Measuring Software Engineering

By Callum Jänicke

Content

Introduction: Life as a Software Engineer

Software engineers spend most of their workdays either doing important projects which have a certain deadline to be held, or doing bug fixes, code polishing or other rather mundane activities.[1] They mostly work in teams, doing their part of the project, before having a stand-up meeting every day of the week. Here they have to be able to present their progress, which puts them under quite some performance pressure. If certain deadlines are not held by individual engineers, they are letting the whole project, and with that, the other engineers down.

In the following report, the development of the software engineering field will be explained, in order to understand the drastic changes that have influenced this field over the last few decades, and that will keep on changing it in the future. After this, the question why software engineering should be measured will be discussed. Then, the report explains what kind of data could and should be measured, in order to be able to judge a software engineer's work. Subsequently, there will be a discussion about the computational platforms able to perform this measuring work. Afterwards, the ethics about these measurement processes will be discussed. Finally, there will be a conclusion made about the different kinds of measuring processes and the ethics surrounding this.

Development of the Software Engineering field

The software engineering field is rapidly changing and improving, as it is still a quite new field, compared to others that have been established for centuries. The field has its roots in the 1960s, being described as a field concerning how best to maximize the quality of software and how to create it. [2] At the start of it all, programmers did not even interact directly with computing devices. Instead, they delivered their programs by hand to technicians, who then implemented these and had them ready to pick up a few hours later. It was in 1968, that the term Software Engineering was really born, as there

[1] https://www.coderhood.com/a-day-in-the-life-of-a-software-engineer/
[2] https://en.wikipedia.org/wiki/History_of_software_engineering

was a conference held, with the intention of standardizing the best practices for software engineering. These were already widely used project management and production practices, which were just applied to the software engineering field. The result of this convention was a report, defining the foundations of software engineering. [3] In the following decades, the field grew more and more, with big projects being broken down into easier manageable modular pieces. In the 1980s, the idea of object-oriented-programming was born, which changed the field drastically. Until this day, there have been many more changes, that have changed the field massively such as the astounding increases in computing power, as well as the emergence of the World Wide Web, invented by Tim Berners-Lee.

In this century, the push for more and more open-source Software has had a huge impact on the field, with a hugely increasing amount of developers striving to code open-source. A quite recent development is the growing importance of the cloud, as well as the development of mobile computers. All these changes have drastically influenced the way software engineers work, and over the next decades, there will be lots of more changes, that will impact the way software is developed.

Why measure software engineers?

A very important question regarding the measurement of Software Engineering is: "Why is software engineering measured?" For one, if the work is done in a team, and one single member of the team is not getting their work done, it should be found out who is letting the team down. As John D. Cook states in his article "Some programmers really are 10x more productive"[4], there are big discrepancies between individual software engineers, and engineers that get more work done in the same time frame, should be rewarded for this financially, whereas firms have to also find out, which of their programmers are being very inefficient. As it is known, big companies measure their engineers, in order to cut a certain percentage, that is not working efficiently enough.

Measuring engineers helps companies monitor problems in the development process, enabling them to react early enough to either reward their employees, or in worse cases warning certain employees, that they must improve their productivity. Problems can be diagnosed early enough, before they become too hard to tackle. In addition to this, the measurement of engineers is detrimental to the time-planning that takes part. The progress must be measured, in order to see if the schedule is being held, or if there is a delay to be expected.

So, software development companies have no other choice than to measure their engineers, as it is the only possible way to diagnose problems in the project progress and to be able to react to these accordingly.

By tracking so-called software metrics, companies determine the quality of the current product or process, improve that quality and predict the quality once the software development project is complete. Managers try to increase the return on investment, identify areas of improvement, manage workloads, reduce overtime and reduce costs.[5]

[3] https://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering
[4] https://www.johndcook.com/blog/2011/01/10/some-programmers-really-are-10x-more-productive/
[5] https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th

Can software engineering be measured?

