Measuring Engineering - A Report

by Cian O'Brady (16326332)

Measuring and analysing the software engineering process is very important to understanding the successes and failures of the process. It is widely used in many forms, with many tools and features available to calculate and measure certain metrics that can later be interpreted and analysed. In this report I will discuss 4 topics related to the measuring of the software engineering process. These topics include:

- ➔ 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.
- ➔ The ethical concerns surrounding this kind of analytics.

Measuring the Software Engineering Process

The measuring of the software engineering process is used for evaluation and analysis of the process. When something can be measured, it can be analysed and as a result improved. However it is crucial to develop what is being measured and how it is being analysed. In software engineering, software metrics are used to measure the process. According to an article written by V.Kharytonov in 2012, software metrics were first introduced in the 1970s, and have been developed and advanced ever since.

One of the types of software metrics is formal code metrics. These include measuring the amount of lines of code written by a developer and the complexity of the code written. While the amount of lines of code is the simplest way of measurement, it is ultimately not a very helpful metric. This is due to the fact that lines of code can mean very little in terms of one's actual contribution to a project. For example, a developer could write 100 lines of trivial code that doesn't contribute to developing a solution, while another person could a 20 line piece of code that is very important and difficult to write. They may have written less but ultimately they have contributed more. Lines of code can also be misleading, as the goal should be to write a more efficient

solution and therefore to write less. Measuring code complexity is a more useful metric as it takes into account what a developer has written as opposed to how much they have written, but is still flawed. A developer could write very complex solution but it should be also rewarded if someone were to find a less complex, more efficient solution. Formal code metrics are useful if they can be analysed correctly but ultimately are less valuable than other metrics.

Another form of software metrics is developer productivity metrics, which involves measuring the amount of time a developer spends on a project and what they do in that time. A lot of useful information can be found using these metrics, including the how often a developer is working on a project, what rate they produce code at, their efficiency and the amount of effort they put into the project. This is useful to see how efficient a developer is. It is also to see the investment and effort a developer is putting into a project as it can be seen how often they are working, how long they are working for and how much they are putting in when they are at work. This type of metrics can also be used to determine the ideal team size, as they can see how many developers are needed to develop something in a given time.

Operational metrics are another form of software metrics. These include, “Mean time between failures (MTBF)”, which is the predicted time between failures in a system and, “Mean time to recovery (MTTR)”, which is the average time for a system to recover from a failure. These are very useful as they check how well a system is running and therefore the quality of the developers work.

Test metrics such as code coverage are a way of measuring how well a system is tested. Testing is a crucial aspect of any software engineering process and should be implemented to check how well the system has been implemented by the developers. Testing can also be used to measure the efficiency of the code as runtimes can be seen and potentially improved upon. It can also be used to check the quality of the code as any potential failures can be spotted and rectified.

One metric that I believe is very important is the satisfaction of customers. The reviews of customers is very important in measuring the software engineering process as it truly determines the quality of what the developers have created. Customers may not understand the ins and outs of the development process,

how long was spent on the project, who contributed what etc., but I believe the customers input is a good measurement in terms of determining the successes and failures of a software engineering process. There are many different ways to measure customer satisfaction such as Net Promoter Score (NPS), Customer Satisfaction Score (CSAT), and Customer Effort Score (CES), as well as simply listening to what customers believed were positive and negative aspects of the project.

There are many different ways that the software engineering process can be measured as there are lots of different metrics available. However it is important to understand that some metrics are more valuable than others and must be analysed correctly in order to evaluate properly and improve upon any potential failures in the software engineering process.

Computational Aspects Available

There are many tools available to measure and assess the software engineering process, such as software development platforms, IDE's and various plugins.

An example of this is software development platforms such as Github. Github has many tools to display valuable information about the software engineering process. Lines of code can be easily evaluated as the amount of lines of code that have been both added and removed by each individual is documented and visible. Developer productivity can also be measured as the amount of commits by each contributed is also recorded. Visualisations are also available to show how often a developer commits and when they commit.

Figure 1 - Screenshot of contributor statistics in Github repository (from Github.com)

In Figure 1, we can see an example of the statistics that be seen from a contributor on Github. The amount of commits by the user can be seen, as well as the amount of lines added and the amount of lines removed. There is also a graph that shows when the commits were made. Github provides a very useful set of available tools but it is possible to see even more, as one can interrogate the Github API in order to access and analyze even more data. It is also possible to use the Github API to develop your own visualisations and make the data more understandable.

Many IDE's have ways to measure some software metrics. One of these IDE's is Windows Visual Studios. According to Visual Studio Docs, Visual Studios offers many ways to measure code metrics. One of these is a maintainability index. Visual Studios is able to evaluate your code and return a score between 0 and 100 that determines how easy it is to maintain the code. It offers many other metrics including cyclomatic complexity, depth of inheritance, class coupling and lines of code.

