/cs224w-033-final.pdf

https://en.wikipedia.org/wiki/Personal_software_process

Continuous GQM: An automated measurement framework for the GQM paradigm (Diplomarbeit von Christoph Lofi)

https://www.researchgate.net/publication/228731372_Experiences_with_hackystat_as_a_service-oriented_architecture

https://en.wikipedia.org/wiki/Neural_network_software

http://www.independent.co.uk/life-style/gadgets-and-tech/news/yahoo-hack-details-personal-information-was-i-compromised-affected-leak-a7981671.html