Measuring Software Engineering

Name: Leona Wolff

Student ID: 17331132

Software engineering is focused heavily on data. We measure and analyse things and use the resulting data to make some incredible products and software. It is ironic therefore that there are so few established methods of measuring our own productivity. We still haven't figured out a one-size-fits-all method for tracking progress in software engineering. Every IT company, small and large, as well as the engineers who work there, can benefit from measuring performance and productivity. However, there is a very fine line between measuring productivity in a way that is helpful, and in a way that is detrimental.

Before I started studying Computer Science, I worked full time as a Content Analyst on the German Team at an outsourcing company called Majorel. I worked on a Google Ads project. My job was to review ads, check if they adhered to Google's policy, and if they did, add extensions that linked to useful parts of the advertised website.

The company used a huge amount of KPIs to monitor our work. They had your standard productivity: number of tasks completed in a day, utilisation: total time spent on tasks in a day, AHT: Average Handling Time per task, Quality, etc. On one hand, I found it useful to know exactly how many tasks I needed to aim for in a day, how long we were supposed to be spending on each task and so on. However, I realised quickly that the way they were using the metrics was essentially a scam. It was impossible to be exactly on target for everything and maintain your quality. The company stressed that quality was "the most important metric" and management pushed the notion that nothing else mattered provided your quality didn't fall below 99%. Quality mistakes were a big deal. If you got even one spelling mistake, for example, you had to attend a meeting with HR, and if you got 3 you could lose your job. However, if you were careful and followed the training properly for each task, it was impossible to meet the AHT targets, even if your productivity was higher than 100%. Our team leads knew this and would privately tell us not to spend as much time error checking. We would still be in trouble if we had perfect quality but an issue in one of

the other metrics, so this meant that (at the urging of management to work faster) most people didn't especially care about quality, and focused much more on quantity.

Every once in a while you'd find a person who would get 200% productivity every day – so they were essentially doing enough work for two people – but when you worked that fast, it meant your utilisation would be very low, and they would *still* get given out to in 1-to-1 meetings for not working hard enough. This meant that any of us who had been working there for long enough to figure out the flaw in the system and hadn't left (the turnover rate was very high), had to get smart about the metrics. We had to do all sorts of tricks like leaving certain tasks open for longer to get our utilisation up, rushing through others, skipping tasks we knew would take too long, and so on. These tricks that we employed weren't unknown to team leads either – when I first started and was struggling to meet my targets, my team lead regularly told me that these were the things I had to do to get my numbers looking good. There were a few people who were so disinterested and fed up that they would mark perfectly acceptable (but complicated to handle) ads as not following the guidelines so they wouldn't have to do them to keep their metrics looking good. If the Quality team caught this, it was only a minor error – as opposed to any of the errors you could get for actually *doing* the task, which were called Quality Errors, and these could have serious ramifications. So essentially, the managers had come up with a metric system that actively discouraged their employees from doing their jobs properly. Had we not been subjected to so many different methods of "measuring our productivity" and having to focus so much on keeping our numbers looking good, we would have had much more time to focus on doing our jobs properly.

Then again, as I mentioned before, I did actually like knowing exactly how many tasks I had to do in a day. It motivated me to know how much I had to do, and it gave me a push if I was working too slowly. The issue only came from the many conflicting methods they used to observe our productivity. In my opinion, there is definitely a happy medium when it comes to monitoring productivity. I was totally fine with all the metrics they used. It makes sense to measure the number of tasks completed in a day. It also makes sense to keep track of how much time you spend on tasks – there were some people who just didn't care and would spend all day watching Netflix while they were supposed to be working if they could get away with it. I also thought it was wise to keep track of the AHT for different tasks, as it

could be a useful indicator to see if you were struggling with a particular type of task, or see which areas you could improve or get some help with. Quality too was a very important thing to keep an eye on. The thing that created a problem was that all these metrics were given almost exactly the same level of importance and were needed to be paid attention to, as it was this that stressed everybody out, got in the way of our progress and created a toxic work environment when it came to relations between the analysts and management.

