Measuring Software Engineering

Lexes Jan Mantiquilla

November 27, 2020

Contents

1 Introduction

This report will focus on explaining the various ways in which one can measure the software engineering process. It outlines what platforms or frameworks are available to perform measurement. As well as that, the report will go through the algorithmic approaches and the ethical concerns surrounding the measurement of the software engineering process.

2 Why measure software engineering

There are many benefits to measuring software engineering. Most notably to produce higher quality code, to reduce the cost of software development and to improve the speed at which software development takes place.

2.1 More specifically, measuring software engineering:

- Helps in identifying possible software defects early in development
- Increase the workload that the current staff can accomplish
- Identify high-output employees to receive rewards and low-output employees to receive help and training.
- Increase the complexity and value of the software with the same staff effort.

[12]

It is clear that measuring software engineering has many benefits. Thus, many companies strive to create and/or use services which perform such measurement. However, there are also some ethical concerns and inefficient methods of measuring software engineering. These will be explored further in the next section.

3 How software engineering can be measured

There are many individual metrics which can be measured during the development of software which may indicate progress. Most of software engineering is measured using a version control system. Version control systems generate vast amounts of data which can then be measured. The main metrics that this report will go through are lines of code, number of pull requests and code review turnaround time.

3.1 Lines of code

The simplest metric to measure would be the lines of code (LOC). LOC is a simple and crude method to measure the productivity of a software engineer.

Using LOC rewards bad coding practices and less efficient code. [6] This is evident in the short Java code example below.

3.1.1 Version 1

```
0 for (int i = 0; i < 5; i++) {
1   System.out.println("Hello world");
2 }
```

3.1.2 Version 2

```
0 System.out.println("Hello world");
1 System.out.println("Hello world");
2 System.out.println("Hello world");
3 System.out.println("Hello world");
4 System.out.println("Hello world");
```

The versions 3.1.1 and 3.1.2 of the code produce the same output. However it is clear which version of the code is more desirable. The version 3.1.1 of the code follows good coding practices. It is DRY (don't repeat yourself). The version 3.1.2 on the other hand is a bad quality code compared to the first version.

Other reasons why using LOC as a metric to measure productivity is not a good idea is that it does not take formatting into account. It is also not language agnostic. This is seen in the examples below;

3.1.3 Version 1

```
0 if (i < 2) {
1   i++;
2 } else if (i > 10) {
3   i--;
4 }
```

3.1.4 Version 2

```
0 if (i < 2) { i++; } else if (i > 10) { i--; }
```

3.1.5 Version 3

```
0   cmp rcx, #2
1   jge ifIge2
2   inc rcx
3   jmp eIf
4 ifIge2:
5   cmp rcx, #10
6   jle eIf
7   dec rcx
8 eIf
```

Yet again all three versions of the code is equivalent however it is not clear from the LOC which software engineer is more productive.

3.2 Number of pull requests

The general idea is that the more pull requests completed, the more productive a team is. This method however does not measure productivity well. Like counting lines of code, using the number of pull requests created also favors bad programming practices. It encourages small and unnecessary pull requests. The size and the difficulty of the pull request is also not taken into account.

Figure 1: Example number of pull requests metric, courtesy of Medium [7]

3.3 Code review turnaround time

A more modern approach to measuring software engineer productivity would be to measure process metrics instead of product metrics. Some examples of product metrics would be the lines of code or the number of commits / pull requests completed.

The code review turnaround time is an example of process metrics. Whenever code is added or changed in a code base, it must undergo a code review process. This usually involves other coworkers looking over code and making suggestions before the code can be merged into the main branch. The general idea is to measure how long it takes until a code review is done.

According to Abi Noda measuring process metrics is a great way to increase productivity. Process metrics directly impact the developer experience in a positive way. It is also a great way to show how the organization is improving. [8]

4 Platforms available

Measuring the software engineering process is a valuable process. Thus, many services have been created to aid companies in analyzing and displaying this process. The data provided by the services help employers visualize whether or not the ongoing projects will be delivered on time, see which employees have a high impact on the company, etc.