One of the most important questions regarding this topic is, whether software engineering can be measured, and if measurement is a meaningful indication towards the productivity and efficiency of a software engineer. According to Robert D. Austin, if you cannot measure 100 percent of something, it is useless to measure anything of it at all. He explains that if one was to measure less than 100 per cent, the result would be some sort of "dysfunction ". [6] As there are so many different variables involved in the development process, this task is nearly impossible. Measuring just one or a few of these factors, would not be enough to capture everything that is involved in this process.

Especially individual productivity is something, that is extremely hard to measure in software engineering. Nick Hodges explains in his article "Can developer productivity be measured? "that "there are plenty of good and effective ideas on measuring team and project productivity, but there is a severe if not complete shortage of ways to measure individual performance and productivity ". [7]

All this makes the measurement extremely difficult. On the one hand managers want to be fair, but with everything being somewhat subjective, this becomes a hard task. It is however not impossible to measure the engineering process, even if the lack of objectivity does make the process seem a bit more arbitrary.

Which data can be measured?

There are certain parameters that can be considered in order to measure a developer's productivity. Some of these are as explained in the paragraph above, not objective but rather subjective, as objective parameters are sometimes not sufficient to judge an engineer's work. For example, the lines of code someone writes, does not indicate how productive and efficient someone is working. In the contrary, such a measurement rewards inefficiency, as the longest possible code is never the most efficient. As well, just measuring the time someone takes to get a task done is not a good indicator of someone's work, as there are so many different factors playing a part and it would be unfair just judging by the time, as some members of the team might have been dealt more difficult tasks than others. The number off commits someone has done has also been suggested as a measurable parameter, and while a lot of commits indicate dedication to the project, the quality of these commits has to be taken into account as well. If only the number of commits were of relevance, developers would be rewarded for simply committing every single change to the code they have made, no matter how small or relevant.

Because measuring just these objective parameters would not be enough to judge a developer's work, the most useful way to go about this, is to combine the objective and the subjective parameters. Only going by time spent on the project, or number of commits, is the wrong way to go about it. But combining these with subjective parameters like their engagement and their attention to quality, helps being able to measure an engineer's work. Below, there is a list of metrics that, when all measured simultaneously, give a good overview of a developer's productivity and efficiency.

[6] Measuring and Managing Performance in Organizations, D. Austin

[7] https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo

Objective Metrics	Subjective Metrics
Coverage	Productivity
Complexity	Engagement
Time	Attention to quality
Commits	Learning and skills
Code Churn	Personal responsibility
Number of Bugs	Quality of tests

Objective Metrics

Coverage

Code Coverage describes the ratio between the lines of code in the software that is being tested and the number of lines of code, which are tested by all the executed test cases. It is agreed on, that about 80 percent test coverage is sufficient. How high the test coverage must be, coincides with how critical the system in question is.

Complexity

Programming complexity is the term used to describe all the properties of a software piece, that affect internal interactions. All interactions between the several entities are measured, and by increasing the number of entities, the number of interactions between these increases exponentially. There are many ways to measure this complexity, such as McCabe's cyclomatic complexity metric, Halstead's software science metrics, the Sneed Metric, Card Metric, Chapin Metric, Elshof Metric, as well as the McClure Metric. These all measure different kinds of complexity. The Sneed Metric focuses on branching complexity, whereas other metrics measure data access complexity, data complexity, data flow complexity and decisional complexity.

Time

Measuring time is a very straightforward measurement type. It is either possible to measure this for a whole project, or for a single engineer. Looking at the whole project, it is possible to measure the so-called Lead Time, which is the time period between the beginning of a project's development and its delivery to the customer. [8] Looking at a development team's history of lead time can be a good indicator to judge how long a project might take until it is finished. As well, a team's performance can be judged by cycle time, which describes how long changes to the software system usually take to be implemented by a team. As well, it is possible to measure the time a single engineer takes to complete their part of the project, in order to judge the work that they are doing. This alone is not a good indicator, as the complexity of tasks vary, but combined with the other metrics described here, it can help in order to assess the quality of someone's work.

