MEASURING SOFTWARE ENGINEERING REPORT

Student ID: 19334721
James Deasley

CONTENTS

INTRODUCTION

Most disciplines are measured in some way in today's world. From student's being tested and examined on their learning, and journalists being tracked on how many articles they produce, all the way up to the highest levels of academia in which academics are measured on not only how much research they are publishing but also on peer review of their work. However, some disciplines are much easier to measure and track in a quantifiable way than others, and one of the most difficult historically has been, and continues to be, software engineering.

In this report I will be reviewing how we currently measure software engineering, including metrics used and how we pick these metrics, platforms which provide analytics based on such measurements, and approaches to providing these analytics and decisions taken based on them. Finally, I will discuss the ethical and moral issues associated with measurement of software engineering, and even touch slightly on legal issues.

The goal of this report is to develop and opinion on whether or not it is possible to measure software engineering, and, if so, how and why we should do it, or if we should even do it at all.

SOFTWARE ENGINEERING PROCESS

1. MEASURABLE DATA

The first thing to be said about measuring software engineering is that the question is not whether or not is it measurable. It is measurable. We do have metrics for measuring software engineering. The real question is whether or not the measurement we can do can actually be useful. For example, the classic example of a measurement of software engineering is lines of code written. Objectively this is a metric which measures software engineering. However, you don't need me to tell you that it is not a useful metric. This particular metric is flawed in many ways, one of the biggest of which is that, in software engineering, often the goal is to have as few lines of code as possible, making our software more efficient and succinct, which is of course the opposite of what this metric encourages. Another flaw of the lines of code metric, and one which is shared by many other metrics, is that it can easily be gamed. It is not hard for a software engineer to spread their code across more lines than they need. Metrics which can be gamed like this tend not only to be inaccurate measures of an engineer's productivity, but they also push engineers to sacrifice the quality of their work for managerial approval upon seeing the data which comes out of such a metric. This is, however, a very coarse example of a metric, particularly in today's world. With all of the information we have access to with today's technology, and undoubtedly more such technology to come, the question is do we have, or will we have, some metric with which to measure software engineering which is useful and accurate.

Before determining a useful metric, an organisation first must ask why they wish to measure software engineering productivity. Are we measuring simply to reward strong performers, or to track progress over time? Or are we striving to identify the most productive development processes and see if we can implement and improve on these processes to remain competitive? At the end of the day, any organisations ultimate goal will be to increase revenue, but each of these goals will have different metrics which are better suited to their advancement.

In order to measure software engineering in a useful and accurate manner, we first need a suitable metric. A good metric will have a few basic characteristics. First, the metric reflects the sort of productivity that the organisation wants, as described above, which will be defined by the reason why an organisation wants to measure software engineers in the first place. For example, a metric which correlates closely with revenue. Next, the metric must include all output, big and small. This includes non-engineering work, such as testing, leadership, and other such impactful contributions. The metric must also be resistant to gaming. As we saw with the "lines of code" example above, gaming often pushes engineers to go against what would actually be productive. A key attribute of a good metric is objectivity and consistency. It must be some quantitative value which is verifiable and repeatable. We can't have engineers measured by their boss's subjective sense of what they're like. We should also be

able to compare data from our metric across different teams and projects. Another feature is that our metric should be independent of the programming language used. This is another perfect example of a flaw in the “lines of code” method. Some languages simply take more lines of code than others to achieve the same functionality, because all languages are simply structured differently.

These are the basics of how any good metric should work, and yet even these are difficult to achieve. And even if we do achieve these attributes, it can be difficult to discern who should be given credit for every contribution, as some contributions are inherently difficult to tie to just one individual. For example, measures such as uptime and bug fixes, which are likely the result of a team effort. There will also be cases where some objectively good developers don’t do well in the team environment, making the team less productive overall, and cases where some moderately good developers have great team synergy and are thus more productive as a whole. For these reasons, I think it is more correct to measure the productivity of a team, rather than that of an individual. In a similar boat, with the difficulty of assigning credit, metrics shouldn’t penalise engineers for attempting large, ambitious solutions, as a measure like “number of pull requests” would. To handle cases like this, it seems more fair to measure the development as a process, rather than the output. We can do this through metrics such as “code-review turnaround time”, however I have also read about this particular metric causing issues when attempted before, as it discourages engineers from making review requests on Fridays because the turnaround time will be longer over the weekend.

While I do still believe that software engineering is indeed measurable, it is, as you can see, a minefield to do in a useful, accurate, and fair way which doesn’t discourage real productivity. For now, the best solution seems to be that organisations should employ multiple different metrics, as opposed to looking for just one. They all have their flaws, but a few metrics put together can paint a bigger more accurate picture of what is really going on.

