MEASURING SOFTWARE ENGINEERING

CSU33012-Software Engineering

Adam Ogórek
18335914

There are multiple ways people try to measure the productiveness, engagement happiness of software engineers. These range from simple to extremely complicated, but the fact that after trying out so many different methods, there still is no agreed upon standard, may hint that there is no single way to measure that has all the desirable properties.

Why do people need measurements in the first place? Having a simple way to know how fast their employees can do a certain job, and what are the particular skills of each employee can be a tremendous help for employers planning the budget and deadlines for projects and managing teams. Employers who are not familiar with software engineering themselves might require measurements as a way to make sure that none of the developers are slacking, and in order to find exceptional employees who deserve to be rewarded or promoted. Companies that are concerned about their employee's wellbeing, will likely want to try and measure whether their developers are happy and excited to work. Not only are happy employees more productive and less mistake-prone, an unhappy employee is more likely to switch jobs, and too many employees leaving can be hugely detrimental to the overall company productivity.

The biggest problem is that there is a lot of data that can be measured while a product is being developed but picking a bad measurement to rely on can be more damaging than helpful. One of the simplest ways to measure how much work an engineer is doing, is checking how many lines of code they have written. While at first glance this makes a lot of sense, after all writing code is a huge part of their job, but this can be easily gamed. An engineer can just keep writing useless code, trying to get the longest solution to a problem, instead of actually spending time working on new tasks. This discourages writing concise, easy to understand solution, which makes the overall product much harder to maintain as well. Not to

mention the fact that it's hard to decide what a line of code is, exactly. A comment is not code, but it's still important to make the code easier to understand and maintain, but if comments count, than it's even easier to exploit the system by writing multiple single line comments, which again ruins the clarity of the code.

Another simple one is measuring how much time an engineer spends sitting at their desk and working on a problem. Again, this is can turn into a problem if it's the only thing that is checked, as a person finishing 5 tasks in a day would be seen as just as good as a person who spent the whole day thinking about a single task without even finishing it. Using only the time spent working on something as a measurement of productivity could decrease the motivation a developer feels, which is definitely something to avoid.

Similarly measuring commits by either their number or their size would be very easy to game. If the number of commits is counted, simply submit a lot of small, insignificant commit. This might not be as detrimental as some other solutions, but it's still not an improvement. Measuring the average size of a commit is similar to measuring the number of lines of code, also introducing the problem that engineers would commit rarely in order to maximize the size, which would make team work much more difficult.

The last simple measurement I want to discuss is measuring the number of tasks completed. This works better on its own than the previous ideas, since the only thing checked are tasks that are needed to complete an iteration. There are still some problems, mainly because of the fact that some tasks are inherently more complicated and time-consuming than others. This results in engineers competing for the easiest tasks, and people doing the harder, perhaps even more difficult ones, are being punished in a way, by getting less recognition. This solution shows some promise, but there's still much more that's needed to evaluate the productivity properly and fairly.

A natural improvement to the problem that tasks are not equal to each other is assigning tasks weights, with harder tasks being worth more than easier. There is also the added benefit of being able to assign higher values to higher priority tasks. This solution is usually called “Story Points” and is wildly used nowadays. The biggest problem with it is finding a way to assign values such that they reflect the differences in difficulty accurately. The difficulty of a task also depends on the team it’s assigned to, which means that story points are not a great tool to compare members of different teams.

Line impact is an improvement on measuring lines of code, it uses a tool that analyzes the code and gives it impact value depending on hum much it impacts the end product. This involves assigning higher values to code in languages that are more concise, since usually the same code can be expressed in less lines, different multipliers for separate categories, like code, tests or documentation, lowering the value of code that is copied multiple times, compared to brand new code, and more. This means that it cannot be easily implemented and requires a highly specialized tool to work with.

These measurements help with tracking the progress of a project, but there’s also a question about the quality of code. Test coverage is a simple way track the quality, but like with a lot of measurements before, it needs to be combined with a metric tracking progress, or it can be easily exploited by writing less code and just adding tests for this small piece of code.

Tracking how many bugs are caught before releasing the code versus the number of bugs that slipped through is another good way to measure how good the testing is. Just like before it needs another metric to make sure the developer is actually writing code, otherwise it’s extremely easy to achieve 0% bug rate by writing no code at all.

Perhaps much less intuitive attribute to track is the happiness and excitement of engineers. As multiple studies have shown, happy

workers are more productive, less likely to make mistakes, and are more creative. There were also some effects in reverse, developers who disliked what they were working on, or felt that they are not doing as well as they would like to, were generally less happy, resulting in a loop. Obviously measuring happiness is a new challenge, requiring innovative solutions. Some studies used surveys in order to measure that but filling long surveys everyday would be much too tedious for employees, and shorter ones wouldn't give a lot of details. What was also found was that engineers disliked filling in the surveys because it disrupted their workflow. One company managed to have some success measuring the happiness by making their engineers include a grade of how they felt about the code on a scale of 0 – the code is terrible, to 5 – the code is amazing. While not very detailed, it helped to find out what's the general mood among the team.

Another popular approach are wearable devices that gather data while the engineers are working, often in the form of "badges" worn on one's neck. One approach is tracking the movement of a person, as it was found that happy people are generally more active and tend to have longer periods of sustained movement. Data about the movement patterns can be fed into a machine learning algorithm in order to measure happiness, and since the badges can also track other people a person interacts with, this can be used to track how a person influences others. Generally happy people can improve the mood of a whole group by interacting with them, boosting the overall productivity.

Another approach using the badges is measuring the audio. Even without processing speech, measuring and processing patterns in which people speak can provide a lot of data, not only about how they're feeling, but also about how people influence others. This can also help measure the true productiveness of engineers, since not all of their work involves actually writing code. Helping other engineers

write or debug their code can be a huge part of an engineer's day, and if you don't account for that with methods like story points or line impact, an engineer could be "punished" for helping others. Since most products are being developed by teams, discouraging teamwork is definitely something to be avoided.

Badges like these could be seen as questionable ethically, though. Being constantly listened to, or just tracked, even if just for the purpose of measuring their happiness, could be extremely unnerving to people. While the audio badges to not require speech recognition to work, they likely could be made to record conversations, and if data gathered this way was leaked for some reason, there could be unseen consequences. These badges are not inherently unethical, but there is a need for special consideration, and likely regulation of how they are used.

One problem that affects all of these measurement methods is that they attempt to measure individuals, when it's often much better to measure teams. Measuring individual work on a team project may not be fully representative of that engineers' real skills, since team members influence each other. Influences like these, cannot be fully understood using just the data from code. Perhaps the best solution to this is using feedback from other team members. While it's not as readily available as feedback from automatic measurement systems, people can still grade other's skills better than most, if not all automatic systems. This is especially true for experienced project managers, as it's a big part of their job to observe and help develop people's skills. Unfortunately, this is not helpful to the project manager, but can provide invaluable data to the company directors. The biggest weakness of this approach is that it's difficult to use this to compare employees who worked with different teams, it's not a universal measure.

There are countless other ways to measure that were not mentioned here, and none of them are perfect, and might never be.

Nevertheless, with careful consideration, a combination of these can provide valuable insight into the skills and productivity of software engineers. Coupled with the fast-growing machine learning technology, in the near future there might be tools that are capable not only of measuring, but also suggesting ways to improve the skills and impact of each individual. This however might come at the price of privacy and the ethical concerns coming with this have to be considered.