4.1 Pluralsight Flow

Pluralsight Flow, called GitPrime previously, is a service which obtains data from git host, e.g. GitHub, and the git information from repositories which the engineers work on. [1] The data is processed to produce useful metrics to indicate the performance of software engineers.

Version Control Systems like git generate a large digital footprint for the developers that use it. Information like the number of commits done by a developer, the lines of code changed by a developer, is produced through the use of a version control system. These metrics alone may not be a good indication of the productivity of a software engineer. However, when processed together, a clearer picture is obtained.

Pluralsight Flow, for example, will use the information found from pull requests (PRs), tickets linked to said PRs and commits to produce useful graphs and metrics.

4.1.1 New metrics extrapolated by Pluralsight Flow:

- Churn — The amount of code which is changed or deleted shortly after it was written. A high amount of churn may indicate that something may be affecting the development life cycle.
- Efficiency — A percentage evaluating whether the committed code is productive work. This metric is directly related to the churn metric.

- Impact — A score given to a commit which measures the effect of changing or adding specific lines of code. This metric uses many factors to determine the score such as the amount of code changed, the number of files affected, the percentage of old code edited and many more.

[2]

4.2 Azure DevOps services

Azure DevOps services is a collection of developer services provided by Microsoft to facilitate the software engineering process. There are many Azure DevOps services available. This report, however, will focus on the Azure Boards and the Azure Pipelines service in particular. [3]

4.2.1 Azure Boards

Azure Boards is a service which allows software engineers to track and manage their work for software projects. Azure Boards is built with software engineering processes in mind and has native support for Kanban and Scrum. It offers built-in scrum boards and and tools to help the team run sprints and stand-ups.

The basis of Azure Boards is an entity called the 'work item'. This entity tracks all the work done for the software project. A 'work item' communicates the details of the work to be completed to the team. Each 'work item' has a status attached which is an indication of the progress of the work. This allows the whole team to be on the same page with regards to what work is assigned to who and whether or not it has been completed.

Each 'work item' has a rich set of information attached to it. It contains information such as links to discussions regarding the 'work item', any commits, PRs or tickets related, as well as a history of all changes to the 'work item'.

The information collected through the use of Azure Boards is analyzed and presented on a dashboard which indicates the health of the project. The dashboard contains all the metrics gathered, such as the velocity of the project or average completion time, in one place. [5]

4.2.2 Azure Pipelines

Azure Pipelines is a cloud continuous integration (CI) and continuous delivery (CD) service. That is, a service which builds, tests and deploys your code automatically. This service is used in conjunction with a git host such as GitHub or Azure Repos.

The metrics which Azure Pipelines provides are the pipeline pass rate, the code test pass rate and the average pipeline duration. The information for the pipeline analytics is obtained from the pipeline runs.

5 Algorithmic approaches

5.1 COCOMO

COCOMO (Constructive Cost Model) is a regression model which is used to estimate the cost, effort and the development time to produce a software project. This model was produced by Barry W. Boehm [4].

Software projects which use the COCOMO model are classified into three categories which are organic, semi-detached and embedded. The definition of each is found in figure 2 below.

	Organic	Semi-detached	Embedded
Project size (lines of source code)	2,000 to 50,000	50,000 to 300,000	300,000 and above
Team Size	Small	Medium	Large
Developer Experience	Experienced developers needed	Mix of Newbie and experienced developers	Good experience developers
Environment	- Familiar Environment	- Less familiar environment	- Unfamiliar environment (new) - Coupled with complex hardware
Innovation	Minor	Medium	Major
Deadline	Not tight	Medium	Very tight
Example(s)	Simple Inventory Management system	New Operating system	Air traffic control system

Figure 2: Definition of the different COCOMO categories, courtesy of Medium [10]

The different categories influence the algorithms used during the calculation of the estimates.

The COCOMO model has three different models which allow the software manager to specify the granularity of the estimates.

The different types are:

- Basic COCOMO model — This model is quite limited and restricted as it uses kilo lines of code for the calculations. This model does not take into account the other factors for the estimations.