2. Development Analysis Platforms

Software development is an intensely data rich activity. At every stage of development of a project, nearly everything can be measured automatically, economically, and in great detail, from code commits and pull requests to testing and bug reports. In this way and many others, it lends itself well to analytics. One of its biggest weaknesses, however, is the difficulty faced by software project managers. Despite the overwhelming amount of data, it can be difficult to make use of any of it as the context from which it is taken can be very hard to grasp, leaving managers to their own intuition to make decisions. While this can of course work out well sometimes, decisions based on intuition alone can often be suboptimal and sometimes even destructive to the development process. This lack of useful information combined with the potential risk behind every decision made means there is a lot of pressure on managers in the software development environment. Managers are always going to turn

to the data for help, so it only makes sense that they should be able to see useful insights made by analytics tools from that data to help them understand and interpret it all.

Fortunately, software development is very well suited to analytics. On top of the sheer volume of data it yields, it is also timing-dependent due to its need to meet strict schedules, which can be helped along greatly by analytics tools which allow leadership to look both to past performance and to what is expected to come in the future. Analytics also help to enable consistent decision-making, which is a key part of the control of a software project. The often distributed nature of the software development process means it can be difficult for leadership to make decisions as it can be difficult to grasp the collective state of projects. This process can be made easier for the decision-makers through analytics which can help them to understand what is going on across projects through deeper insights taken from the data of all projects.

For software project managers, analytics can be of aid in many ways. Analytics can provide managers with the tools to understand the dynamics of a complex project and can help evaluate the effectiveness of changes in the development process. Managers can see where resources and talent need to be allocated and where these things are under-utilized, allowing for improved efficiency. Analytics can help managers to detect and forecast future trends in data, and also to evaluate past decisions, allowing them to make better decisions on these forecasts based on what has worked well in the past. This would also help managers be more objective and consistent in their decision-making.

It is for these reasons that so many software development analytics tools have emerged in recent years, such as “Pluralsight Flow”, “LinearB”, “Waydev”, and many more. These analytics tools, trusted by many large enterprises, with “Pluralsight Flow” seemingly becoming an industry standard, provide precisely the kind of data and insight that managers need. Managers can see how much time is spent refactoring old code compared to writing new code and be advised on project bottlenecks and how to remove them in the coding stage. In the review stage, managers can track how well their team collaborates in terms of productivity and positivity when reviewing pull requests. They can also see insights on interactions within their team more generally, such as responses to pull requests and commit, and internal feedback and mentorship.

Ultimately, managers are always going to turn to the data because they need help with their decisions and because there is so much of it to go around in software engineering. I believe it is better that they have insight along with this data, which is provided by many analysis tools such as those mentioned above. The data is only really effective if the manager understands why the data is how it is. These tools will help with that understanding.

3. Algorithmic Approaches

When considering algorithmic approaches to measuring software engineering, we can consider two approaches: the machine learning approach and the expert systems approach. The machine learning approach is the more basic “artificial intelligence”, simply searching for interesting data and comparable statistics which may highlight some sort of pattern and be useful in that way. This is the easier method to implement as it is much simpler and more raw. Essentially this approach needs a human who is an expert in the field on the end of it to interpret whatever the algorithm may highlight and use their own intuition to decide whether or not that information is actionable. It is difficult enough for a more complex algorithm to understand the subtleties in human behaviour, let alone a simpler algorithm like this. This does however leave the door open to human error, leading to more unpredictable, inconsistent and potentially damaging decision-making. For example, on a day where our human expert isn’t feeling well, they may not perform up to their usual standard, making decisions which they may not otherwise have made.

The expert systems approach on the other hand is much more sophisticated, making it much more difficult to implement. The idea behind an expert system is that it can emulate the decision-making ability of a human who is an expert in the field. However, because it is artificial, it is not open to human error, resulting in more consistent decision-making. The question is whether or not hard consistency is a good thing. Often inconsistencies in human-made decisions are results of the one making the decision interpreting some subtlety in human behaviour which warrants an exception to what might otherwise be a consistent, rule-based decision. While an expert system does strive to emulate the decision-making process of a real human, it cannot be stressed enough just how subtle human behaviour is, making this very difficult to achieve in a way that is fair and empathetic to human issues.

