


Report: Measuring Software Engineering

To 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.

Isobel Mahon
17331358

Table of Contents

| | |
|----------------------------------|----|
| Table of Contents | 2 |
| Introduction | 3 |
| Measurable Data | 3 |
| Algorithmic Approaches Available | 6 |
| Ethics | 7 |
| Alternative Approaches | 8 |
| Conclusion | 9 |
| References | 10 |



Introduction

Most employers want to know the quality of their employees and potential hires. Everyone wants to hire and retain the best people in the business, in any industry. In software in particular, the disparity between a good engineer and a bad or mediocre one can be huge. As such, it is very important for employers to be able to quickly identify the good ones. Considering this, the study of how to measure the quality of a software engineer is an important one.

Measurable Data

There are many different ways in which the software engineering process can be measured, and many different types of measurable data can be gleaned. Keystrokes over time, tasks completed over time, lines of code, number of commits, to name some basic ones.

There are many platforms available that can measure this sort of data, eg. git and SVN can measure commit frequency, total number, bug count, line counts, etc. Similarly, there are many services available to measure things like keystrokes, such as KeyCounter³ and WhatPulse⁴. There are many platforms capable of measuring these metrics.

However, there are upsides and downsides to all of these.



There is no shortage of measurable data, but it can be difficult to get any useful information from it. For instance, keystrokes over time could mean almost anything; they could be constantly writing and deleting the same thing. They could be browsing the internet, playing video games, or just deliberately cheating the measurement system. Tasks completed over time could encourage engineers to prioritise the quickest, easiest tasks. Lines of code could promote verbose programming, number of commits varies more or less randomly from person to person, and at worst could promote quantity with no regard for quality and so on.

Essentially, most of the easy, basic measures of software engineering either greatly favour certain styles of programming, are very easy to cheat, or both. It seems initially possible that a combination of several types of data could help to offset this problem, but really all of the measures suit one type of programming: the more you do, the better a programmer you appear to be. At first glance, that seems reasonable, but the reality is a lot more complex.

More lines of code do not necessarily accomplish more, and more keystrokes do not necessarily imply either more code or even more effective lines. They could be a result of a lot of code, a lot of documentation, or merely indecisiveness. More lines, more commits, more keystrokes could all be a symptom of programming via trial and error. The reality of programming is that there are many ways to go about it, and that sometimes what might seem to be doing very little from a data viewpoint can actually be more effective than a busywork approach. Often, programmers who have hit a difficult problem will take time to think about it, which may involve staring blankly at a screen, writing on paper or a whiteboard, or discussing the problem with colleagues. This time of apparent inactivity can be some of the most productive time spent on a problem.

One other potential way of measuring productivity is pull requests (PRs). PRs are a standard practice in the software development process, and are supported by many platforms such as GitHub, Atlassian, and Azure DevOps. In terms of measurement, the number of issues found in a developer's PR combined with the frequency of PRs could be used as a measure of efficacy. Similarly, the number of issues found or comments made on a colleague's PR could be counted in a developer's favour. Unfortunately, this kind of system would introduce a whole other problem, namely it could promote competitiveness and make good relations hard to maintain in a team. This system is no harder to cheat than any of those above, but the consequences of cheating it could be harmful to others in the team. To avoid this problem, the incentive could be removed to make comments, or the penalty in receiving them could be removed. However, if the incentive to make comments was removed, and the penalty on receiving them kept, most likely far fewer comments would be made, even in cases where they should be. That's not necessarily to say they wouldn't be dealt with, but it would defeat the purpose of measurement if the comments and issues were communicated on a different platform. Conversely, if the penalty was removed and the incentive left, most likely developers would simply game the system guilt and consequence free by spamming PRs with superfluous comments.



Algorithmic Approaches Available

As discussed above, the simplest algorithmic approaches such as line counts and commit frequencies are generally not able to accurately determine the efficacy of a software engineer. In response to this, some have considered various algorithmic approaches. In the paper ‘Software metrics: successes, failures and new directions’ by Norman E. Fenton and Martin Neil, they discussed how “The [Bayesian belief nets] approach is actually being used on real projects and is receiving highly favourable reviews.”¹ This approach can be used to algorithmically make various predictions and assessments of software, but does not directly measure the ability of an individual engineer. Stochastic reliability growth models¹ can predict the reliability of software models in some cases, but again, do not directly measure the developer. There are many examples. There are many software measurement models, but they are either irrelevant or do not work properly.

