Report - Measuring Software Engineering

Author: Jack Joseph Gilbride
Student Number: 17340868
Dates Prepared: 14/09/2019 – 08/10/2019

Abstract

There are different aspects of the software engineering process that can be assessed and measured. One aspect is individual contribution; the productivity of an individual software engineer. Another is team contribution, which is the productivity of a software engineering team as a unit. The third aspect discussed in this report is the software engineering methodology; different routines and tools that improve metrics for the software engineering process. An individual may be concerned with these measurements to improve themselves as an engineer, while a company may be interested to ensure that they get the best practices from their employees to accomplish their business goals.

Each of these aspects may be assessed manually; that is, without some sort of automation or algorithm to analyze the metrics. The agile software engineering model allows for this as it is easy to manually track whether targets are met or not. Other metrics may also be tracked manually separate from an agile model.

For the purpose of scalability and automation, the process may also be tracked using software. Beyond the naïve lines of code (LOC) metric, many distinct metrics may be tracked such as participation in message boards. From an algorithmic approach, it is up to the business what metrics they consider the most important to track. One approach exists which assigns weights (importance) to various distinct metrics and sums them up in a "contribution function". There are many other algorithmic approaches but many follow the same idea. In addition to measuring this "qualitative" data, automated assessment may also be done on "qualitative" data, such as sentiment analysis on communications.

Beyond the ideas of manual vs automatic measuring and assessment, there exists the idea of only focusing on metrics which are linked to business goals. The argument states that for a business, measuring any metric of a software engineer is useless unless it contributes positively to the business outcome of the project. In other words, it is a false positive assessing code as anywhere near perfect by any metric if nobody wants to use it.

After examining the different assessment models, I feel as though there are plenty of issues with fully automating the measurement of the software engineering process. Some are technical, like the ability to collect enough data to make an informed decision, and some are ethical, such as the implications of hiring/firing someone based on an algorithm's decision. This does not mean there should be no measurement, or that the measurement should be manual, as automation brings plenty of benefits. It also does not mean that I agree with the "business success" measurement model, which I see plenty of issues in. I conclude that an ideal assessment model involves the automated collection of qualitative and quantitative data which is presented on a dashboard, allowing humans to make much more informed decisions much quicker while still exercising their judgement. In an agile model, agile metrics should also be displayed on this dashboard. Humans can then make their own judgements on the data. This would be a much more accurate and ethically sound method of measuring and assessing the software engineering process from my point of view.

Introduction

The purpose of this report is to examine the ways in which the software engineering process can be measured and assessed in terms of measurable data. It will look at the different methods of measuring and assessing software engineering under different headings. Each of these headings will consider the assessment method, the computational platforms available to perform it and the algorithmic approaches available. Following the main body of the report will be a discussion section which analyses the ethical concerns surrounding this kind of analytics.

Before analyzing the topic, it is important to define what is meant by measuring and assessing software engineering, and possible motivations for doing this.

Measuring and Assessing Engineering: The Motivations

One of the measurable and assessable aspects of the software engineering process is the individual performance of an engineer may be assessed. There are many motivations for this. On a personal level, an engineer may want a metric for how well they performed over a particular timeframe. They can review that score and compare it with a previous score from another timeframe. Based on this, they can adjust their methodology, tools and schedule to improve the metric. If the metric is accurate, this will lead them to become a better engineer. If the engineer works for a company, another motive to measure their engineering process will come from their superiors. With a method of measuring software engineers, they can be ranked and compared to a particular standard. If the metric shows that an engineer is up to or above a particular standard, they can be promoted, moved to a team that needs them most or given incentives not to leave the company. If the engineer is not up to standard, they can be told to improve their standard, given appropriate training, moved to a more suitable team or maybe even let go. If an engineer's performance is satisfactory, it can also help to ensure constant performance, so their productivity does not drop below the threshold. So measuring and assessing individual performance enables individuals to become better software engineers, and companies to get the best software engineering potential from their employees.

Individual performance is not the only area where software engineering can be assessed, however. Performance may also be assessed across a software engineering team. Again, the motivation behind this is to improve performance and ultimately increase whatever metric the assessment is based on, this time across the team. Comparing against previous measurements and to a particular standard allows the team to increase their performance as a unit. Based on this, similar reactions can be taken as after assessing an individual's performance, this time for the team. In addition to this, new team members can be brought in, existing team members can be put onto new teams or roles can be switched within the team itself. All of this helps to keep the best engineers working most efficiently and the software engineering process to be improved, in regards to whatever metric the company is using.