So it is a question of a hard vs a soft algorithmic approach. A human expert possesses the ability to interpret all of the subtlety of human behaviour but can lack in consistency when compared to an expert system. Human knowledge is also bound to one channel of communication at a time. For example, a human can only take one phone call at a time. Compare this to an expert system which can apply its knowledge across multiple channels of communication at the same time, and the difference in efficiency is plain to see.

Human knowledge is also much more difficult to document. Although there is data acquired by an algorithm which feeds into the decision a human expert might make, it is ultimately up to the decision-making process of the human. This might not be an issue in some areas, but when the whole point of measuring software engineering in the first place is for an organisation to be able to document the performance of their engineers, they ought to be able to document the reasons why decisions are made based on the data taken from these engineers. An expert system makes this much simpler by making its decision based on clearly defined rules, allowing its decision to easily be documented.

Similarly, when considering the needs of a serious organisation, monetary matters must also be a concern. In the case of soft vs hard AI, soft AI will of course be cheaper to develop.

However, soft AI also brings with it a need to employ a human expert to interpret and act upon the data the algorithm believes to be significant, and a human with a high degree of expertise doesn't come cheap, and this is assuming that we only need one human to solve the problem. By comparison, an expert system, although more difficult and expensive to develop, will ultimately be much more cost-effective than employing a real person with expertise in the field. However, this cost-effectiveness can only last if we assume that the expert system is successful in its emulation of a real human expert. Perhaps if a human expert performs better than an expert system it is ultimately worth the extra cost of employing a human because they bring in more revenue through their better decision-making.

In the end, I think the question of soft AI vs hard AI has far too many variables to have just one answer. A soft AI will be cheaper than a hard AI which in turn will be cheaper than a real human to handle the soft AI. And we will likely need more people to maintain the expert system, which opens a whole other can of worms. I believe the decision on which approach is better ultimately comes down to the effectiveness of the human expert behind a soft AI vs the confidence we have in the effectiveness of the expert system, both of which are quite subjective depending on the situation. If we believe an expert system can accurately emulate the decision-making of a human expert while also being empathetic to real-world human issues and not making rash decisions when an engineer is experiencing such an issue, then it would appear to be the more consistent and cost-effective solution. However, until we can have full confidence in such an expert system, I believe the safer option is to stick to the soft AI approach of a data-supported human expert. The human element may not be as easily documented, and certainly isn't cheaper on the surface, and does leave the door open for inconsistencies, but at least it is, if you will, consistent in its inconsistencies, and we can understand it.

4. Ethical Analysis

In terms of ethics, measuring software engineering seems to be something of a minefield for issues, so I have divided this section up into three sections:

- Consequences on individuals
- Accuracy, and
- Societal issues.

4.1. Consequences on individuals

One of the biggest issues currently with measuring software engineering, and one of the reasons why people are so opposed to it as a practice, is the ethical issues it brings to the table. It begs the question of whether or not we can let a machine have control over a human. And if we do let a machine have control over a human, what consequences might this have on the life of the human in question? In the case of software engineering these ethical issues

are particularly apparent because we already have the issue of software engineering being a very difficult sort of work to measure, like discussed above. Assuming we let a machine have power over a software engineer by letting it measure the engineer's performance and judging the engineer based on these measurements, how powerful are we willing to let these measurements be in a workplace? For example, can we fire an engineer based solely on poor measurements? Or do we require that managers and whomever is looking at these measurements use the measurements only to help aid an engineer's performance, rather than remove poor performers to increase the productivity of a project? And this is assuming a manager even really looks at the data collected. In the case of harder expert systems like those discussed above, can the system itself make the decision that an engineer's performance is not up to scratch and that they need to be replaced? If it can come to such conclusions, do managers take it at its word assuming it is as effective as a human expert, as it was designed to be, or must they be cautious and use it merely as a suggestion?

I think this comes back to the question mentioned above of why an organisation would want to measure performance. If they wish to measure the performance of their engineers purely for rewarding high-performers, for example, then there isn't much to worry about. However, if you have some algorithm highlighting patterns in productivity which notices that one engineer is severely lacking, this is going to be hard to ignore, even if you aren't looking for it. In this use case, perhaps the management doesn't pay this slip too much heed, as they are simply looking to reward their top engineers to encourage productivity. But you can see how it can be a slippery slope when the data is presented to you. Of course that's not to say that managers should never punish poor performers. There comes a point where it is simply not sustainable to employ someone who doesn't yield enough return to justify their salary.

