Measuring Software Engineering – A Report

Sean Roche - 17332021

Table of Contents

1 Introduction 2
1.1 Why Measure Software Engineering? 2
2 Software Metrics 3
2.1 Agile Metrics 3
2.2 Production Metrics 4
2.3 Size Oriented Metrics 5
3 Algorithmic Approaches 6
3.1 Putnam Model 6
3.2 Functional Point Analysis 7
4 Overview of Available Platforms 8
4.1 GitHub 8
4.2 GitPrime 8
4.3 Code Time 8
5 Ethical Concerns 9
5.1 Misuse of Software Metrics 9
5.2 Data Privacy 9
6 Conclusion 10
7 References 12

1 Introduction

In this report I will outline the methods in which we can measure the software engineering process. I will discuss the metrics that can be gathered in software engineering, the algorithmic approaches available and the platforms that implement these ideas. In doing so I will attempt to analyse the logic and rationality to the well-known non-sequitur claim that, "the performance of engineers cannot be measured". Lastly, I will highlight some of the ethical concerns associated with measuring the software engineering process.

1.1 Why Measure Software Engineering?

Before analysing the process of measuring software engineering, we must first understand the incentive. Software development is a key factor of success for businesses and organizations that rely on complex computer programs to provide their services. Every software driven organization's goal is to achieve maximum efficiency with money and effort being applied where most needed. As a result of this it seems that logical that we should be able to measure and analyse the software engineering process.

Advantages of gathering this kind of data may include:

- Enabling teams to identify characteristics of their workflow. A project manager/team leader may want to build a statistical model, gathering information like what times developers are most productive, what team size is optimal for different tasks. They might also identify areas of low/high productivity. This data can then be used to allow for better estimation and planning in the future, leading to a more efficient workflow.
- The data can allow for better regulation of the state of a project in relation to budget and schedule.
- Software development teams can use software metrics to better communicate the status of projects. It allows them to highlight and address pressing issues so they can be monitored.
- Collecting this kind of data in large quantities may have some monetary value in the future. After streamlining their own model, an organization/ individual may seek to sell data. This data could be used by others who want to improve their software engineering workflow.

2 Software Metrics

A software metric is a measurement of a software characteristic that is quantifiable. Software metrics allow us to represent aspects of software engineering numerically. They are important in measuring software performance, engineer productivity and many other use cases.

In the software engineering process, there are many metrics we can collect for any given project all of which are related to each other. When choosing which of these metrics to use it is important that they are relevant and will add value to the process making engineers more effective and code more efficient.

In this section I will separate some metrics into categories or types and discuss their features and limitations.

2.1 Agile Metrics

Agile metrics are software engineering metrics that are used when measuring the performance of an Agile team. Before we look at the metrics for measuring the performance of Agile teams, I will give a brief description of Agile.

Agile software development is a group of software methodologies based on iterative development. In Agile, requirements are tackled by groups of self-organized teams that are cross-functional. The management process for agile generally promotes frequent inspections and adaptation while of encouraging teamwork and self-accountability.

Scrum, is a subset of Agile. It has some of its own set of practices but is still a subset of Agile. Some of Agile’s distinguishing characteristics include the use of development cycles called “sprints” and an “XP” framework that requires pair work. [1]

Agile Metrics focus on how an agile team plans and makes decisions. They don’t describe the software itself but can be used to enhance the software engineering process.

Cycle Time

Cycle time is a measure of the elapsed time from when work starts on an item until it’s ready for delivery. In this way, cycle time indicates how long it takes to develop and implement in production.

Cycle time is extremely easy to track, with only two times needing to be recorded for any given task. However, cycle time does not account for the amount of work involved in any given task and this should be taken into account when tracking and using the metric.

Team Velocity

Team velocity is an agile metric that measures how many software items a team completes in a sprint (development cycle in Scrum). Since each team has different tasks, this metric should not be used to compare agile teams. However it may be used to estimate the performance of a team that is assigned similar tasks for a number of iterations. In general, high velocity indicates that a team is being productive or performing efficiently. Much like cycle time, it is affected by the size of the items to be completed.

Open/Close Rate

This metric measures the rate at which production issues are opened and subsequently, closed over a set period of time. Identifying unhealthy trends for this metric could lead a team trying to reduce issues, improve QA or spend more time on production issues as they arise. [2]

2.2 Production Metrics

Production metrics are measure of how much work is completed and the efficiency of a software development team. [3]

Code Churn

Code churn is a metric of the number of lines of code, added, modified or removed over a period. Code churn is an extremely useful metric. If code churn displays high values, it may indicate that the project requires the attention of a software engineer and is becoming ineffective. A corollary to this is that if it displays low values it may indicate that development is effective with little code wasted.