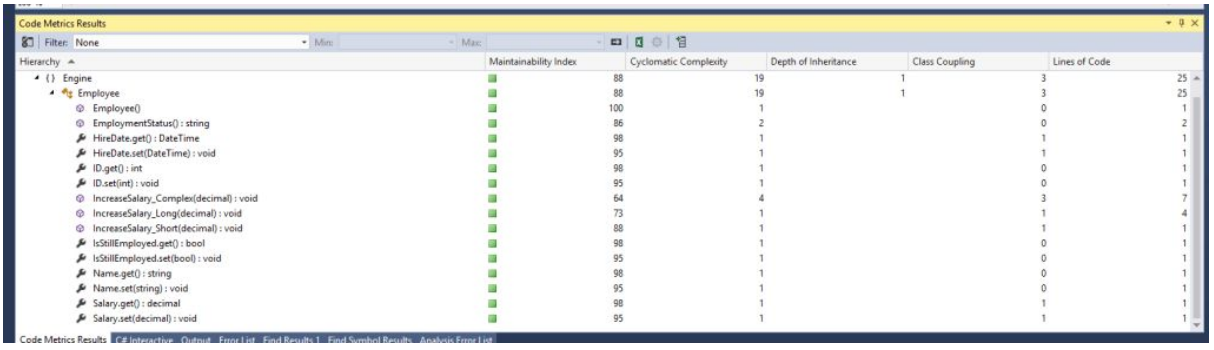

Figure 2 - Screenshot of Visual Studios metrics (from https://scottlilly.com/cyclomatic-complexity-in-visual-studio/)

Another example of an IDE that calculates metrics is Eclipse through many plugins. One of these plugins include the Metrics plugin. There are many available metrics measured including simple ones like lines of code, number of classes and numbers of methods, as well as more complex metrics such as cyclomatic dependency and lack of cohesion of methods. The most useful aspect about this is that it provides visualisations and therefore make the data much more readable and understandable. According to the documents for this plugin, many of the metrics are based off of “Object-Oriented Metrics, Measures of Complexity" by Brian Henderson-Sellers, Prentice Hall, 1996.

CCCC is a free tool for software metric measurement. It was developed by Tim Littlefair. It is a metric analyser for C and C++. It can calculate many metrics such as lines of code, cyclomatic complexity and is able to generate HTML and XML reports of various other metrics.

There are many available tools that can be found online (a lot of which are free) and can be used to measure many metrics in many different coding languages.

Algorithmic Approaches Available

Cyclomatic Complexity

One algorithm to calculate the complexity of code is cyclomatic complexity. This algorithm was developed by Thomas J. McCabe. The first step of this algorithm is to create a Control Flow Graph of the code. A node of this graph is any instruction and an edge of this graph is made between any two nodes where one node is a potential next instruction of the other node. After creating this graph the amount of nodes and edges can be counted in order to find the cyclomatic complexity of the code. For example, the following code segment in figure 3 would produce the Control Flow Graph in figure 4.

```
X = Y + Z
if (X>10)
{
W = X
X = 0
}
else
{
W = 0
}
print(W)
```

Figure 3 - sample code segment

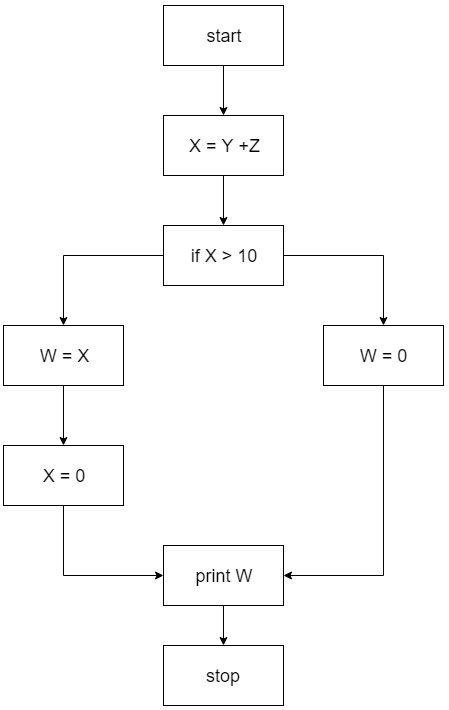

Figure 4 - control flow graph of code in figure 4

In figure 4 there are 8 nodes and 8 edges. The formula to calculate cyclomatic complexity is M = E - V + 2, where E is the amount of edges, N is the amount of Nodes and M is the cyclomatic complexity. Since we have 8 nodes and 8 edges, M = 8 - 8 +2, therefore M = 2.

Halstead's Software Metrics

Halstead's Software Metrics provide many algorithms to calculate certain metrics. In each of these algorithms, the following variables are used:

- n1 = no. of distinct operators in program
- n2 = no. of distinct operands in program
- N1 = total number of occurrences of operators
- N2 = total number of occurrences of operands

Halstead's Program Length (where the length is N) is defined by N = N1+N2.

Halstead's Program Volume (where effort is V) is defined by V = N * log2(n1+n2).
This value represents the size, in bits, of space necessary for storing the program.

Halstead's Program Effort (where effort is E) is defined by E = (n1 + N2 * (N1 + N2) * log2 (n1 + n2)) / (2 * n2). This value is the number of mental discrimination needed to implement the program.

