Software Engineering Report

Measuring The Engineering Process

Richard Noonan | 15320499

"A good programmer looks both ways before crossing a one-way street"

-Unknown

Abstract

In this report, I will give an overview of the various ways of measuring the software engineering process as a whole with regards to data from the process itself, the platforms available to analyse the process, the algorithms used and the ethical implications of applying metrics to the process.

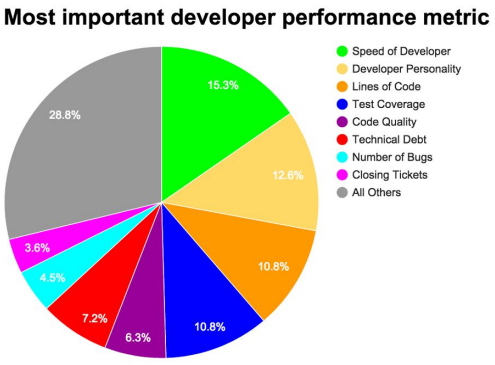
1.Measurable Data

1.1 Overview

When you talk about measuring data from software engineering, it is important to first discuss what is the most important piece of data attributed to the software you're producing. If it is imperative to get something working quickly without much regard for quality, speed of production is the most important measurable factor. If you need someone to be cooperative within a team, how well they interact and collaborate with their colleagues could be an important metric for that specific project. In the following, I will go through different data metrics that can be used to determine the the proficiency of a software engineer and instances where that metric is the best one to extrapolate insights from.

1.2 Speed

In a survey done by the website medium.com of 300 software developers, speed came out as the number one metric for measuring the performance of a software engineer(Medium, 2017). Speed is an important metric regardless



of the project as there will never be an inexhaustible period of time in which to complete a project. Clients awaiting the implementation of a piece of software have to have some time frame to reference.
We can look further into what actually encompasses measuring speed as a metric and what it looks like when applied to the real world.

- Leadtime: Leadtime is the time between drawing up the idea and the finished implementation of the software. Used as a measurement for projects where producing something that works quickly is paramount. Although this is a useful piece of data, it is only data produced on the team developing the software as a whole and not on any individual within the team. It is also hard to determine how to interpret this data as it only pertains to a specific project and it is difficult to draw comparisons between other projects.
- Cycle time: This is the amount of time it takes for a piece of software to modified and delivered back into the software.
- Open/Close rate: How quickly a reported problem is resolved.

1.3 Lines of Code

Using lines of code (LOC) to measure the engineering process can be an arbitrary metric. A developer can be committing a large amount of LOC per day but could be producing sub standard product. If the emphasis is on pushing as much code out as quickly as possible, quality could fall by the wayside as the focus isn't on producing quality code, just lots of it.
If you wish to access and compare one process with another, it doesn't make sense to draw conclusion based on LOC if the projects are using different languages for example java vs ruby. This is why using this metric is quite limited and should be used with caution.

1.4 Security

Security is an important metric to measure of any software system as it underpins its reliability and safety when implemented.
Security of a piece of software can be measured under two headings:

1.4.1 Software Vulnerability: This is a mistake in the software that can be exploited by someone to gain unauthorised access to the system or run malicious software.

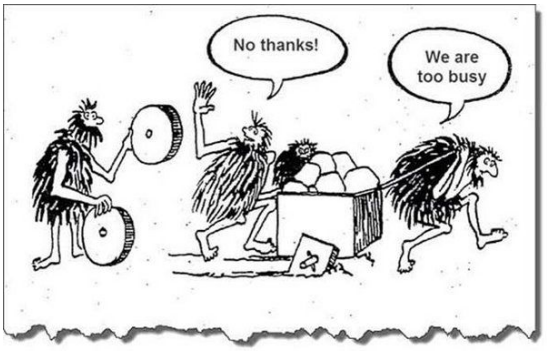
1.4.2 Software Weakness: This is the general bugs, flaws in the code, vulnerabilities and all other errors that can be made when implementing the software. Software weakness includes software vulnerability.
The aforementioned are important for any software system and need to be considered at length when the solution is being designed. A systems weakness is good basis for comparison when assessing the software engineering process as tests can be designed that are specific to the software being tested.

"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite."

-Ward Cunningham

1.5 Technical Debt

Technical debt is the accumulating time deficit that is built up by implementing short term solutions that will be needed to be fixed at a later date instead of designing a more elegant solution that address the problem as a whole.