One of the shortcomings of code churn is that the volume of code produced does not correlate with the code’s effectiveness or quality. Much like in SLOC a developer could produce thousands of lines of code where in reality the optimal solution would consist of a fraction of that amount. A developer could also potentially increase code volume

intentionally when writing code to look better when assessed which leads to inefficient development.

Another issue is comments and whitespace. These can have a huge effect on the result, leading to inaccurate assessment and assumptions.

Active days

Active days asses how often a software developer contributes code to a software development project. It does not account for planning or administration but simply assesses the cost of interruptions.

A point of concern when using this metric in the real world is that the number of days a developer has been "active" on a project for a given period may not accurately represent the value of their contributions or even their frequency. It may not account for hours where they were absent or were planning code or external administration that was not acknowledged by the given system.

Assignment Scope

Assignment Scope, in basic terms, is a measure of the amount of code that a programmer can maintain and support. It can be used to plan how many people should be assigned to support a certain software system. It is also often used in comparing teams.

2.3 Size Oriented Metrics

Source Lines of Code (SLOC/LOC)

Source lines of code (SLOC) is a metric that generally evaluates the size of a program or codebase. The metric is useful in communicating to software engineers whether their team isn't producing enough code or alternatively when their code isn't "lean" enough or needs to be more concise or efficient.

SLOC comes with many drawbacks. Much like what was mentioned previously for Code Churn it lacks accountability. Many people believe it useless to measure the productivity of a project based only on the coding phase without accounting for any planning/design. While there is often correlation between SLOC and the productivity of a developer/team it also

does not speak for quality. Another point worth mentioning is that different languages produce very different quantities of code and with GUI -based programming languages like VS Basic programmers can achieve a lot of functionality with minimal lines of code. Psychology also suggests that using SLOC to measure the performance of a developer may incentivise writing unnecessary lines of code. Even with all of its other disadvantages aside it is difficult to decide what passes as a line of code in the counting process, comments and whitespace may affect the result

3 Algorithmic Approaches

Algorithms can be used in measuring the software engineering process by allowing for cost estimation which can then be compared to team performance. These algorithms provide mathematical equations based on research and data from the past. They input metrics such as Source Lines of Code (SLOC) as well as other cost factors such as language, skill levels etc. [4]

We also see the use of algorithms in measuring the complexity and functionality of software.

3.1 Putnam Model

The Putnam model is an empirical software cost/effort estimation model. The original model was published by Lawrence H. Putman in 1978, it was a pioneering piece of work for that time. [5]

Empirical models collect data from other software engineering projects (e.g. size and effort) and use it to form an equation. This equation is then used to make future estimates of effort based on the estimated size of the project.

The Putnam model can be defined as:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

Note: Calculating for effort is a common form of the model

Where:

B = Scaling factor (function of the product size)

Size = Product size (Estimated using a variation of the LOC metric)

Productivity = Process Productivity

One of the main advantages to the model is that software organizations that are mature can easily gather size, effort and duration (time) metrics from past projects. Organizations may convert exponential "Process Productivity" into a linear productivity index for use in estimations.

The main drawback to the model can then be said to be the fact that the Putnam model is reliant on the user knowing or being able to accurately estimate the size of the software to be developed.

3.2 Functional Point Analysis (FPA)

Functional Point Analysis is a widely used algorithmic estimation technique that was created by Allan J. Albrecht in 1979. The initial definition given by Albrecht was:

"FPA gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer." [5]

It is a method that of Functional Size Measurement, used to assess the functionality delivered by software. It focuses on the logical view of the software, assessing the functionality it can deliver to end point users. It does not assess the internal technical view of the software or "care" about how it is implemented.

FPA has many use cases in measuring software engineering:

- It is commonly used to estimate cost and resources required for software to be developed and maintained.
- It is used for software comparison.
- It measures the units of a software product so that quality and productivity can be better analysed.

4 Overview of Available Platforms

4.1 GitHub

GitHub is a hosting service for Git, a Distributed Version Control tool. While GitHub's primary function is just to host repositories, it does offer some tools and insights that can be used by management to measure the performance of a software development team.

GitHub integrates a web based graphical interface that makes viewing a developer's contributions to a project quite simple. It also graphs and compares the quantity of commits for different users and presents the commit history in an easy to understand manner.

4.2 GitPrime

GitPrime is a popular engineering logistics platform for commercial use. It uses data from Git repositories to give management software engineering metrics like those discussed earlier. With this information teams can optimize their workflow and patterns in their development. It aggregates data from commits, merges, issues, pull requests and more and presents the user with a number of dashboards to represent this data visually in a clear and concise manner.