The problem was that the metrics in and of themselves were not bad, it was the way they were used. Had their official metrics just been based on productivity and quality, and the other metrics been used as a tool to identify problems for people who weren't meeting their productivity targets, I think people would have been able to get more done, been less stressed, and as a result, stayed at the job longer.

Although my experience with performance tracking metrics is not quite the same as in Software Engineering, given that these metrics would not be applicable, it's relevant for two reasons. I wanted to illustrate how having poor metrics or just too many of them achieves the opposite of the intended goal: they cause stress, low motivation due to employees being unhappy in their jobs, and take focus away from the job itself, forcing employees to waste time trying to prove how hard they're working, rather than just doing it. It also helps illustrate why Software Engineering companies have such a hard time implementing performance monitoring metrics. My old company, though they would adamantly deny it, cared little for its employee's wellbeing, happiness or job satisfaction. They were constantly implementing new rules that would make employees unhappy but would supposedly boost productivity, such as a clean room policy where you could have no personal belongings with you in the office – not even a cup of coffee. This indicates one thing: they don't really care about their employees. Why? Although it can be tricky to replace German speakers, there are still plenty of them to go around. College degrees weren't necessary (I started working at this company having just done my Leaving Cert) and salaries were quite low. Software engineers, however, are a much more valuable resource, which can be seen in companies like Google and Facebook that provide many incentives to keep their employees happy. This means that it's not only a matter of finding metrics that work for measuring productivity in software engineering, the metrics also need to be non-invasive, not interfere with the engineers' productivity and not aggravate them to the point of leaving the company.

All the speculation on the right way to implement performance monitoring metrics is irrelevant, however, if we don't have any metrics to measure performance with. So, what are our options? Since the beginning of software engineering, people have been trying to come up with ways of measuring progress. In software engineering, there are many things that *can* be measured, but many of them won't help us. Unfortunately, there is no one metric that can be used on its own.

The most obvious 'solutions' are perhaps lines of code and number of commits.

Counting lines of code is, for obvious reasons, a terrible way of measuring productivity in software engineering, and yet it's been around for decades. Firstly, different languages and frameworks have very different syntaxes that can result in discrepancies where one language could be used to solve a problem in 10 lines and another language might take 50 lines to achieve the same. Aside from this, we can all agree that in most cases, fewer lines of code are far better than having longer code that does the same thing. And how do we account for refactoring code in a system like this? What happens to your productivity score when you start removing lines of code?

So, using lines of code as a metric to measure productivity doesn't really work. It gives some indication that work is happening, but very little about that work. A piece of code could be completely broken or bloated and by this logic, the engineer who wrote it would appear to be working harder than his colleague who wrote a piece of code that is functional, concise and streamlined. Rewarding people for how many lines of code they've written encourages bad practices that are contrary to successful software engineering. Despite these issues, this system of measurement is still commonly used in some companies.

Another approach is counting commits or pull requests – this has quite a few things in common with the 'lines of code' approach. It too gives some indication as to whether work is indeed happening, but aside from that, does it really tell us anything? Similarly, to how in the last method one could write excessively long code just to boost the productivity score, with this system of measurement, developers can easily exaggerate how much they are doing by committing every time they write a new line of code. This would slow them down massively and be of no use to anyone, all the while making it look as though they are the 'Most Valuable Player' of the team.

An alternative approach that is used is number of bugs closed. This approach has similar flaws. Bugs come in different shapes and sizes, and as a result, solutions to different bugs vary greatly in terms of complexity. Bug fixes themselves can introduce more bugs, and if bug fixes are how your productivity is measured, is this not the best thing you could do for your productivity? Provided you can fix the new bugs, of course. And if the code was bug-free originally, does that mean the engineers are all completely unproductive? This is another example of a metric that can incentivise bad practice.