This becomes a much bigger problem if the reason an organisation wishes to measure the performance of their software engineers is to identify effective development processes and implement these processes elsewhere to improve efficiency and remain competitive. In a competitive environment there is no time to waste on poor performance. Look at any competitive sport, for example Formula 1 racing, and you can spot the similarities. In Formula 1, cars are kitted out with tools to measure the performance of not just the car but also of the driver. This data is presented in real-time to their team in the form of analytics, which the team uses to make decisions during the race to streamline their strategy. However, if they notice that their car is performing perfectly fine but the driver is performing poorly, you can bet that as soon as their contract has expired that driver will be gone. Otherwise, they are simply wasting precious time in which they could be racing competitively and scoring points for their team. Similarly, an organisation which aims to be competitive in their field cannot and will not waste time on what appear to be poor "drivers" behind the wheel of their development process. There is far too much revenue on the line to be lost or won.

4.2. ACCURACY

In a fast-paced competitive environment like the one discussed above, it becomes paramount that the analytics are absolutely accurate and a fair representation of what is going on. When the consequences of an engineer's measurements reflecting a poor performance is the difference between them having a job and being unemployed, we must be very, very careful about what we measure and the context in which we present these measurements to management.

Going back to the Formula 1 example, we can look at our car and driver as a parallel for a software development team, in that both members must be performing to a high standard or the entire operation is hindered greatly. As mentioned above, F1 teams have analytics for both the car and the driver to measure their performance. However, let's imagine that, hypothetically, an F1 team doesn't measure the state of their engine. Obviously in the real world this would be ridiculous, of course they are keeping a close eye on that which makes the car drive, but go with me on this because I think you will see some similarities here. Now imagine that instead of measuring the performance of their engine, which is inside the car, they are measuring only the performance of the tires, which they can see from outside the car. Our driver places badly in a number of races, but the team can see from their analytics that the tires are performing just fine and are in a good state. So the management say "Well clearly the car is performing perfectly as we expected", and blame their losses on the driver. The driver could lose their job over this getting pinned on them. Meanwhile, the engine simply does not have the power it needs to race competitively, but nobody is measuring this, so the car gets off scot-free while the driver takes the brunt of the blame. Blame which is supported by the data measured. I'm sure you can see the parallels here between two competitive scenes which are so reliant on multiple members of a team. When the stakes are high, we need to be measuring not only the right things, but also measuring these things accurately and in a way which fairly represents the performance of all team members. If we only measure one or two surface level things, even if these things are indeed important things to measure, we can quickly see terrible consequences which not only don't solve the problem but can make it worse, and then leave someone unemployed for no good reason to boot.

4.3. Societal issues

We must also consider, in a world in which software engineering is measured, and measured effectively and fairly, what consequences could this have on society as whole? Currently, software engineers are among the most powerful people in the world. Firstly because they are hard to track. Software engineering is a difficult discipline to quantify. But on top of the that, the world today runs on software. Anything you want to do, "there's an app for that". Anything you want to know, "Google it". Most of the tools and information that people today access regularly are controlled by software engineers who build these systems. The combination of these two things, the power software engineering brings and how difficult a discipline it is to quantify, makes for a person who can build something that many millions

if not billions of people might rely on, but who also is very difficult to hold accountable for any issues they cause, either by accident or through their own intentional wrongdoing. Now one might think then measuring software engineering must be a good thing. It is a goal that we must strive to achieve, and to achieve as soon as possible.

However, as I have alluded to time and time again, software engineering is not like other disciplines. Software engineering is by its very nature highly data intensive. You can track almost every single thing a software engineer does on their computer, and in a job that is based mostly in a computer, that's a lot of data. Eventually, however, even assuming that all that we measure is not only effective but is measured accurately and fairly, there comes a point where we must ask if a line has been crossed. Because almost every aspect of software engineering can be tracked, if we wanted, we could be measuring every second of a software engineer's day. It is unlikely that all of this would actually be useful data but let's assume that it is. Doesn't it cross a line between measuring and enhancing efficiency and productivity and infringing on a person's personal data and right to privacy?

Very few disciplines are tracked every second in real time as this simply is not very effective to do in many disciplines. For example, a primary school teacher cannot be tracked at every second of every day. Their performance is based on their students' performances in testing and exams, which usually occur at well-defined and spaced-out intervals of time. It would be ridiculous to send someone into a classroom to write down every single thing a teacher is doing. Timing every second they aren't speaking or writing on the board, and every time they have to go to the toilet. Even if this resulted in some effective data which improved a teacher's performance, it would be absolutely shocking to see someone getting tracked so closely as they try to work. No other profession would stand for being followed around all day as they go about their job. This is effectively where we could end up with measuring software engineering, it just doesn't look so intrusive because measurement is done though data which can be taken from their computer, rather than needing to have someone watch them.

