Measuring Engineering – A Report

Darragh Hurley **16319250**

In this report I will be discussing the ways in which the software engineering process can be measured. The four main topics of this report include:

1. Measurement and assessment in terms of measurable data.
2. An overview of the computational platforms available to perform this work.
3. The algorithmic approaches available.
4. The ethics concern surrounding this kind of analytics.

Using these four topics as a general guideline, I will discuss how software engineering is measured, the platforms available to support the measurement of this software engineering and how this analysis can have an effect ethically.

Measurement and assessment of software engineering in terms of measurable data

Software productivity is a measure of the rate at which individual engineers involved in software development produce software and associated documentation. Quality assurance is a factor in productivity assessment. Essentially, we want to measure useful functionality produced per time unit. The three mains points of measurement that I believe can be used simultaneously to achieve a good estimate of the productivity of a software engineer are:

1. Value of Code

- This is a way of assessing each line of code and providing a rating to it showing its quality and significance to the project.

2. Speed

- This is a great indicator of productivity, but it's not a complete view as the quality of the code or the long term effects are not taken into consideration.

3. Technical Debt

- This is extra development work that arises when code is implemented. It might solve an immediate problem, but create other problems that need to be solved in the future.

Using these three metrics I believe that the productivity can be analysed fairly. For each metric there are multiple ways of obtaining the necessary data.

Value of code:

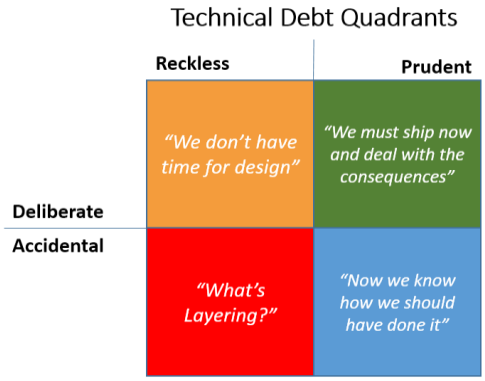
Identifying the value of code involves more than just counting the number of lines. Originally the software industry had attempted to measure productivity around the number of lines that a developer writes. This process is flawed for multiple reasons. The first reason is that some more efficient code can be written using less lines of code, therefore the worse solution would be considered as having more work done for simply having more lines of code. Another reason is that software engineers could simply increase the number of lines in a code bundle by adding in unnecessary comments and white space in order to make it look as though they have been more productive. In order to really figure out the value of code the number of lines should be incorporated, but only as part of it. Each line of code needs to assessed to quantify the impact it has on the code base.

Speed:

How long it takes a software engineer to complete their work is another way to measure productivity. In reality the more productive a software engineer the quicker they will be done, but again there are certain flaws to just measuring the speed at which the developer completes the task. If speed was the only metric used to measure the productivity of software engineering then the likelihood of the program

running into future issues and errors would increase massively. Developers would be more inclined to produce the product working at its bare minimum as they would seem productive. This is one of the reasons that software engineers are given deadlines for their assigned tasks to ensure that they have enough time to produce the most thorough solution possible for the client. The maintainability and functionality of the code are vitally important. The faster that these requirements are achieved the more likely it is the at the developer is being productive.

Technical debt:



Technical debt can be described as extra development work that arises when code is implemented. Code duplication and complexity, low test coverage, lack of documentation, and programming rule violations are all factors that contribute to technical debt. There is no single formula to calculate technical debt. However, identifying the debt sources and estimating the amount of added development time is a good place to start. The main ways of calculating technical debt is simply by looking at the amount of code duplication (0% being the ideal scenario), the complexity of code (which is calculated from the number of functions, classes, each code branch and every exit path or exception), the test coverage (how much of the code is tested), the amount of documentation and the number of dependency cycles and coupling.

In my opinion these metrics combined is the best and most fair way of measuring software engineering. I don't believe that there will be just one way of measuring

how productive a developer is but I do think that right now this is a good way to compare their productivity.

An overview of the computational platforms available to perform this work.

One computational platform that allows for the measurement of software engineering is static object. Static object has a system called “line impact” which assesses and visualizes developer output. Line impact is a metric designed to capture the degree of cognitive load required to fashion a line of code. The calculation of Line Impact begins by assessing the action that occurred during the commit, and then factors in the context surrounding this action. There is a list of actions that line impact recognises in commits:

1. Addition
2. Deletion
3. Move
4. Update
5. Revision
6. Find and replace
7. No-ops

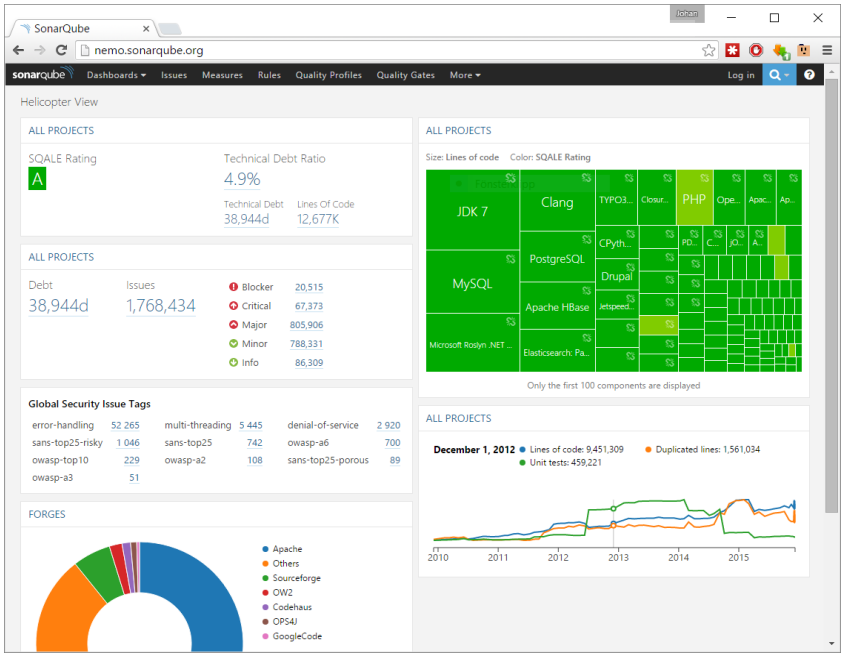
Each of these actions have a specified value that corresponds to how much the line of code impacts the overall code bundle. Alongside these actions is an assessment of the lines context to yield their final estimate of cognitive load. Examples of the assessments include:

- Proximal Changes
- Churn
- File type

- Keywords and syntactic lines
- Comments

Combining both of the actions and assessments can produce a value to demonstrate how impactful the specific line of code is. Static Object also includes a velocity measurement that allows managers to see each developer's average Line Impact per day and observe how contributions trend over time.

Another useful computational platform that gathers, analyses and displays data is SonarQube. SonarQube is an open source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages. An example of the data that is retrievedand displayed by sonarqube is shown below.



The outcome of this analysis will be quality measures and issues (instances where coding rules were broken). However, what gets analyzed will vary depending on the language:

- On all languages, "blame" data will automatically be imported from supported SCM providers. Git and SVN are supported automatically. Other providers require additional plugins.
- On all languages, a static analysis of source code is performed (Java files, COBOL programs, etc.)

- A static analysis of compiled code can be performed for certain languages (.class files in Java, .dll files in C#, etc.)
- A dynamic analysis of code can be performed on certain languages.

Another way of measuring software engineering are by the testing suites that can be found in multiple integrated development environments (IDEs). IDEs such as Eclipse, Visual Studios and Intellij IDEA. For Eclipse there is a code coverage plugin called EclEmma and also has multiple code quality plugins such as jSparrow and pmd. Visual Studios has a tool for code coverage also called dotCover. IntelliJ IDEA provides certain features like code completion by analyzing the context, code navigation which allows jumping to a class or declaration in the code directly, code refactoring and options to fix inconsistencies via suggestions.

These computational platforms are only a small part of the vast number of technologies that are available to test software engineering.

Algorithmic approaches available

A very basic cost model for the amount of effort a software engineer has put into the code:

$$\text{Effort} = A \times \text{Size}^{B} \times m(X)$$

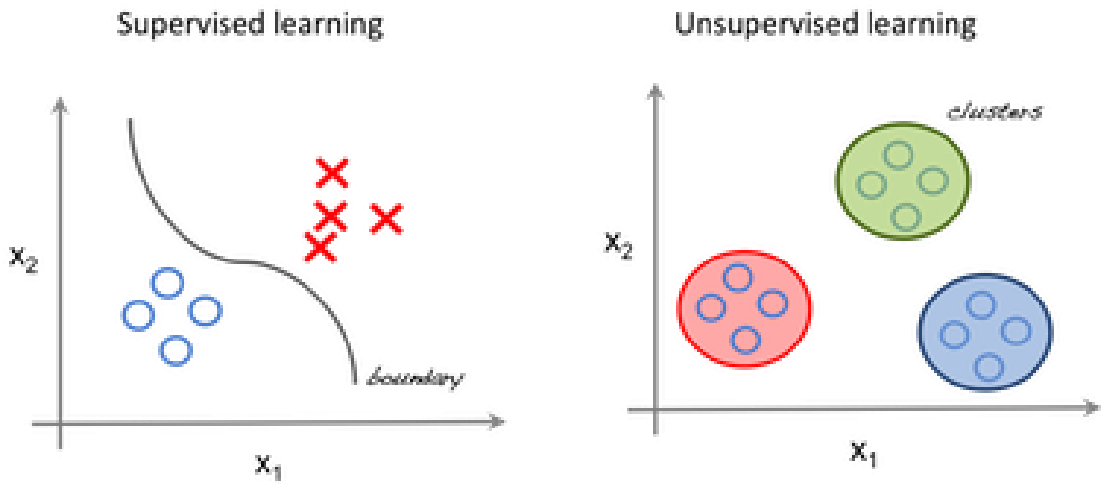
Where ‘Size’ is some measurement of the software size, ‘A’ is a constant factor that depends on organizational practices and type of software, ‘B’ usually lies between 1 and 1.5, ‘X’ is a vector of cost factors and ‘m’ is the adjustment multiplier. This model is purely based in the amount of time and effort a developer has put into their project but a more useful algorithmic approach to measure the level of software engineering done by this developer is machine learning.

Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions. There are three main algorithms that machine learning can be divided up into.

Unsupervised Learning:

Unsupervised machine learning algorithms infer patterns from a dataset without reference to known, or labeled, outcomes. Unsupervised machine learning methods cannot be directly applied to a regression or a classification problem because you have no idea what the values for the output data might be, making it impossible for you to train the algorithm the way you normally would. Unsupervised learning can instead be used for discovering the underlying structure of the data. The best time to use unsupervised machine learning is when you don't have data on desired outcomes. Some applications of unsupervised machine learning techniques include:

- Clustering - allows you to automatically split the dataset into groups according to similarity.
- Anomaly detection - can automatically discover unusual data points in your dataset.
- Association mining - identifies sets of items that frequently occur together in your dataset.
- Latent variable models - commonly used for data preprocessing, such as reducing the number of features in a dataset (dimensionality reduction) or decomposing the dataset into multiple components.



Supervised Learning:

Supervised machine learning algorithms uncover insights, patterns, and relationships from a dataset that already contains a known value for the target variable for each record. Since the machine learning algorithm is provided with the correct answers for a problem during training, it is able to learn how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data. Techniques for supervised learning include regression, where the algorithm returns a numerical target for each example and classification, where the algorithm attempts to label each example by choosing between two or more different classes. Supervised machine learning turns data into real, actionable insights. It enables organizations to use data to understand and prevent unwanted outcomes or boost desired outcomes for their target variable. It can be used to solve problems such as:

1. Reducing customer churn
2. Determining customer lifetime value
3. Personalizing product recommendations
4. Allocating human resources

5. Forecasting sales
6. Forecasting supply and demand
7. Detecting fraud
8. Predicting equipment maintenance

Reinforcement Learning:

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. As it is an automated learning scheme it means that there is little need for an expert who knows about the domain of application. Two elements make reinforcement learning powerful: the use of samples to optimize performance and the use of function approximation to deal with large environments. There are four main algorithms for this type of learning:

- Criterion of optimality
- Brute force
- Valve function
- Direct policy search

These three forms of machine learning provide a very strong form of test coverage and therefore can measure the quality of coding provided by the developer.

Ethics

When measuring software engineering the ethics involved must always be taken into consideration. The main ethical problems that could occur from the measurement of software engineering include issues to do with privacy and security.

The main ethical concerns deals with the sharing of information that is necessary nowadays. For programmers to get their code analysed by a computational platform

they are more than likely going to have to provide not only their code, but some further information. The primary ethical issue seen is the selling or leaking of information provided. This information is genrally supplied by the client of the service with the expectation that it would stay confidential. Unfortunately that is not always the case. If poor security systems are in place that information could be found and sold immediately. Not all companies are reliable either, some would be willing to share your information for their own benefit as there are a large amount of advertising companies that can use this data to configure the types of advertisements that you receive. A lot of services online can also receive more data than what the client had initially thought. Although the client may seem content with the information that they have provided as they believe it is very basic and irrelevant information they may be providing more information while using the service. For example these services may use a location-based tracker that the client may be unaware of. One of the other ethical issues seen with the use of these computational platforms is that intellectual property could be stolen.

Another large issue seen ethically with the measurement of software engineering is the altering and falsification of data. The resulting output could be altered by the company in order to satisfy a client instead of providing the legitimate result in order to entice the client to use the service further. Similarly the resultsmay be altered to return a poor result as an attempt to prove that the service can show flaws to the client.

References:

https://www.jetbrains.com/idea/

https://www.jetbrains.com/dotcover/features/?gclid=CjwKCAiAiarfBRASEiwAw1tYv3PTHN2yxZBlJ7YcqTuoJGTu2ciTNOlGAecSdPNEEy1Fcq8OmrrQNxoCSbIQAvD_BwE&gclsrc=aw.ds

https://marketplace.eclipse.org/category/free-tagging/code-quality

https://www.techemergence.com/what-is-machine-learning/

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

https://www.datarobot.com/wiki

https://www.sonarqube.org/

http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/slides16.pdf

https://medium.com/static-object/3-key-metrics-to-measure-developer-productivity-c7cec44f0f67

http://thinkapps.com/blog/development/technical-debt-calculation/

https://www.staticobject.com/