Measuring Software Engineering Report

Sarah Klein, 19321148

January 3, 2022

Contents

1 Methodology

Analyzing a software engineer's output isn't a quick task - it requires someone to sit down, read the code, read the documentation, browse the repository, and that's simply for a basic analysis. So, wouldn't it be wonderful to automate that? After all, a script can analyze thousands of lines of code in the same time a human can read a function or two. So, to do this, there's a number of techniques to try.

1.1 Code Complexity

Code complexity seems like a straightforwards metric - after all, if the code is complex, there must have been a good deal of effort put into its engineering, right? Unfortunately, this isn't perfect. If you define it as the number of functions and subclasses a coder has created, you will find simple, straight-forward solutions, which often require a great deal of development to write, are rated worse than spaghetti-code with redundant, unused, or wasteful functions, classes, and methods. Perhaps implementing merge sort into my function will make it count as more complex, but it's likely to perform worse than using my coding language's built-in sort method.

1.2 Commit Frequency

Commit frequency is a better metric, but it's not perfect either. Though simple, the frequency of commits can show effort, especially over time. If you consider the numbers as-is, without examining the contents, then a coder who commits, makes one tweak, and then re-commits, is apparently more productive than a coder who only commits once, without needing to tweak and resubmit. That said, if one takes that into consideration, or combines the commit frequency with another metric, you can get a much, much more accurate analysis method.

1.3 Personal Considerations

I opted to use Commit Frequency for my visualization. I understand that it is easily spoofed and is certainly not perfect, but the idea for my program was less centered on analysis and comparison, instead opting for visual conveyance. I wanted something clearly understandable by the user, something they could understand at a moment's glance, even if it wasn't the most revealing data set.

2 Data Sourcing

2.1 Repositories

The source of data for these types of analytics is a straightforward choice - use the version control system the company already uses. With services such as GitHub and Subversion becoming incredibly popular and widespread, this data is growing more and more available. GitHub provides a robust API schema for interacting with repositories, analyzing them, and even altering them.

3 Data Handling

3.1 Services

The computational power to process this data, as well as the capacity to store it, is thankfully not a problem in the modern day. Thanks to services such as *Microsoft Azure*, *Google Cloud*, and *Amazon AWS*, which are already very prevalent in the Computer Science and Technology fields, the requirements are within reach for even a small-business owner. Subscriptions are relatively inexpensive, especially when compared to constructing and maintaining your own datacenter. Additionally, there's been an increase in the number of tools provided either for free or for a cost, as we are seeing a new market growing in this niche.

4 Ethics

Ethics are always going to be contentious in analytics. As is, there are no ethical quandaries over simply making an analytic, it's the context surrounding it - how the data powering it is obtained, and how the analytic is used. One only needs to look at controversies like Amazon's blunder with an application-ranking AI that wound up failing in its duty.

4.1 Obtaining

Obtaining the data is one hurdle that thankfully, isn't too likely to derail any analytic attempts. There are issues, of course, but were any company seek to create a metric for analyzing their programmer's effectiveness, they would use in-house data rather than random people on the internet. But for a small-time project or an individual experiment? There might be some issues. Sure, there's plenty of information you can grab and experiment with on sites like GitHub, but just because it's there doesn't mean you can share it with everyone. This was one of the reasons that I only tested my visualizer on my own GitHub accounts, as I didn't feel comfortable "spying" on strangers on the internet.

4.2 Using

The utilization of the metric is the large issue, however, especially if the metric is imperfect (which can be safely assumed to be true). If used blindly, it can cripple the morale and work ethic of an entire company. For example, say a company decides that year-end bonuses will be divvied up by how active their coders were. If the metric is simply commits, then someone who makes a thousand useless commits a day will beat someone who commits two times a day, but with completed, tested, and useful functions. If the metric is some arbitrary measure of complexity, then all one needs to do is make every basic operation require five function calls, and they've managed to "win" compared to someone who makes clean, clear code.