Currently software engineering is at the cutting edge of society, so it seems fair to assume that analytics of software engineering would follow suit, pushing the boundaries between what is reasonable for an organisation to measure and what is too far. It may sound somewhat out there, but I believe that if software engineering were measured as intensely as it could be, it would only be a matter of time before other disciplines would follow suit, either from people in other fields who have seen how software engineers are measured or from software engineers who build the software to track workers in other disciplines. Similarly, I think if we develop tools which measure software engineering with a good degree of context, most of these tools can be applied to any person's computer, not just the computer of a software engineer. Now this again might seem a bit far-fetched, but considering how much we as people are already tracked by organisations like Facebook and Google, does it really seem that unlikely that someone would eventually find a way to implement this? And maybe even sooner rather than later. Software moves quickly. This would of course bring up a whole slew

of GDPR issues and even violations, making this not only a moral issue but a legal issue as well.

CONCLUSION

I think it is fair to say that software engineering can in fact be measured. As I said in the beginning it is not a question of whether or not it is possible, but a question of how we do it effectively, accurately, and fairly, as well as ethically and legally. As I have mentioned several times now, software engineering produces a large volume of data which could potentially be measured. Every second an engineer's computer is turned on it is producing a datapoint. And every second an engineer's computer is turned off is also a datapoint. We have powerful tools too analyse this data and present it back to us, or more specifically to project managers, in the form of detailed analytics. I think these tools are in a good place right now. They make use of lots of objective data and they present it in such a way that a manager can review it and make their own decisions. However, I think this is as far as we ought to go with measuring software engineering.

This is not to say that I think measurement of software engineering is bad. In fact, I think that it is a very good thing not only for organisations to improve their efficiency and productivity, but also ethically to hold engineers accountable for their work. However, I think that one of the only ways to move forward from where we are now is to essentially automate the management process, letting an algorithm decide if an employee is worth having around, or if they should be tossed out on the street. This may be a slightly extreme example, but I believe that this is ultimately where we are heading if we pursue heavily AI-based performance measurement. Although I know it would be naïve to think that big companies would stop pursuing profits in order to maintain "that human touch", I do believe there is an aspect of humanity that ought to be maintained in every workplace which is more important than any amount of efficiency. I have watched too many dystopian sci-fi movies to believe that a society controlled by AI is a harmless and good idea.

I do think that there is room for growth, of course. I think we could take our current system, with a human at the core making decisions, and run with it to great lengths. As long as we maintain that level of humanity, we can push our measurement tools to hell and back for all I care. As long as we have a human at the end, I think we can feel safe enough to try new things, measure and review lots of different datapoints, and push the envelope as much as we can, but all while knowing that, if it becomes too much, we can reign it in.

An AI doesn't care much for ethics, nor does it recognise when it goes beyond what is ethical unless we teach it to. But teaching ethics is not something we can do forever. Ethics are constantly changing. There are plenty of things that would have been considered perfectly acceptable 30 years ago which today are totally unethical and unacceptable. We don't even

teach ethics to humans really. Obviously, we try to push children in the right direction, we try to call people on their biases, but we don't just teach it, because it is too complex. If there's one thing that you learn from studying computer science, it is that computers are based on 1s and 0s. But not everything in the real world is a 1 or a 0, especially when it comes to ethics.

In short, I believe that measuring software engineering is very possible to do, to do accurately, to do fairly, to do ethically and legally. I believe we should continue to do it and should also continue trying to improve on how we do it. I believe AI can be a powerful tool to assist in measuring software engineering, but that the final decision should always be left up to a human. Finally, I believe that it is an area that ought to be closely monitored, as the impact of any serious change in a field as powerful as software engineering is in today's world can bring about major paradigm shifts in many fields which would follow suit and in society as a whole.

REFERENCES:

YOUTUBE VIDEOS:

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

"Lies, Damned Lies, and Metrics • Roy Osherove • GOTO 2019"
https://www.youtube.com/watch?v=goihWvyqRow

OTHER ARTICLES AND PAPERS:

"Analytics for Software Development" - Thomas Zimmerman
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/MSR-TR-2010-111.pdf

"The ecstasy and agony of measuring productivity for Software Engineering & Development" - Cathy Reisenwitz
https://www.getclockwise.com/blog/measure-productivity-development

"Expert Systems in Artificial Intelligence (AI)"
https://omnilegion.com/expert-systems-in-artificial-intelligence-ai/

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

"Software Measurement and Metrics" - pp_pankaj
https://www.geeksforgeeks.org/software-measurement-and-metrics/?ref=lbp