Aside from individual and team evaluation, there is a third aspect of software engineering that can be measured and assessed. Specific tools and methodologies can be measured to see how they impact the engineering process. For example, a project may be done more productively in one set of programming languages compared to another. A specific development environment may yield better results for this metric also. A third example is the communication platform used during the process; different platforms such as instant messengers, email and meetings may all yield different results. By not just assessing the engineers, but the tools used regardless who's using them, "productivity" can be further increased, i.e. the most productive tools can be enforced throughout a team or organization.

On a more fundamental level, for a business, the motivation to improve the performance across the software engineering process is to deliver better quality software. Better quality software will be easier for the business to sell; therefore, the business will make more money from a more productive software engineering process. This report will primarily focus on the individual assessment and measurement of software engineers, but will also touch on the measurement of software engineering teams and the tools used during the process.

METHOD 1: Manual Evaluation

The measurement and assessment of software engineering may be done manually. Certain metrics, or a combination of them, can be taken manually and assessed by real people. Whether the assessment is focused on individuals, teams or methodology, a lot of the metric headings will be the same. A difference among the three may be how the results are interpreted, i.e. what measures of each metric are desirable for an individual, team or methodology.

The manual measurement and assessment of the software engineering process may be desirable for a number of reasons. Firstly, it may be too costly to implement software which automates the assessment of engineering. A company or individual may not have the time, money or resources to invest in a more complicated method of assessment. A manual measure or assessment may therefore suffice for them, and they may be able to make appropriate adjustments to their software engineering process based on this.

Many of the ways in which software engineering can be measured through this approach relate to the agile development model.[1] One example is keeping track of burndowns, which measure how many development tasks are completed over time. Sprints are created with a set number of tasks, and the burndown shows whether tasks are completed to stay on schedule. This helps to show return-on-investment (ROI) and progress in smaller bursts. This data can be easily measured manually as it is mostly binary yes/no data. A sprint being

[1] https://stackify.com/measuring-software-development-productivity/

completed on time contributes positively to this metric, while not being completed on time does the opposite. An expansion of this assessment method would take note of how much ahead of/behind time the sprint was completed. This method of assessment enables quick feedback in an intuitive way of thinking, allowing individuals, teams and methodologies to improve.

There are other measures by which software engineering can be assessed manually. One simple metric that may be put into the manual evaluation is the time spent in meetings. An engineer or team's productivity score may be affected if they spend too much or too little time in meetings every week. More possible measurements are how long the work takes relative to original estimates, and how many new features are completed in a particular timeframe. These simple metrics can be measured using a number of manual methods. For example, the metrics may be entered manually into a system or even kept track of in a simple spreadsheet. Entering the data manually means that there is always someone there to keep track of it, and it is likely that they understand the meaning of all the data that is given back. If the information is correct, it is likely that it is reflective of the metric that the company or individual wants to use.

So there are multiple manual methods of measuring and assessing the software engineering process which may be the most desirable methods in some cases. However, in many cases the manual approach may be too costly in terms of people-hours, as at least one person is tasked with keeping track of the metrics. If a company or engineer is truly concerned with productivity, then it may be best to automate most, if not all, of the measurement of the software engineering process.

METHOD 2: Technological and Algorithmic Evaluation

As software engineering is inherently a technological process, most of the actions of a software engineer and their team can be tracked using technology. While manual measurement may be desired for its simplicity and intuitiveness, there are plenty of reasons why automating the evaluation may be more desirable for a company or individual.

The most naïve method with this approach would be simply counting the lines of code (LOC) submitted over a particular period. This has for long been the de-facto standard of measuring software engineering. It is an easy method of automatically assessing an individual or team. A simple tool can be written to count the lines submitted by a particular individual/team, or even update the respective LOC counts on commit. However, there are plenty of problems[2] with this metric. Firstly, lines of code are not even a true metric of what a program actually does; the same problem may be achieved in a number of different ways depending on functions, loops, conditions and various shorthands. Secondly, LOC does not provide any insight into the quality of code actually produced, just how many lines were written. Thirdly, following on from this, if a software engineer becomes aware that LOC is

[2] https://dzone.com/articles/lines-code-bad-metric-either

being used to assess them, it is very likely that they will abuse this metric and commit vast amounts of code without any regard for the quality. This is very likely if any measures mentioned in the introduction to improve the software engineering process; software engineers will notice that those who commit the most LOC are treated more favourably in comparison to others.