Another approach I will look at is Agile Velocity, one of the most popular software development metrics in the world. It consists of a simple calculation that measures the number of story points that have been completed in a certain timeframe.

Story Points are a unit used to estimate the work required to complete a project. The unit of time used can be measured in different ways, for example by sprints or weeks. Teams would usually track the number of story points in a sprint, and use this figure to predict how many story points should be completed in other sprints. This can be applied to the entire project to help estimate how many sprints it will take to complete the project, and keep track of how things are going progress-wise. It is important to note that story points are subjective; each team or organisation will usually have their own interpretation of what a story point consists of.

Seeing as this method is based heavily on estimation and guessing, it should only really be used for planning. Unfortunately, this is not always the case. When used to compare different teams to one another, we encounter a host of issues. Seeing as the story points are subjective, it is easy for teams to manipulate the weight of the story points in a way that sheds them in a better light if they feel that this metric is being used to evaluate them. If one team does this, it can lead to two outcomes: other teams are unfairly painted as unproductive, and/or the other teams will likely figure out what's going on, leading to broken trust and frustration. Another side effect is that by inflating story point value, the metric loses much of its value in terms of its usefulness.

Another issue is that in large scale projects, velocity does not work. It is useful for small projects, as an estimation of how long aspects of the project will take, but when used to predict the delivery date of an entire product it is highly unreliable.

There are many more approaches like this, all of them suffering from similar issues. Most of the simple metrics are inaccurate, tell us little, and encourage bad practise. So, this means we need to find a solution that takes a few different factors into account. In order to do this, third party tools can be used. I will now talk about some of these tools. One of them, Humanyze, is not specific to software engineering and can be used in all kinds of teams to analyse how they are working. The other two are Pluralsight Flow (formerly Git Prime) and Code Climate.

Humanyze

Humanyze is a people analytics software provider. It helps managers measure how work gets done and resolve issues that reduce efficiency. Employees wear a badge that gathers data on the way employees interact with one another via a microphone that analyses conversations, Bluetooth sensors that track location and accelerometers that measure activity. Approximately 4GB of data is collected from each employee per day, and is sent back to Humanyze, where it is fed into its software platform, Humanyze Elements. The resulting data tells the manager who an employee talks to, the frequency, duration and tone of face-to-face interactions, and can also be used to make determinations on diversity, inclusion, engagement teamwork and policy compliance, among other things.

Pluralsight Flow

Flow is an organisational tool that provides a dashboard containing a variety of Git data regarding a team's productivity. Code metrics provided include number of 'coding days' (any day in which a team member pushes a commit to a project), number of 'commits per coding day', 'churn' (code that is rewritten or deleted shortly after being written), 'efficiency' (a measure of the amount of 'productive code' submitted – i.e. code that has not been rewritten and resubmitted), 'impact' (how much the edits made by team members affect the codebase), 'new work' (code that isn't replacing or updating existing code), 'risk' (how much effort is put into resolving a bug), 'raw throughput' (all code contributions made by a team member), 'productive throughput' (code that remains unchanged in the repository for

over three weeks) and ‘tt100 Productive’ (the average amount of time taken for an engineer to contribute 100 lines of code). The review metrics include ‘reaction time’ (time taken for a reviewer to respond to a comment or review a pull request), ‘involvement’ (an overview of how many pull requests a reviewer participated in), ‘influence’ (how often a reviewer’s comments result in changes being made) and ‘review coverage’ (how much code is reviewed per pull request). Finally, Flow also includes Team Collaboration metrics, which will give insight into how well the team is working together, by measuring ‘time to resolve’ (average time taken to close a pull request), ‘time to first comment’ (time taken for a review comment to be added after a pull request is created), ‘follow-on commits’ (number of commits added after a pull request is opened) and ‘PR Activity Level’ (used to identify pull requests that have solicited a lot of attention from the team).

All of these metrics can be used to identify problem areas in teams. They give managers insight into their teams and how things are going. Theoretically, this tool prevents engineers from manipulating the system by pushing lots of small commits or resubmitting the same section repeatedly, as this would be shown in the tt100 Productive metric.