- Intermediate COCOMO model — This model builds on top of the basic COCOMO model and takes cost drivers into account which further enhances the estimate. A cost driver is a term used to define factors which have an effect on the effort, cost and time during the development phase of a software project.
- Detailed COCOMO model — Yet again this model builds on top of the basic and intermediate model and incorporates the Waterfall model e.g. the cost estimates are completed at every step of the waterfall model. This is the most accurate model available in the COCOMO models.

5.2 Machine learning

"Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience." [11] Nowadays machine learning is used extensively. They power things from self-driving cars to spam filtering. The applications of machine learning are endless.

In order to use machine learning, one must have a large data set to work with. One of the side effects of using a version control system is that it produces a large data set regarding software engineering. One application of machine learning is to process the data from version control systems to measure software development productivity.

Firstly, we must define what it means to be a productive software engineer. A highly productive software engineer is one which produces more source code and source code of a higher quality. Through the use of machine learning techniques, the complex code structure produced by software engineers is reduced into quantitative measures of quality and quantity. [9]

6 Ethics

Based on the research that was presented in this report, there seems to be little to no ethical concerns as the methods aforementioned above are not intrusive in any way. However, negative effects can be introduced with improper measurement such as the use of lines of code or the number of commits as a metric. The use of improper metrics may encourage people to perform negatively.

Regardless there are other methods of software engineering measurement which can pose some ethical concerns due to their invasive nature. This is a generally grey area; the level of invasive nature of the other methods can depend on the individual whose performance is being measured. Moreover, there are a number of benefits that accompany the other methods that help improve software engineering. Therefore, it becomes difficult to completely condemn those practices, and remains to be an area of constant discussion among the software engineering populace.

7 Conclusion

There are many ways to measure software engineering, most of which is a by-product of using a version control system. Many services and platforms have been created to aid companies and organizations analyze the data which has been collected such as Pluralsight Flow and Azure DevOps Services. Many algorithmic approaches exist to process the data collected to aid the software engineering process.

However, it should be of note that while measuring software engineering provides a lot of benefits, it also brings about a number of ethical concerns among the software engineering group. It is an ever-evolving practice; many other methods of measurement will undoubtedly be developed or improved upon. Therefore, one should not completely disregard one method of measurement based on ethical concerns. Instead, the measurement method should be viewed as a whole, advantages and disadvantages alike, in order to aid the greater picture of improving software engineering as a practice.

References

[1] Pluralsight flow. https://help.pluralsight.com/help/chapter-2-flow-import-and-manage-your-data, 2019.

[2] Pluralsight flow. https://help.pluralsight.com/help/metrics, 2019.

[3] Azure devops services: Microsoft azure. https://azure.microsoft.com/en-us/services/devops/, 2020.

[4] Boehm Barry et al. Software engineering economics. *New York*, 197, 1981.

[5] Aaron Bjork. Deep dive into azure boards. https://azure.microsoft.com/en-us/services/devops/, 2018.

[6] Norman E Fenton and Martin Neil. Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3):149–157, 1999.

[7] Simon Gerber. Looking for metrics in all the wrong places. https://medium.com/better-programming/looking-for-metrics-in-all-the-wrong-places-db63ca81d4e9, 2020.

[8] GitHub. The elusive quest to measure developer productivity - github universe 2019. https://www.youtube.com/watch?v=cRJZldsHS3c, 2019.

[9] Jean Hélie, Ian Wright, and Albert Ziegler. Measuring software development productivity: A machine learning approach. In *Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC*, volume 18, 2018.

[10] Warakorn Jetlohasiri. Cocomo: A procedural cost estimate model for software projects. https://medium.com/@warakornjetlohasiri/cocomo-a-regression-model-in-procedural-cost-estimate-model-for-software-projects-65ab5222a1f5, 2020.

[11] Tom M. Mitchell. Machine learning definition. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html, 1997.

[12] Walt Scacchi. Understanding software productivity. In *Software Engineering and Knowledge Engineering: Trends for the next decade*, pages 273–316. World Scientific, 1995.