Junior Sophister Essay Submission Form

Form Student Name:	Rory Saunders
Student ID Number:	15302899
Programme Title:	Computer Science and Business
Module Number and Title:	CS3012: Software Engineering
Essay Title:	Measuring Engineering
Lecturer(s):	Professor Stephen Barrett
Date Submitted:	06/12/17
College Tutor:	Dr. Arthur White

Signed: Rory Saunders Date: 06/12/17

Measuring Engineering

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.

1. Introduction

"Software, in its most general sense, is a set of instructions or programs instructing a computer to do specific tasks."[1]

Most would agree that software arose in the mid 20th century from the minds of visionary inventors like Alan Turing and John von Neumann. The invention of the computer gave rise to an explosion in technology and software production that revolutionized the way people live their lives. Nowadays, over 40% of the world's population has access to the internet[2], and a far greater number have access to computers. Furthermore, software, in many cases, is responsible for the security of financial assets, the stability of medical systems, the efficacy of education, and much more. Software determines the way most people in the developed world interact socially, do business, and enjoy their free time. As such, it is of vital importance that software be held to a supremely high standard to protect the privacy and lifestyle of those who use it.

The way that the quality of software is measured is through the implementation of software measures and metrics. With respect to software:

- A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process.
- A Metric is a measurement of the degree that any attribute belongs to a system, product or process. [3]

[1] https://www.techopedia.com/definition/4356/software
[2] http://www.internetlivestats.com/internet-users/
[3] https://en.wikiversity.org/wiki/Software_metrics_and_measurement#Overview

Examples of software measurement include the number of lines of code in a software engineering project, the number of bugs within the code, the number of developers, or anything else similarly quantifiable. Metrics are more simply described in terms of how they are used within the software engineering process. A company might be interested in how many lines of code each developer writes per hour, the percentage of test cases that the code passed, or something external like user feedback. However, in many cases, the term software metrics is used to describe the way companies analyze their code. These practices allow developers and others to analyze the extent to which their software performs the duties asked of it, and makes software more effective and safer for all of its users.

2. Measuring and Assessing Data

There are thousands of ways to analyze code, and the quality of said code is largely subjective. However, there are a few software metrics that are employed by most companies and have endured over the years as useful and worthwhile. I will describe a few particularly prominent strategies as outlined by Norman E. Fenton and Martin Neill in their article 'Software Metrics: Successes, Failures and New Directions'.[4]

- Lines of Code:

Otherwise known as LOC, this is a simple measurement of the amount of code a developer has written. LOC is particularly apposite for monitoring the productivity of software engineers. While the obvious drawback is that it isn't consistent across varying programming languages or development environments, it is still useful to make comparisons between similar languages. For example, a supervisor might want to know the relative speed at which his three Java developers code. It becomes

[4] https://www.sciencedirect.com/science/article/pii/S0164121299000357#!

more difficult to implement when one engineer is coding in machine language and the other in Python.

- Defect Count:

The LOC metric becomes significantly more useful when paired with a defect count metric. An engineer doubling another's LOC output becomes significantly less impressive when it is revealed that he also gives rise to twice as many bugs in the code. Almost all companies employ some sort of defect count metric, as it is integral in promoting good engineering practice and training new developers.

- Cyclomatic Complexity:

Cyclomatic Complexity, invented by Thomas McCabe, is a measure of the number of independent paths through a program's source code. In essence, it measures how chaotic and difficult to understand a particular piece of code may be. It's uses include determining the number of test cases to be written, determining whether the code can be worked on by other engineers, and, in extreme cases, throwing out overly chaotic code. One hundred lines of simple code is much easier to implement in a larger code body than one hundred lines of intensely recursive, object heavy code.

- Function Points:

Similar to cyclomatic complexity, the amount of function points in a body of code is independent of its number of lines. Function points uses a formula and a five point strategy[5] to quantify the size of code in terms of functionality rather than the number of characters used to achieve said functionality. For this reason it is purportedly a more accurate measure of developer productivity than LOC.

These four strategies comprise most of the software analysis done industrially in the corporate scene. Other strategies that are commonly employed and worth mentioning include code coverage, maintainability indexes, program execution time, and program load time. Ideally, all strategies would be implemented in order to assure the safest, most efficient code possible. However, time constraints

[5] http://www.softwaremetrics.com/fpafund.htm

and the financial reality of software development means that companies often skip out on this step in order to save time and money.

3. Available Computational Platforms

In addition to all the ways of measuring and assessing data there are also systems which developers employ to consolidate and streamline the process. Some of these platforms are done by hand, counting up bugs and lines of code in order to self assess. Others are automated and require little effort on the part of the developer. I have outlined a few notable examples below, taken from Philip M Johnson's article 'Searching under the Streetlight for Useful Software Analytics.'[6]

- PSP:

An acronym which means Personal Software Process, PSP is one of the aforementioned platforms in which a developer must manually input and calculate values in order to gain insight into their own work. Supporters argue that this onerous process leads to more introspection and genuine consideration of the coding process, but the time involved fails to impress others. Another advantage of the PSP is its flexibility. Due to the labor intensive nature of the process, it also allows developers to focus on software metrics that are particularly relevant for the current project. Despite this, a further drawback is that the lack of automation leads to errors, which Philip M Johnson found to hover around 5%. In order to remedy this, many automated processes were created.

- The Leap Toolkit:

The Leap Toolkit still required developers to manually input code, however the calculations were done for the developer by way of a client. Furthermore, developers were able to keep the data

[6] http://ieeexplore.ieee.org/document/6509376/

files created. This allowed them to transfer this data across projects or even organizations. Despite the improvements in speed and transferability, Leap didn't provide the requisite flexibility and depth.