So there are multiple fundamental problems with the LOC metric. Because of the problems with the LOC metric, other measurements may be taken using the algorithmic approach. One example is the measurement of some sort of "contribution function".[3] A contribution function may take a naïve measure of LOC but also look at other metrics. Each metric would have a different weight, i.e. a different level of importance. The advantage of this approach is that the weights can be adjusted based on whatever metrics the individual or company consider the most important. Actions can have a positive or negative effect on the contribution function. Examples of positive contributions include committing fixes to code style, starting a new thread, closing a bug or updating a wiki page. Examples of negative contributions include introducing a bug to the code, committing with an empty commit comment, or closing a bug that is then reopened. This approach takes a much more general view of the software engineering process than just the lines of code submitted.

The implementation of the "contribution function" method of assessment is a plug-in to the Alitheia software evaluation tool. The Alitheia platform consists of a set of core services, such as accessors to project assets, continuous updating of monitored projects and relational data storage, and it is extensible through the use of plug-ins. The contribution metric implementations is a compound plug-in that calculates basic metrics such as lines of code and mailing list participation. It is designed to perform an in-depth analysis of thousands of projects on a per-repository basis and allows full automation of the assessment and measurement of the software engineering process. The different weights of different actions are recalculated at regular intervals based on statistics from use and feedback from users.

The "contribution function" method is one example of how much assessment in software engineering is proposed; take a collection of measurements, assign them relative importance and come back with a score. It can be taught of as a template model for automating the measurement of the software engineering process, especially as the weights of each metric are not fixed for the model. When humans come to reason about how machines can judge engineers, this is the most intuitive model. Machines work with numbers, and "measurement" tends to involve quantitative data, so any automated assessment would take numerical values and work with respect to those.

---

3 https://www.researchgate.net/publication/221215616_Measuring_Developer_Contribution_From_Software_Repository_Data

This comes from the fact that machines do not "think" in the way a human thinks.[4] As a machine does not have a consciousness, it does not have the same empathy that a manager or other team member would when judging a software engineer; a manager's decision may also take account of what's going on in the engineer's personal life, the pressure they are under at the moment or the contribution that they made to team morale. Because a machine cannot exercise this level of "judgement", it can be said that it can only assess an engineer based on quantitative, not qualitative data.

However, some disagree with this conclusion.[5] They say that this way of thinking is something called AI Fallacy; "the mistaken assumption that the only way to develop systems that perform tasks at the level of experts or higher is to replicate the thinking process of human specialists." A simple example that goes against the AI Fallacy is IBM's Deep Blue Machine, which beat reigning chess champion Garry Kasparov in 1997. The machine did not go about beating Kasparov by trying to mirror a human's way of playing chess, its processing power allowed it to outperform Kasparov and use its own algorithms to achieve these results.[6]

This argument goes further and says that because of the many examples which contradict the AI Fallacy, a machine does not need to feel human feelings to come to the results that humans would come to through feeling. This has implications in various sectors such as the legal and medical sectors but also applies to the qualitative measurement of the software engineering process. Given enough data, perhaps a machine could come to the same conclusion that a human would in every case, simply using processing power and regular pattern matching techniques.

This argument could apply to any amount of qualitative data. One example is the sentiment an individual contributes to the team. This is not a numeric measure in the natural human way of thinking. But tools already exist to measure sentiment analysis in emails and group communications.[7]

Building on this, it is not hard to imagine some sort of chatbot system which exists to collect qualitative data. Indeed, many already exist.[8] When the quantitative data shows that an engineer is not performing to standard, a chatbot could enquire with them what is wrong. It could then take this qualitative data (e.g. the engineer is stressed), take account of it in its quantitative research, and repeat. A fully automated system may also inform relevant employees (other developers, managers) of this, which would improve the software development process from a team perspective.