Commits

Measuring the frequency of a developer's commits is also a popular metric in the software engineering business. Here both the quantity and the quality of commits should be assessed. A lot of commits

[8] https://anaxi.com/software-engineering-metrics-an-advanced-guide/

indicate a lot of work and dedication in most cases, but if someone committed every single line of code changed, this number alone would not be a good indicator for the work this programmer has done. There are many companies which desire daily commits from every developer, in order to assess this person's everyday work. Looking at the quality of each of these daily commits is definitely a good indicator over how much progress this individual is making. By committing often, developers can prove the work that has gone into a certain task, as it is sometimes hard to see all the work that has gone into a project by just looking at the end product.

Code churn

Code churn is another objectively measurable metric, that serves as a good indicator of a software engineer's work. It describes the percentage of a developer's own code that is an edit to their own work. This is measured by inspecting the lines of code that were modified, added and deleted over a certain period. Code churn can be seen by comparing separate commits, as it becomes easily visible to see which lines have been deleted, added or modified. Especially at the beginning of a project engineers tend to do this, as they do not yet have a clear solution for the problem at hand (Figure 1: "Evolution of code churn all along a project"). One difficulty with assessing code churn is, that it might be viewed as unproductive work and could be an indicator for poor management, unclear requirements or difficulty in the solving of the problem. It must be found out, with which intention the churn has been implemented, as good developer's do a lot of code churn in order to improve their code and optimize the performance. It is a good indicator to see how dedicated an engineer is and to diagnose problems in the project early enough to react to these.

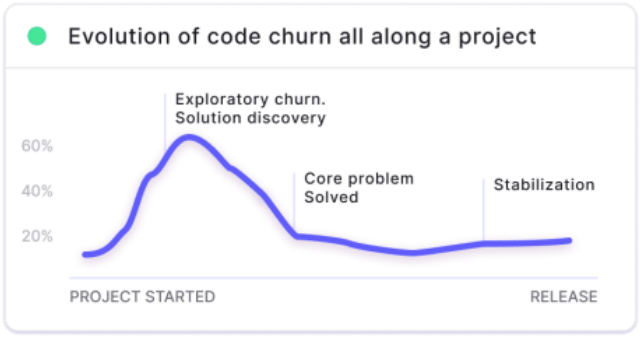


Figure 1: „Evolution of code churn all along a project "[9]

Number of Bugs

The number of bugs also indicates the quality of an engineer's work and is an important metric in the measurement of software engineers. It tends to be the highest around the middle of a project's lifecycle, with the focus on removing these bugs becomes increasingly higher in the weeks before the project deadline. While the number of bugs someone introduces into the project is an important metric to measure, the bugs also have to be inspected separately. It is possible to subdivide these bugs into different priorities, with the highest priority of bugs meaning they danger the functionality of the whole project. These are the bugs that a developer should really avoid implementing, and they are the ones which have to be detected quickly enough, before they endanger the whole project.

[9] https://anaxi.com/software-engineering-metrics-an-advanced-guide/

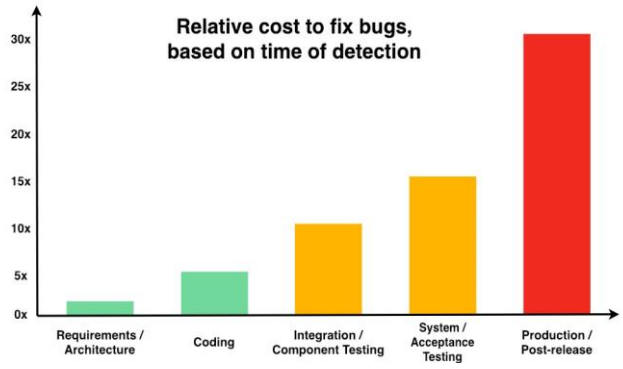


Figure 2: „Relative cost to fix bugs, based on time detection“ [10]

As one can see in the above figure, bug fixes become exponentially more expensive during the course of the project, which means that the priority should be to detect the bugs, especially the more severe ones, early on in the development lifecycle. Because of this, measuring an engineer's input of bugs is a very important metric in software engineering.