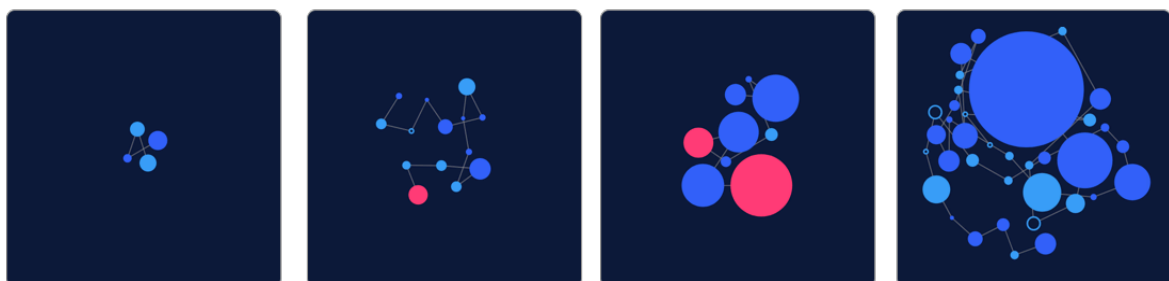
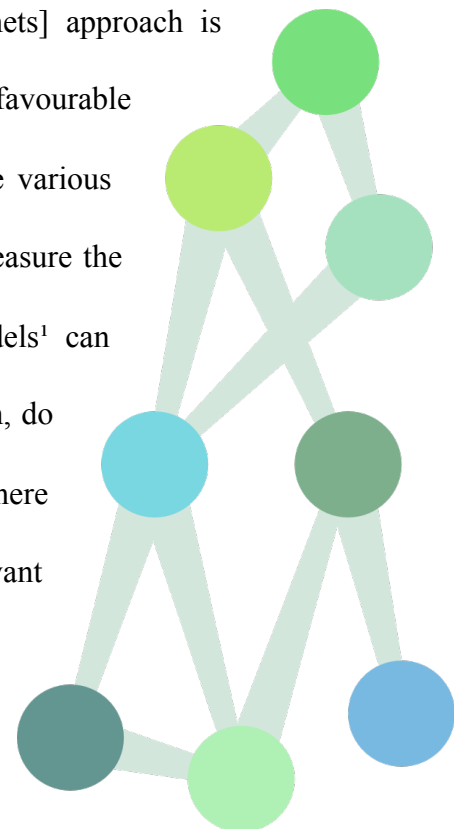


image from <https://hackernoon.com/>

Ethics

There are quite a lot of ethical concerns surrounding the collection and usage of any kind of personal data, the quality of software engineering being no exception. In particular, what the data is being used for is of considerable concern. In any situation where data will be used to make decisions, in particular big decisions like the hiring and firing of employees, ethics must be carefully considered.

As such, the problem of measuring engineering is far more complex than simple data collection. In fact, the process is complex to the point where many experts argue that it is a pointless endeavour: there is no reasonable way to objectively measure a developer's productivity. According to Joel Spolsky in his article 'Hitting the High Notes', "It's rather hard to measure programmer productivity; almost any metric you can come up with (lines of debugged code, function points, number of command-line arguments) is trivial to game, and it's very hard to get concrete data on large projects because it's very rare for two programmers to be told to do the same thing." ⁶ As said by Steve McConnell, "So while I see the value of measuring individual performance in research settings, I think its difficult to find cases in which the effort is justified on real projects." ²

It can be easy enough to verify whether data is reliable, although not always, and it is very difficult to verify whether data is valid. As discussed above, in almost all cases, it is possible to game the system being used to measure productivity, and even where software developers make no effort to artificially improve their standing, the results will vary wildly

depending on their style of programming, and often reflect very little of what we are trying to measure.

Given all of this, it is very difficult to make a case for using such unreliable methods to determine how a person's career should or should not be advanced.

Alternative Approaches

While useful numeric metrics are very hard to come by, it is possible to know how good a software developer is. The way to find out is to work with them. This method is not quick, and it is not objective, but the information obtained will be far more relevant than any algorithmic approach tried to date. We talk, and ask questions, and complete pull requests, and as a result we develop an understanding of each other's abilities. In order to know before



hiring someone, there are internships and references. The industry knows this; software engineering internships are common and well paid, and often give realistic industry experience in order to allow an employer to properly evaluate a potential employee. Similarly, structured references are commonplace. Structured references ask specific questions in order to get the necessary information and avoid overly vague testimonials. Probationary working periods serve a similar function to internships. All of these methods are imperfect, but software engineering involves skills like problem solving and communication, which are far easier to get an impression of than to quantify. It is to some degree more art than science, and does not measure well numerically.

Conclusion

It is very difficult to automatically measure the productivity of an individual software engineer. There simply aren't sufficiently reliable and valid metrics. A lack of objective, computerised measurement does not mean we can't know how productive a worker is. We have been managing without objective numerical measurements of our efficacy for as long as people have been employed. In fact, most of us know quite well how effective our colleagues and employees are. We figure this out over time, from working with them. Of course, that doesn't work so well when we want to know whether to hire someone; in that case, we must rely on a process of interviews, references and a probationary working period. It is not a perfect system, but it is probably more reliable than counting the lines of code they've written.

References

1. Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.
2. <https://dev.to/nickhodes/can-developer-productivity-be-measured-1npo>
3. <https://www.ghacks.net/2009/03/30/count-your-keystrokes-every-day/>
4. <https://whatpulse.org/>
5. <https://hackernoon.com/measure-a-developers-impact-e2e18593ac79>
6. <https://www.joelonsoftware.com/2005/07/25/hitting-the-high-notes/>