The metrics provided by GitPrime gives answers to management on topics like, how much a team is refactoring old code, what was accomplished in a week, how well is knowledge shared in code review and comments etc. [6]

4.3 Code Time

Code Time is an open source extension for VS Code that, like GitPrime, tracks many of the software engineering metrics mentioned in this report. What makes Code Time different is that it provides programming metrics conveniently in the developer's code editor.

A major advantage to Code Time is that it presents generated dashboards from VS Code menu commands. This makes it more convenient for a developer to view their own progress as they don't even have to leave the editor.

Code Time also integrates with Visual Studio Live Share so that you can see the progress and productivity of other team members. [7]

5 Ethical Concerns

5.1 Misuse of Software Metrics

Software metrics should answer questions, but the question that needs to be asked is rarely the KLOC figure for the week. As discussed previously, no software metric carries significant weight in isolation.

Developers may struggle with the idea of their performance being assessed by management or other team members using software metrics that are not comprehensive. This brings me back to the point of quality over quantity. A developer could clock up thousands of lines of code and appear to be the greatest contributor when in reality another developer could be producing more efficient code that is more valuable. While by using a number of software metrics and algorithmic approaches we can make certain assumptions, in many cases misuse of this data can lead to a false narrative about team productivity.

5.2 Data Privacy

Measuring the performance of software engineering involves collecting and storing data on developers. GDPR (General Data Protection Regulation) is a hugely in the software industry with massive corporations and smaller organizations alike both having to abide by the rules. The basic principles of GDPR can be summarised as follows:

- The collection of the data must be lawful, fair and transparent
- The purpose of the data collection must be limited
- Personal data that is not in focus should be minimised
- The data collector must ensure that the data is accurate and fit for the purpose intended.
- The amount of time you store personal data should be limited to amount of time needed for the purpose of the data.
- The integrity and confidentiality of personal data should be guaranteed with security preventing unauthorised access.
- The controller is responsible for and must demonstrate compliance with the above.

[8]

While collecting data on teams or developers isn’t currently a topic of widespread dispute, it is important that organizations are transparent in discussing how this data will be collected and maintained. The lifespan of the data and its endpoint should be made clear and, in certain cases, measures may be required to ensure that it is guaranteed.

Sale of Data

If a company does make the decision to sell data and has abided by the GDPR regulations outlined above they may also intend to sell the data.

Although they have, in this case, been transparent with their intentions and may have stated they will sell or share the data, this is still another point of ethical concern. While the GDPR regulations are enforced to ensure a user’s data remains private there is no guarantee that it that third parties will abide by it.

Another point worth mentioning is, does an organization have the right to profit of its member’s data? While the handling of developer data hasn’t been widely disputed yet, this could be a huge ethical concern in the future.

5 Conclusion

In software engineering there are numerous tools, metrics and methods we can use for measuring the software engineering process.

Software metrics, when used correctly, tell us useful pieces of information. Most metrics come with shortcomings, for example not accounting for quality when measuring quantity. For this reason most of the time one metric alone is not adequate when measuring performance.

We also have a long list of algorithms that combine, historic data, estimates and conditional constants to do things like estimate the resources needed to carry out a task or project. While these are obviously subject to a degree of error, they are useful resource that can effectively measure a team’s performance.

I also discussed the plethora of existing tools, for example GitPrime that already compile these principals into a powerful tool for the developer.

There is always ethical concerns surround collecting any type of data and measuring software engineering is no different. The other, more specific concern I mentioned was the incorrect use of software metrics. This is an issue that could lead to inaccurate conclusions, false assumptions and just generally prove problematic for a developer or team.

While people continuously try to present software engineering as an art that cannot be measured, the research outlined in this report suggests that it can. Despite the fact that some projects have a lot of underlying complexity, and can’ t be measured using SLOC alone, there is no logic in suggesting it can’ t be done. As long as we continue the effort to measure engineering, it should only get easier.

6 References

[1] “Agile Alliance,” [Online]. Available: https://www.agilealliance.org/.

[2] “Atlassian Agile Coach,” [Online]. Available: https://www.atlassian.com/agile/project-management/metrics.

[3] “stackify,” [Online]. Available: https://stackify.com/track-software-metrics/.

[4] “DCU Software Cost Estimation,” [Online]. Available: https://www.computing.dcu.ie/~renaat/ca421/LWu1.html.

[5] “GeeksForGeeks,” [Online]. Available: https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/.

[6] “GitPrime,” [Online]. Available: https://www.gitprime.com/.

[7] “Code Time for VS Code,” [Online]. Available: https://marketplace.visualstudio.com/items?itemName=softwaredotcom.swdc-vscode.

[8] “Data Protection Commission,” [Online]. Available: https://www.dataprotection.ie/.

[10] “Scrum,” [Online]. Available: https://www.scrum.org/about.