Subjective Metrics

The subjective metrics help with measuring an engineer's work in other ways than pure output and objectively measurable data. They help with cases, in which it is obvious to see who of the developers is doing the best work, without having measurable data to back up this sentiment.

Productivity

First, the question how productive a developer is working can be answered by asking two questions. For one, the question whether the developer gets a reasonable amount of work done in a given period of time can be asked. Furthermore, does the developer fix bugs with a sufficient velocity? These are very important aspects, that are not easily measurable with objective parameters, but they can be easily assessed by precisely looking at the work a developer is doing.

Engagement

The second subjective parameter that is taken into consideration is the engagement a developer shows with the project. It is to be found out how dedicated they are to their job and how committed they are to deliver the software on time. Furthermore, one can measure a developer's dedication to the company's success by assessing his work and especially by having conversations with said developer.

Attention to quality

Another metric, that gives a conclusion towards a developer's work and dedication is their attention to quality. First of all, the question to what degree the developer's code works has to be answered. As well, the readability of code, even though not too important regarding the functionality, is something that should be considered. Developers mostly work in teams, and that means that other people should also be able to read and extend someone's code. If the code is very clustered or there are hardly any comments, this can be quite a difficult task for the other developers.

[10] https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

Learning and Skills

Even though it might not seem to be too important on the first look, a developer's willingness and ability to learn and adapt is something that should not be underrated. Many developer's that are good at their work, do not show too much willingness to improve and to adapt to new challenges. Therefore, willing and being able to for example learn another coding language is something that makes a very good software engineer. It shows that this person has a passion for their job and that they are looking to progress in their career. Because of this, these qualities should not be overlooked when measuring a developer's work.

Personal Responsibility

A quality that many managers look for in their developer's is personal responsibility. Having a developer that understands the responsibility they have, and that do not look for errors in other people's code, but firstly in their own, is of a very high importance. Measuring this might a be bit more difficult than other metrics, but it is possible to get a feel for someone's responsibility by having conversations and monitoring their behaviour in certain situations.

Test quality

Lastly, test quality is something that is measurable, and it is a very good indicator to how well a developer works. Coverage of code has already been talked about and is objectively measurable, but it is also important to look at the quality of tests and not only at the quantity of code that is covered by the test cases. Questions that must be asked regarding test quality are, whether a developer thoroughly tests his code and checks it before handing it in, and whether the developer's test code covers all kinds of parameters, that can be passed on.

How can the data be measured?

Personal Software Process (PSP)

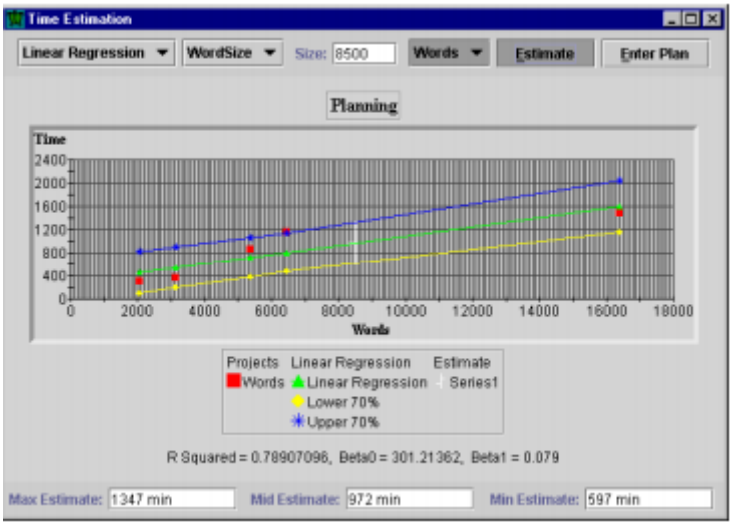
The personal software process was developed by the Software Engineering Institute (SEI) and is designed in order to assist software developers with their work. It is a method which gives an overview how software developers should plan and track their projects, use a measured and defined process, establish goals, and track their performance against these goals. The quality of an engineer's software is managed from the start until the end of the project, with the results being analysed and them being used for the next upcoming project. PSP has its priority in helping individual engineers and not whole teams and is therefore a good tool to assess an individual's progress. For teams, the SEI has developed a different method called the Team Software Process.