Goal Question Metric

Goal Question Metric (GQM), is a goal-oriented framework for software metrics. It was developed by Dr. Victor Bassili, and involves a 7 step model of the engineering process. The 7 steps are as follows:

1. Develop goals
2. Develop questions that characterise goals
3. Define metrics needed to answer questions
4. Develop mechanisms for data collection and analysis
5. Collect and evaluate data
6. Analyse
7. Provide feedback

GQM is a way for developers to properly outline what they are setting out to achieve, and therefore it is much more possible to accurately measure the successes and failures of a project.

Net Promoter Score (NPS)

One way to calculate customer satisfaction is through Net Promoter Score (NPS). This method collects reviews with values between 0 and 10 and returns a score ranging from -100 to +100. There are three types of values between 1 and 10. Values from 0-6 are considered negative reviews, values from 7-8 are considered passive reviews and values from 9-10 are considered positive reviews. The formula to calculate the NPS is NPS = P - N, where P is the percentage of positive reviews and N is the percentage of negative reviews (the passive reviews are ignored). For example in a survey of 1000, if there were 750 positive reviews, 150 passive reviews and 100 negative reviews, P = 75% and N = 10%. Therefore NPS = 75 - 10, which is an NPS of +65.

Ethical Concerns

It is clear that the measurement of the software engineering process is a very useful way to receive feedback on the process and as a results and make improvements as a result. However there are some ethical concerns present that must be acknowledged.

One of the ethical concerns surrounding this kind of measurement is the ability for the measurements to be misread. It is very possible for someone, perhaps a member of management, to look at a project and make blanket assumptions based on data collected. For example one might see that one member has contributed the most amount of code and it could be suggested that they have worked the hardest and were the most important member on the team. However this is not necessarily the case. This person may have written more because they had to write trival lines of code while someone has written a lesser volume, but more effective and efficient code. This leads to developers being rewarded for quantity over quality which I believe should not be the case. Similarly just because a developer has spent more time developing a solution, does not mean they are working harder. Developers should be rewarded for being able to find a solution sooner, not for taking longer. The interpretation of these measurements is very important. In order for these measurements to be ethically correct, they must be interpreted properly.

I believe that Leon Tranter's message about software metrics is very important when it comes to ethics and software process measurement. According to sealights.io, Tranter outlined 3 key aspects about software metrics.

Metrics should be used by the development team

Metrics should not be used by management as a tool to blame developers. It should instead be used by the developers themselves to learn from their own mistakes and failures, and allow them to improve both the project and themselves as engineers.

Metrics should be part of conversation

Metrics should not be simply taken at face value. They should instead be included as a part of greater discussion that involves aspects that cannot be

measured. To look simply at numbers is an incorrect way to interpret these metrics.

Metrics should be used as part of an evaluation

Metrics should be used with a specific purpose, with the intention of learning about the process. Metrics should not just be used for the sake of using them, there should be a hypothesis in mind before they are used.

I believe these ideas outlined by Tranter are the most effective way of utilizing metrics and remove any ethical concerns that one may have about using the measurement of the engineering process.

Conclusion

Through my study of the measurement of the engineering process, I have learned a lot. There are many different ways to measure this process, with many forms of useful but very different metrics. In order to use these metrics, there are a lot of tools available in order to calculate them, such as software development platforms, IDE's, plugins, and there also many algorithms that have been developed in order to calculate these metrics. In terms of ethics, while I think there are valid concerns, I believe when used correctly, the measuring of the software engineering process is a crucial and effective tool.

Bibliography

V.Kharytonov. (2012). *Software Measurement: Its Estimation and Metrics Used | CISQ - Consortium for IT Software Quality*. [online] Available at: https://it-cisq.org/software-meausrement-estimation-metrics/ [Accessed 21 Nov. 2018].

Sealights. (n.d.). *Top 5 Software Metrics to Manage Development Projects Effectively*. [online] Available at: https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/ [Accessed 21 Nov. 2018].

Scottlilly.com. (n.d.). *Cyclomatic Complexity in Visual Studio – ScottLilly.com*. [online] Available at: https://scottlilly.com/cyclomatic-complexity-in-visual-studio/ [Accessed 21 Nov. 2018].

Metrics.sourceforge.net. (2018). *Metrics 1.3.6*. [online] Available at: http://metrics.sourceforge.net/ [Accessed 21 Nov. 2018].

Cccc.sourceforge.net. (n.d.). *Software Metrics Investigation*. [online] Available at: http://cccc.sourceforge.net [Accessed 21 Nov. 2018].

Fricker, S. (2018). *What exactly is cyclomatic complexity? • froglogic*. [online] froglogic. Available at: https://www.froglogic.com/blog/tip-of-the-week/what-is-cyclomatic-complexity/ [Accessed 21 Nov. 2018].

Sunnyday.mit.edu. (n.d.). [online] Available at: http://sunnyday.mit.edu/16.355/metrics.pdf [Accessed 21 Nov. 2018].

GeeksforGeeks. (n.d.). *Software Engineering | Halstead's Software Metrics - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/ [Accessed 21 Nov. 2018].