Measuring Software Engineering

The question of how to accurately measure the productivity and/or the quality of a software engineer is a long standing one. It is not as simple as counting the number of cars a salesman in a dealership sells, or the number of patients seen by a doctor in practice. There is no clearly outstanding metric by which to gauge the output of a software developer.

This question is of extreme importance to the ever expanding number of companies employing teams of developers in our software centric world. Software Engineers command a high salary, with payscale.com reporting an average entry-level salary of $76,000 in the US. To any company spending this amount of money, measuring what they’re getting for their dollar is paramount, allowing them to ensure their money is well spent and that they’re getting the best out of their employees.

So what data is available to companies to track their engineers’ performance from day to day? The obvious first port of call is number of lines of code shipped or amount of time spent coding. Does this mean that a developer implementing a feature with 200 lines of code is being less productive than the developer who takes 1000 lines and the same amount of time? Obviously we must ascribe value to other aspects of a developer’s output rather than simply the volume. With modern design practices such as agile development, there are much more telling data sources than simply output amount. The amount of time taken to resolve a particular issue found in a previous build for example, the amount of technical debt incurred while adding a new feature or recoding certain sections of a project. A secondary aspect to consider is the developer’s role within whatever team they are working, they may be incredible on their own, but can they perform in a team based environment? This kind of information is just as crucial as the number of lines of code written, if not more – assuming our engineers are all of at least a passable standard.

|  |  |
| --- | --- |
| **Type of Data - Code** | **Pros/Cons** |
| Number of lines of code shipped | + Clear sign of effort and work done  - Not reliable, number of lines does not represent quality of code |
| Time spent coding | + Again, displays effort and commitment to project  - Quality and efficacy of code not taken into account. |
| Git Commits | + Provides a more accurate, in depth look at the contribution of the developer.  + Both additions and removals of code can be tracked and considered.  - Requires a lot of analysis and can be gamed if developer knows what the system is looking for |
| Technical Debt (Bugs introduced) | + A solid marker for how good an engineer someone is, how many new bugs appear when they ship a new build  - When working in teams the source of these bugs may be hard to track and may not even be the fault of a single person potentially allowing for false positives to occur. |
| Test Coverage | + Shows understanding of code  + Asserts that code is stable and fit for purpose  - Tests can be written to pass, do not ensure that code cannot crash, if all possible scenarios are not covered |

In a large team, within an even larger company, a lot of non-code data is available which – although it may not directly show the proficiency of a software engineer – can give an incredibly useful insight into how they interact with their co-workers, managers, and how they perform as a part of the machine that is the company, rather than strictly as an individual. These metrics are things such as punctuality, emails, slacks, meeting attendance etc. they affect not only the output of the engineer in question, but those who rely on them to perform.

|  |  |
| --- | --- |
| **Type of Data – Interpersonal** | **Pros/Cons** |
| E-mails (Speed of reply, volume sent) | + Shows communication (or lack thereof) of the engineer with peers.  + Speed of reply can be a used to measure efficiency, is their slow responding slowing down their team.  - Potential ethical/data protection issues  - Not the only method of communication between team members. |
| Meetings Attended | + Great measure of engineer’s engagement in the company/team. This is vital to a productive team environment. |
| Slack/Internal Messages etc. | + Similar to emails shows good work ethic and provides the closest thing to measurable communication between team members  - Again, potential ethical issues.  - Communication is not guaranteed to be useful and in the pursuit of improving team productivity |

A final note on choosing what data to use - it is highly product dependent. Certain data will be more closely related to engineer productivity on different tasks, be it front end GUI development, database management software building, rapid prototyping of a new feature or concept, there is not (at least at time writing) a definitive set of metrics which, when measured during any project, will always solidly provide a measure for the workrate of an engineer.

The second question in terms of measuring software engineering is the how and where. Does one analyse the developer from their phones, like apple do with their inbuilt health and fitness tracker on iPhones, and send off the results to some manager’s spreadsheet? Farming the data out to a separate company is also an option. Companies such as gitprime, have built their entire business model around providing incredibly useful insights from developer data provided by companies, although availing of this service may seem counter intuitive as a money saving exercise, the results show that this analysis saves the company more than it costs them for this service, hence why these companies are popping up. Time is money in the software engineering world, and any tool that can ensure that your engineers are making the most of their time on your clock is well worth a look at.

There are different types of analytics on offer, many – such as the aforementioned gitprime – look at the git contributions of users, while others take data directly from the user’s development environments to calculate their work-rate. This raises the question of data protection and privacy, which will be explored more later on.

Other services such as codacy can track the change in code duplication, style and quality throughout sprints. Allowing for easy tracking of teams and assessment of their standard over time. These are not only a great way to ensure code quality is maintained throughout a project, but also to tell what effect a certain engineer has on a team when they join or leave.

The ethics of measuring software engineers is a bit of a grey area. On the one hand companies are paying potentially millions of dollars per year to teams of software developers, so one may say it’s fair that they would like to know that every single cent is being spent on quality engineers providing quality work. Alternatively, the company’s engineers have a right to privacy and may feel that the methods employed by their superiors to measure them paint them in an unfair light and/or put them in a position of unnecessary pressure at all times. Your company storing all of your emails or internal instant messages or even memos may leave you with a bit of an uneasy feeling, however they will say that once you send said email it is their property and there is not much an engineer can do to avoid this.

Another question entirely from the company you work for measuring your communications and coding habits is the tracking of seemingly unrelated metrics. Discussions on whether or not companies should be able to track an engineer’s location within their building or their eating habits for decisions on health insurance. Is this a step to far considering this kind of data does not directly impact the product that their employee was hired to provide?

The debate is interesting. On one side you have the fact that an engineer chooses to work at a company, knowing that certain metrics of themselves will be tracked and monitored, and the company may use it to make decisions about their future. Alternatively, the fact that a person would rather be viewed as just that, rather than a long set of statistics or a plot on a scatter graph. What if this information becomes public and it affects future career prospects? The tracking of these metrics is in my opinion necessary, and most everything non-development related that you need to learn about an engineer can be learned over a cup of coffee and a chat with a manager or co-worker. The excessive mining and storage of employee data is, in my opinion, exploitative and unnecessary.

All in all, the world of measuring software development productivity is even younger than the software development space, which itself is in its infancy. We will continue to see new theories on the optimal measures as well as new products and companies arising, purporting to have cracked this puzzle. It is an exciting area, and one that I feel is lucrative if you can develop the correct product, I will reiterate, people are spending an awful lot of money on software engineers, they’ll pay well to ensure that money isn’t wasted. In my own personal opinion, I believe that an algorithm will never be able to discern a great engineer from an average one as well as a human and that there is no substitute for good man management and an excellent communication model in any company, although I’m sure I’ll live to see the day where I slave away behind a keyboard at the mercy of the newest employee scrutinizing, machine learning project manager.