The aims of PSP are mostly to improve the engineer's planning and estimating skills, meeting commitments and schedules that have been made, reduce defects in their projects and managing the quality of their plans.[11] Originally, PSP was designed as a pen and paper method, but by now there are many tools for this method. In the next stages of this report, some of these tools will be discussed, with the focus lying on Leap, Hackystat and PROM.

[11] https://explainagile.com/agile/personal-software-process/

Leap

The Leap toolkit was designed in order to "provide Lightweight, Empirical, Anti-Measurement dysfunction, and Portable approaches to software developer improvement. "[12] By using Leap, software engineers can gather and analyse personal data, regarding time, patterns, defects and size. The Leap project began in 1998, with the aim of pursuing "low overhead approaches to collection analysis of individual software engineering metric data. "[13] One of the main differences to PSP is, that Leap views automated support as essential, whereas this is optional in PSP. Furthermore, while PSP gives a rather strictly predetermined process script to follow, Leap allows one to collect and analyse data more freely.



The figure on the left demonstrates a sample analysis in Leap. With this, the developer can make use of historical data in order to estimate the time required for a new project. The estimates are produced by applying linear regression or average/min/max analyses to historical data. The size metric which is used can be defined by the user.[14]

Figure 3: Leap analysis[15]

The Leap toolkit contains two central activities. First, Leap gathers primary information and then it performs Leap analyses. These central activities can be augmented by secondary activities which refine definitions, checklists and patterns. For instance, example goals can be set, and leap definitions can be refined, in order to reduce the overhead of data collection and aid the analysis. As well, Leap can collect primary data on size, time and defects and evaluate the progress towards the set goals. One weakness of the tool is the constant switch between doing work and telling the tool what work is currently being done. This was made out to be the major reason for the low level of adoption the toolkit received. This led to the Hackystat project being initiated in 2001.

Hackystat

In Hackystat, metrics are collected automatically, which is a big difference to the Leap Toolkit. This is realized by attaching sensors to development tools. These sensors can send the gathered information to a Hackystat server, which analyses and interprets the collected data. Furthermore, it can visualize the data to better understand the raw data that is extracted by the sensors. By this, the overhead of collecting data is effectively eliminated.

[12] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.7514&rep=rep1&type=pdf

[13] https://www.academia.edu/22196769/Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined

[14] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.7514&rep=rep1&type=pdf

[15] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.7514&rep=rep1&type=pdf

This figure shows the data that can be collected and visualized by Hackystat. In this instance, the two metrics Coverage and Complexity are measured. It must be noted, that not every development tool is compatible with Hackystat. Though many of the more popular IDEs like Emacs, JBuilder IDEs and the Junit testing tool are in fact compatible with Hackystat.

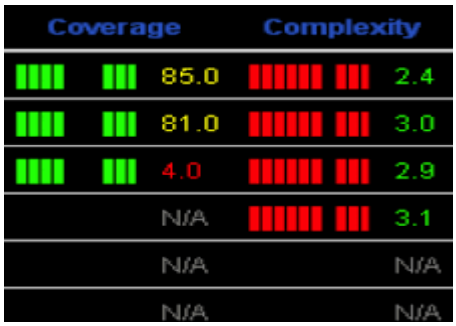


Figure 4: Hackystat visualization[16]