In my opinion, technical debt is by far the most important metric when assessing the software engineering process. Other metrics like speed and LOC are good for measuring the activity of a software process and determining if code is being committed but as far as determining the quality of the overall implementation, technical debt is a far superior metric. This is because the onus is placed on designing long term and well thought out solutions, not producing quick fixes that only work in the short term. According to techdebt.org, for every 1000 LOC written, 5h 48m is the technical debt accumulated. That is it'll take 5h48m to rectify the mistakes made in the short term (Visual.ly, 2017).

	Comparable	Measure activity	Client Orientated	Quality Orientated
Speed	No	Yes	Yes	No
LOC	No	Yes	No	No
Security	Yes	No	Yes	Yes
Technical Debt	Yes	Yes	Yes	Yes

1.6 Why do we collect this data?

Collecting this data is useful only when used to answer the right questions. What I mean by that is, in isolation, collecting the aforementioned data is effectively useless if it isn't used in a practical sense to gain insights into how effectively we are producing quality viable solutions for clients. Essentially, only when this data is used to answer business questions does it give the user a comparative advantage. For example if a manager was collecting data on their subordinates, the data could be

used to establish how well they were each contributing to providing solutions and fulfilling their role on the project, giving the manager the advantage of favourable discernment.

Internally, a combination of these metrics can be used purely for comparative reasons to analyse an individual and to contrast them to their peers. Useful insights can be gained to how a person is performing, within a group for example, compared to how they performed within another group. Although this isn't an exact science and there are a lot of unknown variables in these situations, the more detailed, accurate and varied your data on their software engineering process is, the better the chance is you'll be able to make informed and correct decision.

2. Computational platforms

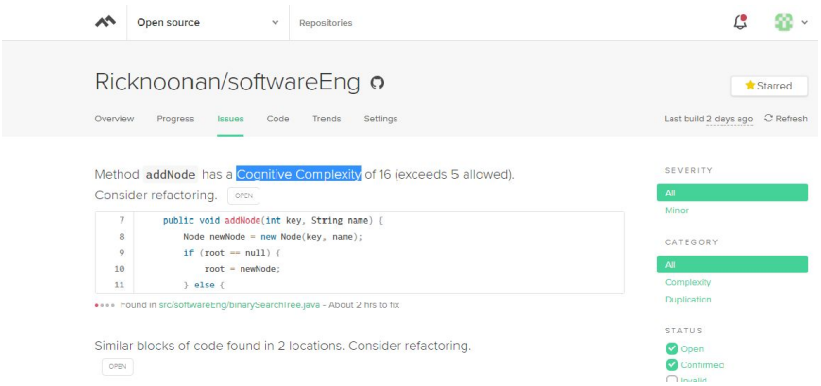
2.1 Overview

With the measuring of data and the comparisons between the different metrics discussed, it is now time to look at the different computational platforms out there that can perform and automate this level of analytics. In the following I will look at three companies in particular, Code Climate, Scrutinizer and Code Beat, and will consider what each one offers to its users.

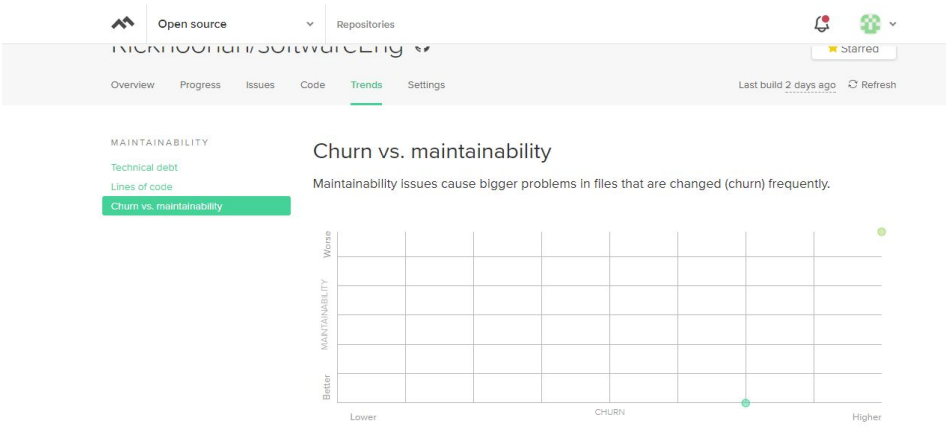
2.2 Code Climate

Code Climate is the most popular platform used analysing over 2 billion lines of code daily and used by over 100,000 projects (CodeClimate.com, 2017). It is an open source platform with a very attractive user interface and is extremely easily to navigate. But what do they offer to your project?