- Hackystat:

As many developers were dissatisfied with the stop and start process of working and subsequently writing down all they had done, developers endeavoured to make a program which automatically recorded important data server side and client side without impeding the progress or speed of a developer. The client recorded incrementally all that the developer did, and even what other developers contributed to their projects, and made the calculations and such automatically. Unfortunately, many developers weren't comfortable with management having access to such personal information and despite reading ethical practice agreements rejected the software. Another downside was privacy, in that important corporate data would need to be secured within the software of Hackystat as well so as not to jeopardize valuable information.

- Agile Development:

While Agile Development isn't merely a system to analyse code, it contains some interesting practices for doing so. Agile Development encourages engineers to work alongside each other and test each other's code in order to maximize understanding and minimize bugs. There is an extremely short feedback loop of tasks and analysis, and while engineers are required to write a sizeable amount of documentation, the process avoids large scale errors that might be encountered with something like the waterfall method of software development.[7] Furthermore, Agile Development lessens the need to personally keep track of each developer's stats and efficacy, as the burden is on the teams to cooperate for success. The short feedback loop also allows management to assess who is completing tasks quickly and efficiently without resorting to programs that collect data on their engineers over significant periods of time.

- Sonar/Open Hub:

[7] https://www.agilealliance.org/agile101/

Sonar is a software much like Hackystat, however it also provides help fixing bugs, and personalizes the advice to each project and programming language.[8] It also has less privacy issues as it creates a company repository locally. Open Hub markets itself as an open source version of Github with a larger emphasis on software analysis and metrics.[9]

4. Available Algorithmic Approaches

There are three particularly notable algorithmic approaches to analyzing software, two of which were already discussed. They are Cyclomatic Complexity, Function Point Analysis, and Halstead Complexity measures.

- Cyclomatic Complexity:

To calculate Cyclomatic Complexity, the developer first models his code in a graph with nodes and edges representing the loops, statements, and functions contained within. The rules for creating this graph are available all over the internet.[10] The complexity is then calculated using the following formula:

$$M = E - N + 2P.$$

Where M is the complexity, E is the number of edges, N is the number of nodes, and P is the total number of components.

- Function Point Analysis:

[8] https://www.sonarqube.org/
[9] https://www.openhub.net/
[10] https://www.guru99.com/cyclomatic-complexity.html

FPA relies on five major factors: External Inputs, External Outputs, External Inquiry, Internal Logic Files, and External Interface Files. These factors are then ranked and plugged into a series of tables in order to inform developers about issues in scope or direction concerning the project.[11]

- Halstead Complexity:

Similar to Cyclomatic Complexity, Halstead Complexity is used to more accurately measure the efficiency and productivity of software engineers. However, with Halstead Complexity there are a number of different measures of complexity for which you can solve. The variables themselves are: number of unique operators n1, number of unique operands n2, total number of operators N1, and total number of operands N2. These values can then be used to calculate relevant metrics such as[12]:

$$\textit{Program Length} = N1 + N2$$

$$\textit{Program Vocabulary} = n1 + n2$$

$$\textit{Program Volume} = N * \log2(n)$$

$$\textit{Difficulty} = (n1/2) * (N2/n2)$$

$$\textit{Program Level} = 1 / \textit{Difficulty}$$

$$\textit{Effort to Implement} = \textit{Volume} * \textit{Difficulty}$$

Halstead Complexity can even be extended to estimate the number of bugs a new project will bring to a code base and a number of other more complex calculations.

5. Ethics Concerns

Some aspects of software measurement have serious ethical ramifications.

[11] http://www.softwaremetrics.com/fpafund.htm
[12] http://www.verifysoft.com/en_halstead_metrics.html

- Client Side/Privacy:

For example, in order to better collect data about an application it may be necessary to jeopardize a user's privacy. Some users even claimed that the Facebook app was listening to them and displaying advertisements based on what it heard.[13] Other similar examples include Google tuning search results based on past history. While these features are intended to improve the functionality of the respective services, they may also anger users who feel their privacy has been infringed upon.

- Managerial Supervision:

Many developers rejected the aforementioned Hackystat platform because they felt it gave manager an unnecessary amount of oversight into the work that they do. If developers are judged based solely on quantitative measures rather than as people this could lead to serious problems within the industry. This could lead to developers not wanting to cooperate for fear that their numbers will drop by comparison, and thus they look worse to management. The potential for the creation of a toxic work environment is a serious ethical concern.

- Automation:

Automating the process of analyzing code removes part of the job of a software developer, and sets a precedent that could end really badly. If engineers find a way to perfectly analyze software through automation then the ability to perfectly create software automatically is just around the corner, which would put most software engineers out of a job. This hysteria with regards to automation is ever present in the public mind and is frequently represented in science fiction books and movies. In our day and age it is not an irrational fear.

13
http://www.telegraph.co.uk/technology/2017/10/30/facebook-listening-conspiracy-theory-refuses-die/

6. Conclusion

Software measures and metrics are an integral part of software development too little discussed or put into practice. The amount of research academically dwarfs the industry implementation which is a shame. While it may slow down progress from a lines of code per hour standpoint, it ensures that the software that everyone uses is bug-free, safe, and simple. Quality code is what all genuine software developers should strive for, and software measures and metrics make it easier to write good code. In a world where software is rapidly becoming ubiquitous, software measures and metrics are of the utmost importance.