---

[4] https://business.linkedin.com/en-uk/marketing-solutions/blog/posts/content-marketing/2018/Can-a-machine-have-empathy

[5] https://www.youtube.com/watch?v=Dp5_1QPLps0

[6] https://www.nytimes.com/1997/05/18/nyregion/what-deep-blue-learned-in-chess-school.html

[7] https://becominghuman.ai/sentiment-analysis-and-employee-engagement-how-companies-can-leverage-ai-3aa7d2e0dd4b

[8] https://insightplatforms.com/5-ai-enhanced-qualitative-research-platforms/

METHOD 3: Business Goals

While many pure algorithmic approaches exist that evaluate the software engineering process in isolation, there is an argument that any metric given by these methods are not very useful, at least from a business' perspective. This argument states that there is no point in making software engineering measurements unless they are paired with business goals. Steven A. Lowe, a product technology manager at Google makes the point "There is no point in producing academically perfect software with five-nines of reliability that the users won't use."[9] From this point of view, no piece of software should be viewed favourably by success metrics if it is useless to the user. The business value and the assessment result of the software engineering process should have strong correlation.

This method of evaluation further goes against the previous methods mentioned as it states that any metrics for individual performance should not be used.[10] That is, they should not be used in comparison to other individual performances. There is an analogy that every component in software engineering is like a snowflake; unique, valuable and incomparable. It is pointless comparing component metrics as no two components are the same, as are no two people, teams or projects. The only valid comparison is relative to individual history, and that should be tied to some sort of business goal.

In the "business success" assessment method, metrics should only be used to answer questions and to test hypotheses that support a specific business goal.[11] For example, adding an additional feature to the software will lead to more user interaction. If the evidence supports the hypothesis you may continue building on this feature, but if not, the feature may not be worth putting more resources into. Building on this, the success metrics can be combined with agile process metrics such as leadtime, cycle time, time velocity and open/close rates. These metrics aid planning and inform decisions about process improvement, while the success metrics will inform the business how to best achieve their business goals in the software development process. This combination for a business, it may be argued, is much more useful than a purely isolated and algorithmic approach to measuring and assessing the software engineering process.

Discussion: My Opinions and Ethics Concerns

My research for this report presented me with a number of interesting ideas about the analysis and measurement of the software engineering process. Earlier in the report, the multiple fundamental problems with the lines of code metric are mentioned. In my opinion, many of these problems are extensible to most of the metrics taken in the automated

[9] https://techbeacon.com/app-dev-testing/why-metrics-dont-matter-software-development-unless-you-pair-them-business-goals
[10] https://hackernoon.com/how-to-use-and-not-abuse-software-engineering-metrics-3i11530tr
[11] https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams

measurement of software engineering. Much as the LOC metric can be abused by engineers (those who know about it will write more lines regardless of quality), many other metrics can fall to the same weakness. If message board engagement is a metric, engineers will spend more time on message boards. If sentiment in communications is a metric, engineers will force positive sentiment regardless of how they feel. If there is some sort of upvote system in regards to engineers ranking each other on how helpful they are, there is a danger that it will become victim to office politics where engineers constantly favour their friends, again a way to abuse the system.

These flaws in the automation of the software engineering process are the area that give me the most ethical concerns. Businesses will get the most value out of the automated assessment and measurement of the process if they use them to make judgements on employees in regards to pay, contract status etc. as mentioned in the motivations section of this report. If the assessments are being used in this way, then job progression and opportunity will become available based on how well someone can abuse metrics – not how "good" of an engineer they are. A counter-argument could be made that the metrics could be hidden from the engineer – but this would give the engineer more grounds to argue about unfair treatment. If they do not know how they are assessed, then how can they improve?

At an even more fundamental level, I do not believe that any automated measure would be a proper representation of an engineer's level of productivity efficiency or productivity. I make the example earlier in the report of a system which could take quantitative metrics, then use some sort of chatbot interface to assess the qualitative metrics, such as the stress an engineer is under, or do sentiment analysis to assess the sentiment that they contribute to the team. However, I do not believe that even this model could derive the same results as human judgement. An employee would not give the same responses to a chatbot or any type of virtual system as to another team member, a level of trust builds up between actual humans to facilitate this type of interaction. I also agree with the argument put forward earlier in this report that every unit – whether it be a person, team or piece of software – is unique, and cannot be compared to another. An engineer or team can only be compared to themselves and their best work ethic. I do not see this approach as viable as such a large amount of data would have to be collected on each individual/team just to compare to their previous data.