2.2.1 Issues identifier: One testing the platform with my own github repository I noticed a couple of snippets had been marked with a cognitive complexity rating. This is how difficult it is for a person to read and understand your code. Also displayed is code snippets where refactoring is applicable. For each of these issues, an estimated time to fix is also given.



2.2.2 Trends: Code Climate gathers your code, analyses it using various metrics and displays this data using clear graphs and diagrams. The metrics they use are: technical debt, lines of code and churn vs maintainability. Each of these metrics are elegantly displayed, giving the user a good overview of their code and if there are some negative trends visible in their code. An interesting metric that they present, and one that I didn't consider, is churn vs maintainability. This is how frequently a file is changed versus how difficult it is to maintain that file. Maintainability could be considered an abstract metric but Code Climate lay out how they determine the maintainability of you code and allow you to edit the settings to choose which aspects of maintainability you want to include. Some of these settings include: argument count, file size, method complexity, method count and length and similar blocks of code.

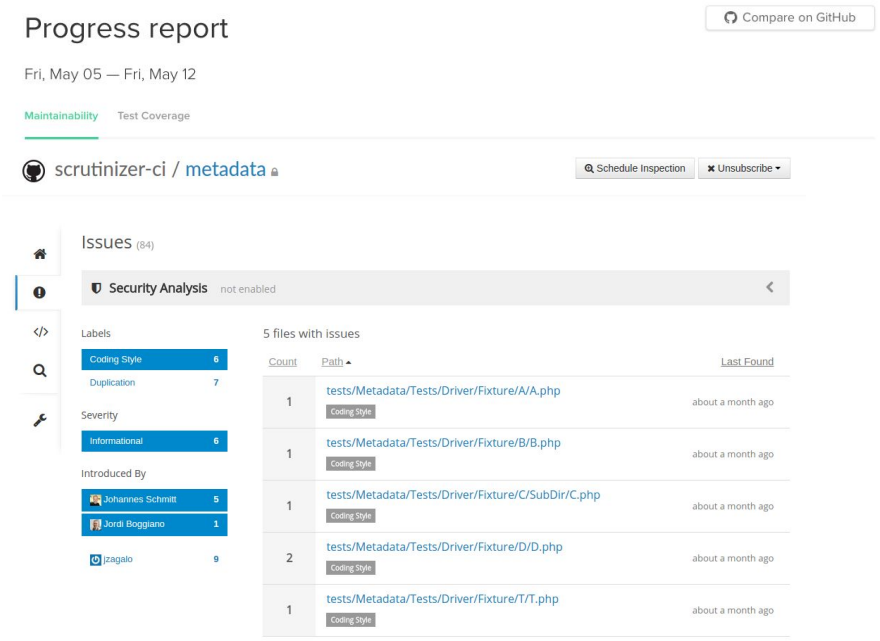


2.2.3 Progress: The progress section is one of the more appealing functions that Code Climate has to offer. It shows how files are improving under the headings of maintainability and test coverage. As you can see below, each file is graded with a letter. It is a very clear way to show if a file is improving or disproving under those headings.

2.3 Scrutinizer

Similar to Code Climate, the first thing to note is how well designed the website is. Their layout is extremely intuitive and user friendly.

2.3.1 Code Quality Control: Scrutinizer offer an instant rating of your source code which gives you a quick insight on the overall health of your code. Similarly to Code Climate, they rate files by grading them and show if a file has improved or disproved by changing the grade. Again, this gives the user a quick and easy to read depiction of the quality of your files.



2.3.2 Issues: One of the interesting issues Scrutinizer reports is coding style issues. You can enforce coding styles on the project to ensure that all code produced is standardised. Bugs are identified in the code throughout development and the person who introduced the bug is identified.

2.3.3 Metrics: Scrutinizer have a really nice way of breaking down a file a measuring it against four different metrics: Complexity, Duplication, Size and Importance. Their analysis goes through each function in a file, measuring it against the four metrics, and produces a report. This report indicates what area needs to be fixed and as you can see in the example below, they even give you a suggested course of action in order to rectify the mistakes.