The sensors can collect all kinds of data, like activity data, size data and defect data. Looking at the server side of things, analysis programs are run regularly over every single metric of each developer, with the raw metric data stream being abstracted into a representation of the developer's activity at 5-minute intervals over a full day. This abstraction is called the Daily Diary, and it contains very important information regarding the measurement of an engineer's work. With this information it serves as the basis for analyses such as the amount of effort a developer puts into a module per day, the change in size of a module per day, the distribution of unit tests across a module and their success rate, as well as the average number of new classes, methods, or lines of code written in a module in a certain time frame. These analysis results can be made available to every developer over their account home page on the web server. Furthermore, Hackystat can send alerts to each developer, meaning that some threshold value has been exceeded. These alerts provide a "just-in-time" approach to metrics collection and analysis and can make developers aware of impending problems.[17]

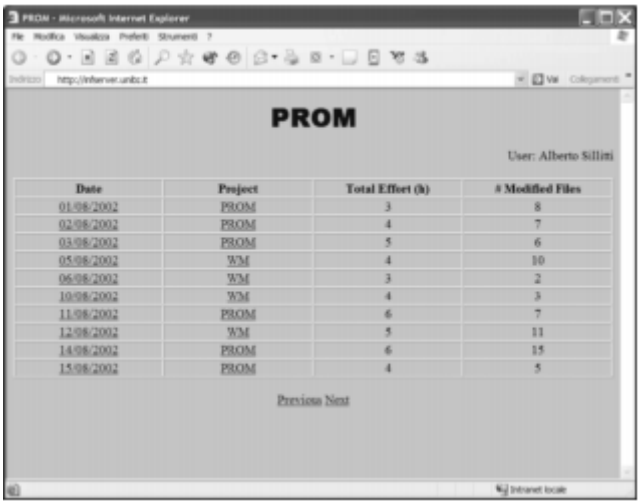
PROM

PROM is a tool, which collects code and process measures. The data that is collected includes all PSP metrics, procedural and object-oriented metrics and more. PROM is a component-based tool, this means it is based on a package-oriented Programming development technique. Both developers and managers can get very useful information out of PROM. The developers work with tools, which have an integrated plug-in which can automatically collect data, and have the system analyse this data. The system can then visualize this data according to the role of the user. The managers can only access aggregated data at project level, meaning they respect the developers' privacy. The picture below shows possible statistics a user might be shown while using PROM. [18]

[16] https://github.com/hackystat

[17] https://www.academia.edu/22196769/Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined

[18] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf

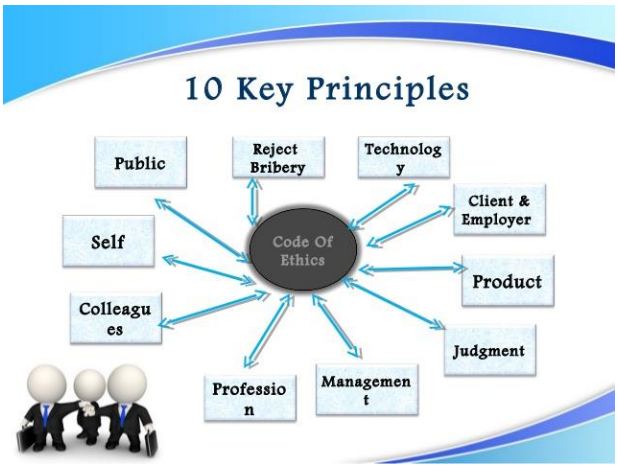


Compared to Hackystat, PROM offers support in C, C++, Java, Smalltalk and C#, whereas Hackystat only supports Java. Also, the list of supported IDEs is bigger for PROM. A big difference lies in PROM being package-oriented, whereas Hackystat is object-oriented. Both rely on PSP and PROM is rather project oriented, whereas Hackystat is more developer oriented. So, to conclude, PROM helps with monitoring and improving all of the development process, whereas Hackystat only focusses on the improvement of a single developer's work.

Figure 5: PROM statistics[19]

Ethics

The ethics surrounding this kind of analytics are widely discussed in the software engineering world. The IEEE Computer Society has complied a code of ethics, which software engineers generally agree to follow.