The area where this automated assessment sees the most use to me is not in the measurement of individual teams, but of tools and methodologies. When measuring tools and methodologies we do not have to consider the same human factors as we would for teams and individuals. Specific categories of problems can be measured using different tools, which can then be compared. E.g. a dashboard may show one category of problem, feeding on past data on how that problem was solved using an object-oriented language, and how it was solved using a functional language. In theory an algorithm could feed on that data to tell you what type of language is best to solve the problem in. But that solution would eliminate human judgement; for example, an engineer may be twice as proficient in

one language than another, which the algorithm isn't aware of because the engineer spent most of their time coding in these languages on another company. Certainly the dashboard would help greatly, but human judgement over algorithmic decision making would be more beneficial in that example. So automation could help to inform a much better decision, but the human factor would still be crucial.

In a more general sense, this is my view of how the software engineering process should be measured and assessed. Fully automated systems should be crafted with great care and plenty of testing, taking account of the importance of certain metrics and how they can be abused. The algorithm may even take into account qualitative data such as sentiment and workload. But these systems should not make decisions about an individual; say the top 10% of engineers in the company, or the 20% least efficient. It would be unethical of a business to neglect their responsibility of this decision making to an algorithm which may fall victim to manipulation or insufficient data.

With this in mind, I believe that the best form of measurement and assessment of the software engineering process involves automation of data collection to help to inform decisions, but ultimately human factor should still play a significant role in making them. As mentioned near the beginning of this report, automation, when affordable, is a good investment that saves plenty of people hours of collecting manual data. It also allows you to collect more data. This data can help to much better inform a manager, HR department or team leader when it comes to assessing engineers or methodologies. At most, an ideal system will collect data and display it in a dashboard of statistics and visualizations to make it easier to handle. Indeed, many already exist for tracking software development practice.[12] [13] The system could even prompt invested parties if it "thinks" that certain information should be known, via email or an instant messenger bot.

I also agree wholeheartedly that a business following an agile development model should primarily focus on agile metrics. After all, once tasks are completed correctly and on time, the engineers are being as productive as required. In this case, a dashboard could also take account agile metrics such as if a team is on target for a sprint. I feel that a team following an agile approach would gain the most benefit from the measurement and analysis; a dashboard could show engineer quantitative data, engineer qualitative data and unique agile metrics. To me, this, along with human judgement would be a valuable and efficient method of assessing and measuring the software engineering process.

I take some aspects from the "business success" metrics into my recommended assessment model, such as the "snowflake" analogy and consideration of agile methods. However, I do not believe much more should be taken from this assessment model. In a small company, business needs and engineer tasks may overlap. But as the business scales, there should be a separation of ideas. The business ideas should be passed down to the engineers, who best

[12] https://www.klipfolio.com/blog/dashboards-agile-software-development
[13] https://chartio.com/learn/product-analytics/software-engineering-dashboard/

implement them. It is not a scalable model for engineers to be trying to appease metrics of "business success". What if their personal ideas for business goals are different? It also leads to a danger of software engineers trying to overlap into other areas such as product or design as they have no niche focus other than the profitability of the company. So while the "business success" metric was an interesting point of view that prompted many ideas, I do not think it is an ideal model for assessing or measuring the software engineering process.

Conclusion

In conclusion, it is important to note that the software engineering process can be measured and assessed under multiple different headings. An individual, team or particular tool or methodology may be assessed for some metric of effectiveness or productivity. Measurement may be manual, automatic or a combination of both. Different metrics may be important to different people; for example, lines of code committed, participation in chat threads, positive team sentiment, success under agile methods or alignment to business goals & contribution to profit. No matter what metric is used, there are ethical concerns when the metrics are used to judge individuals or teams. The measurements may affect an employee's career progression, for example. So it is important for a business to ensure that the measurements they are taking are reflective of the metrics that they actually judge by, and to consider how much of the process can be truly automated by a system. Overall it is a complicated proposal to "measure" an engineer or any aspect of software engineering, and certainly not one that can be solved by throwing any naïve algorithm at it.