2.4 Which one is better?

With only limited usage, it is difficult to determine which one is better than the other. I did however, test both with my own github repository so a small level of insight was gained. In my opinion, Code Climate offers a better analytics platform for your code the reports it offers appear more intuitive and its test coverage is more extensive. They also support a wider ranger range of languages than Scrutinizer do. Also an important factor to consider if you were to purchase either software is price, which Code Climate comes out on top again at $16.67/user compared to Scrutinizers $200/month(Netguru.co, 2017).

3. Algorithmic Approaches Available

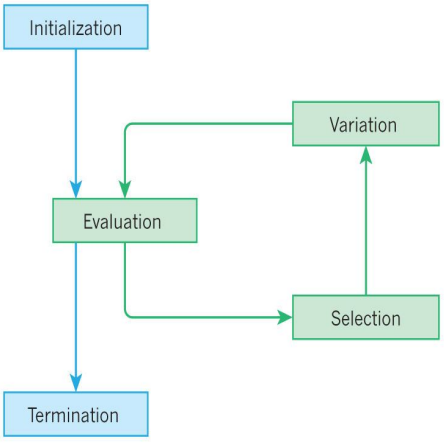
3.1 Computational Intelligence

Computational intelligence is considered to be the ability of a computer to learn a specific task from data or experimental observation. The methods used by this type of intelligence is a computer's attempt to be able to rationalise real world problems in a similar way humans do; using incomplete knowledge and inexact information. There are five areas under the umbrella of CI:

3.1.1 Fuzzy Logic: This is a computing approach that is based on degrees of truth rather than basis of modern computing, boolean logic. This technique was first introduced by Dr. Lotfi Zadeh in 60's when he was working on the problem of translating natural language into computational language. Fuzzy logic is much closer to the way humans think, aggregating data and forming a degree of truth as a result. Interestingly, fuzzy logic is used a lot in household appliances such as washing machines, microwaves and ovens.

3.1.2 Neural Networks: These are systems that aim to replicate biological neural networks in the brains of animals. The technique is based on connected artificial neurons, which are models of biological neurons. These are primarily used for pattern matching, game playing and decision making.

3.1.3 Evolutionary Computation: As the title suggests, this section is based on Darwin's theory of evolution and is more of a family of algorithms than a technique. In evolutionary computation, an initial set of candidate solutions is generated and iteratively updated. Each new generation is produced by randomly removing less desired solutions, and introducing small changes, also at random.



3.1.4 Learning Theory: This class of supervised learning where algorithms aim to make accurate predictions based on previous observations. Computational learning theory also studies the time complexity and feasibility of learning. A computation is determined feasible if it is completed in polynomial time, that is if your result yielded is positive.

3.1.5 Probabilistic methods: An element of Fuzzy Logic, probabilistic methods determine solutions defined by randomness and prior knowledge. Firstly introduced by Paul Erdos and Joel Spencer in 1974.

3.2 Machine Learning: Supervised vs Unsupervised Learning Algorithms

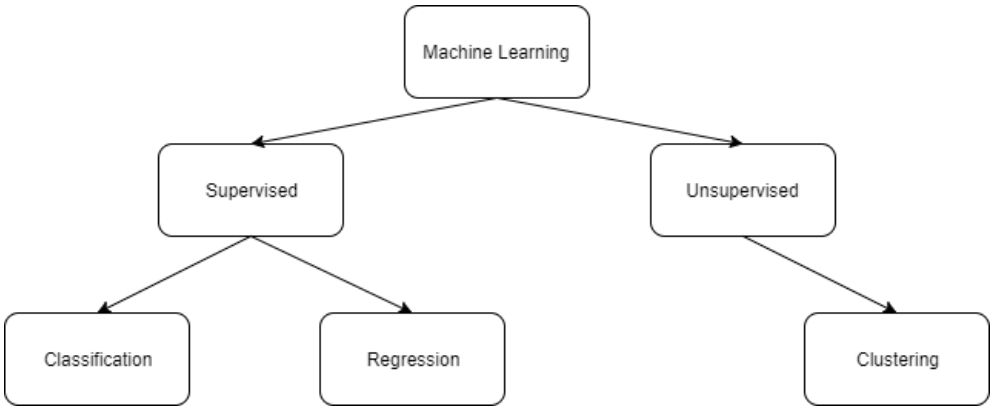
Broadly speaking, machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.
The field explores the study and construction of algorithms that can learn from and make predictions on data.