Code Climate

Code Climate is a well-developed tool, considered by many to be the best option for automated code review. It analyses code looking for duplication issues, code smells and many other factors to determine issues in code quality, and provides an objective review of code quality as well as information and tools that may be used to address any issues. Code Climate runs the code in a given repository and provides details on any problems it detects, such as high churn or too much complexity. It also gives a ‘grade’ from A to F to describe code quality.

Perhaps it is because I found comparatively little information on Code Climate compared to Pluralsight Flow, but if I were in the position where I had to choose which I wanted to use for my own team, Pluralsight Flow seems like the better candidate. It appears to do everything Code Climate does and more, the only thing I worry about is the abundance of metrics that it provides. Having never worked in the software engineering industry and having never used a tool like this, I cannot say I am wholly confident in a piece of software’s

ability to identify good quality code. In 'The 2019 State of Coding Review' by SmartBear, it was revealed that only 45% of respondents were satisfied with their current code review system. It seems to me, inexperienced though I am, that if a manager with little technical background were to use a tool like this as the be-all and end-all in terms of monitoring an individual engineer's productivity, issues may arise in many situations, for example if they are working on a difficult task, or if the employee is of a more meticulous nature. There is no guarantee that the tool will determine the code quality correctly in 100% of cases, and if that is paired with what appears to be low code output or a similar issue, I fear that it may cause problems for employees who are despite appearances, working very hard and not being recognised for it. Another issue is that, as I mentioned was the case in my old job, I have found indications that if these tools are used as a definitive way of measuring an employee's productivity, it can indeed cause engineers to waste time in an effort to produce favourable diagnostics of their contributions. Unfortunately, I personally cannot see any way to prevent this seeing as any method of measuring productivity would likely produce the same result. We can only hope to minimise this issue as much as possible, and having it be automated is likely the best course of action since were it to be done manually, it would be even more inefficient.

Despite the issues I have mentioned, a tool like this is likely the best option (at least for now) and if used in a constructive way, appears to be highly effective in identifying problem areas seeing as it aggregates data in a way that would be infeasible for a human being to do on a regular basis. Another bonus is that such tools can be used for self-evaluation. An employee can look at their scores on a dashboard every day and identify problems in their own code, rather than having a manager have to come and nag them over every little mistake, hopefully alleviating tension between team members and management. At the end of the day, I think the effectiveness of these tools comes largely down to the manager wielding them.

Ethics

In the field of performance monitoring tools and software, ethics cannot be ignored. For the most part, the ethical issues we must consider emerge from the gathering of data. All of the

tools I spoke about work by gathering data and using this data to draw conclusions about the employee and their work. In my opinion, data that is relevant to the job and only the job is fair game. Monitoring number of commits, code quality, etc is totally fair, and if it is done right, I am fully in favour of doing so. The code produced by the employee is after all the property of the company who hired them to write it, and I believe the company also has the right to know how that employee is getting on – if an employee just sits around all day, why should the company have to pay them for this? And is it not in everybody's best interest to have identify problem areas that cause progress blocks? That way, support can be provided to whoever the unfortunate employee dealing with the issue is.

The issue is when these tools have access to personal data. These tools all claim not to collect private data, but they do have access to it, in particular Humanyze.

Human beings are social creatures. As it so happens, most of our time awake is spent at our workplace (coronavirus aside). People make lifelong friends and often meet their partners at work. The thought that all of these interactions were being listened to and analysed would make me uncomfortable beyond belief. To be fair to the creators of Humanyze, they seem to have good intentions – their website is full of assurances that data privacy is of their utmost concern. However, I have a hard time believing that if this is indeed the case, that it would truly remain this way for long. Big tech companies are constantly having data leaks, for example Amazon who just last week had a data breach where customer information was exposed, or the Facebook-Cambridge Analytica scandal. The amount of information Amazon and Facebook (among many others) have on us is bad enough, and that's without having a badge that contains a microphone and monitors exactly who we talk to and what we're doing. This points towards a dystopian future where we are monitored 24/7 and all of our personal information is owned by large corporations.