3.2.1 Supervised Learning:

A machine learns from a dataset and is able to classify something into a class. An example of this would be be an algorithm recognizing a picture of a cat after been shown pictures of cats before. The algorithm would you use the previous data set to classify the new piece of data. Regressions is also another aspect of supervised learning where continuous predictions are made like changing temperatures.
A very commonly used application of supervised learning is email spam filters which identify spam using previous information on what email are actually spam.

3.2.2 Unsupervised Learning:

Unsupervised learning occurs when an algorithm identifies intrinsic underlying structures within the data and “clusters” the data based on common characteristics. It is called unsupervised learning as there are no correct output per say and no prior guidelines or rules which the algorithm has to adhere to.



4.Ethics in Software Analytics

4.1 Overview

So far, key areas of the software process have been analysed; what to look at, what platforms to use and what approach to take. The softer aspect of these analytics is perhaps the most important: what are the human implications and ethical consequences of undertaking such analytics. Data collection is a huge topic in this digital era, not only located within the software engineering process itself but with everyone who interacts with software. Everyday 2.5 exabytes of data are collected daily. To put that into perspective, that is 530,000,000 songs everyday(Khoso, 2017). In the following, will outline the issues with collecting this data, how it should be handled, who owns this data and what are some of the legal implications surrounding this data.

4.2 Privacy

Privacy is a big issue with data collection. Is collecting data on software engineers even ethically correct to begin with? You could argue that just because they are part of a process of software engineering doesn't entitle their right to privacy to be stripped from them. Some tools to gather data used in the workplace include fitness trackers, telematics and sociometric badges (devices capable of measuring face-to-face interaction, conversation time and the wearer's physical proximity to other people), computer monitoring software and sentiment analysis tools, which scan unstructured data (for example email text) to build up a picture of someone's personality or feelings about a particular subject. What is most concerning is that as the technological capabilities advance, the ethical safeguards that should be in place to protect the employee are yet to reach a level of equal consideration. It has been suggested the data collection in the workplace should be defaulted to anonymity and aggregated so that the privacy of the individual remains intact(Harvard Business Review, 2017).

4.3 Data Sovereignty

One of the most important issues with gathering data in the modern era is the sovereignty of it. Who actually owns this data?

Within an organisation, all data collected belongs to them. They can do whatever they want with that data having received prior consent. When it comes to data that is related to the employees of the company itself, again, they have full control over the data once it is stipulated in your contract. This, of course, means that it is the responsibility of the company to carry ethical weight in their data collection is on them.

4.4 Data Transparency

When a company is measuring data on their workforce, it is important for their to be a high level of transparency between what the employer is collecting and what the employee understands is being collected. This extends to what the employer is using the data collected for. In August of 2017, the EU Data Protection Working Group published their opinion on this subject specifically outlining that the usage of the data, both in the present and future, by the employer should be outlined to the employee(European Data Protection Supervisor, 2017).

4.5 Conclusion

While the incessant collection of masses of data will likely continue to grow and that employees will be subjected to an increasingly Orwellian environment, it should always be the responsibility of the data proprietor to ensure that it is secure, legal and ethically used, even if it isn’t explicitly in the law.

Bibliography

Codeclimate.com. (2017). *Code Climate*. [online] Available at: https://codeclimate.com/ [Accessed 28 Nov. 2017].

European Data Protection Supervisor. (2017). *A - European Data Protection Supervisor*. [online] Available at: https://edps.europa.eu/data-protection/data-protection/glossary/a_en [Accessed 28 Nov. 2017].

Harvard Business Review. (2017). *Collect Your Employees' Data Without Invading Their Privacy*. [online] Available at: https://hbr.org/2014/09/collect-your-employees-data-without-invading-their-privacy [Accessed 28 Nov. 2017].

Khoso, M. (2017). *How Much Data is Produced Every Day? - Level Blog*. [online] Level Blog. Available at: http://www.northeastern.edu/levelblog/2016/05/13/how-much-data-produced-every-day/ [Accessed 28 Nov. 2017].

Netguru.co. (2017). *Comparison of Automated Code Review Tools: Codebeat, Codacy, Codeclimate and Scrutinizer*. [online] Available at: https://www.netguru.co/blog/comparison-automated-code-review-tools-codebeat-codacy-codeclimate-scrutinizer [Accessed 28 Nov. 2017].

Visual.ly. (2017). *Technical Debt in figures | Visual.ly*. [online] Available at: https://visual.ly/community/infographic/technology/technical-debt-figures [Accessed 28 Nov. 2017].