Humanyze claims that in companies who avail of their services, all employees have the option to decline to participate in this system. I appreciate this measure, but are some issues with this.

Firstly, Humanyze are trying to sell their product, and managers in the company availing of the service will want as many of their employees to comply as possible. Does this mean employees are only hearing assurances that everything is fine, and that their data is made

anonymous? Are they being made aware of the risks? I suspect that at the very least, the risks are downplayed. Secondly, I believe that many employees may be too afraid to say no. Will it then look to their managers as though they are hiding something? Especially if what Humanyze CEO Ben Weber says is true when he claims that an average of 90% of employees at each company he provides his service to agree to participate. I also believe that in certain companies, for example my old company, were they to start using Humanyze's services, it would quickly become a requirement. Sure, you would have a "choice" – but I suspect the choice would really be either compliance with the system or leaving the job. And if such technology becomes widespread in big tech companies, what option will we really have if we want to keep our jobs?

I believe that this level of monitoring is too much. It breaks trust and creates a toxic environment, causing frustration and ultimately lower motivation and productivity.

I think we need to be careful that the quest for measuring productivity in software engineering does not go too far when it comes to collecting personal information, but hopefully, with enough transparency in the companies that make the software and by managers placing trust in their employees, we can strike the right balance.

Sources:

The elusive quest to measure developer productivity - GitHub Universe 2019: https://www.youtube.com/watch?v=cRJZldsHS3c

Software Developer Productivity: What we know and how to make it better: https://www.youtube.com/watch?v=NhP7R2NRL8E

The myth of developer productivity: https://nortal.com/blog/the-myth-of-developer-productivity/

Can You Really Measure Individual Developer Productivity? - Ask the EM: https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/

How to Measure and Improve Productivity of Your Software Development Team: https://www.thirdrocktechkno.com/blog/how-to-measure-and-improve-productivity-of-your-software-development-team/

What is Velocity in Agile? https://www.planview.com/resources/articles/lkdc-velocity-agile/

Why Agile Velocity Is The Most Dangerous Metric For Software Dev Teams: https://linearb.io/blog/why-agile-velocity-is-the-most-dangerous-metric-for-software-development-teams/

Development Leaders Reveal the Best Metrics for Measuring Software Development Productivity: https://stackify.com/measuring-software-development-productivity/

Measuring productivity of software development teams: https://www.researchgate.net/publication/237050541_Measuring_productivity_of_software_development_teams

Measuring Productivity Of Your Software Development Team With Agile Metrics: https://dzone.com/articles/measuring-productivity-of-your-software-developmen

How to Measure Developers Productivity: https://www.devteam.space/blog/how-to-measure-developer-productivity/

Humanyze Provides Powerful Insights into Organizational Design and Workplace Behavior Through Data Analytics: https://www.cardrates.com/news/humanyze-analytics-provide-workplace-insights/

Humanyze-ing the workplace: gathering and analyzing unique data on employee interaction to improve productivity: https://digital.hbs.edu/platform-digit/submission/humanyze-ing-the-workplace-gathering-and-analyzing-unique-data-on-employee-interaction-to-improve-productivity/

How tracking staff boosts revenue – and why we should worry: https://www.irishtimes.com/business/work/how-tracking-staff-boosts-revenue-and-why-we-should-worry-1.3305888

What is Flow exactly?: https://help.pluralsight.com/help/what-is-flow-exactly

Pluralsight Flow Review – Is It Worth It: https://iqunlock.com/pluralsight-flow-review-is-it-worth-it/

Code Climate: http://dandemeyere.com/blog/code-climate

The 2019 State of Code Review: https://static1.smartbear.co/smartbearbrand/media/pdf/the-2019-state-of-code-review.pdf