On the left, there are the 10 key principles agreed on. Public means that engineers shall always act in the interest of the public. Developers should also always act in a manner which is in the interest of their client and employer. As well, engineers should always ensure their products meet the highest possible standards. Further key principles include that software engineers should always maintain integrity and independence in their professional judgement and that the management should always make sure to deliver an ethical approach to their work. The engineers should always try to advance the integrity of the field and be fair and supportive of their colleagues. Lastly, software engineers should keep on being open to learning new things during their career and take an ethical approach during all this.

Figure 6: IEEE ethics code[20]

Having an ethics code is by itself nothing special compared to other industries. This must be agreed on when signing the contract in the first place. An engineer has, like any employee in another field, a responsibility to work within an ethical and legal framework. This is to make sure that an engineer does not make use of his skills in a way that affects the public negatively. But not only engineers have to follow this code, also their managers and the client must follow what has been agreed on.

[19] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf
[20] https://www.slideshare.net/Amjad061/ieee-code-of-conductethics

The question whether the monitoring of an engineer's performance through computational platforms and/or human observation is ethical is a very complex one to answer. First, the engineer that is being measured must be informed and have agreed to this kind of monitoring. Second, it should be made sure, that the software used to track this information gives a representative indication of someone's work and gives a fair comparison, when compared with other developers' work. The metrics that are covered should be agreed on with every developer, so that they know which stats are being monitored. By knowing this, they can focus on improving in these key areas like for example Code Coverage, Complexity and Churn. The monitoring of performance is not only good for managers to keep an eye on the project's progress, but also for each developer, because by knowing that they are measured concerning certain metrics, they will put a lot of effort into their work, which in turn helps them improve as engineers.

Developers must understand that it is in the manager's interest to improve the company's innovations and sales. To understand what must be done in order to improve in these aspects, managers need this historical data. The data provides them with valuable information regarding future projects, for example it becomes easier to tell how long a certain task will take, when comparing with the time it took for a similar task to be completed. Managers also want the best possible developers and if they can find out via a tool, that a certain developer is struggling to do what the others get done in a short period of time, it is in their right to do so and to send out warnings to engineers who have to improve their work. Also, very efficient engineers can be rewarded by measuring this, so measuring is not only there to discipline the worse developers.

Nevertheless, some software engineers feel like the monitoring has gone too far. While their sentiment is understandable and nobody likes being watched all the time, it is not like they are being monitored in everything they do. It is only when they work on the software that their effort is being measured, and this is fine if it has been agreed on. If the monitoring went much further than this, it would be unethical, and engineers would have a right to be upset about it. Developers might argue that the monitoring of subjective metrics might go beyond the measurement of their work, but these all still lie within what the developer achieves in the project on a day to day basis. All these metrics are essential for managers to make sure they deliver the client what they want and when they agreed on providing it. As long as the measurement does not surpass this, developers have to live with this form of monitoring. In no job would it be possible to never be assessed for the work one has done. The difficulty in software engineering is, that the measurement is often not very easy and is not as black and white as it might be in other fields.

Conclusion

Because measurement is so difficult to do in software engineering and as there are so many factors involved, the question arises whether all the data that is collected is even indicative of a developer's performance. While it is difficult to pick the right metrics to measure and the selection might seem a bit arbitrary, I do think that the metrics are a good indicator for a developer's work. There are so many different sets of data that can be measured, and together with the subjective measurement I do think that one can make good conclusions about how good of a job someone is doing. I also think that the measurement is very important, as "some programmers are really 10 times more productive" like John D. Cook figured. It is important to reward developers who consistently do good work and it is also of high importance to find out, which developers are struggling, in order to assist them or replace them with someone better, if there is no improvement to be seen over a longer period. As I have already mentioned, I think that measurement is not only good for managers to see who is doing good work

and who is not, and how well the project is progressing, but also for the developers as these tools give them a good indication on what to improve. Tools like Hackystat and PROM give feedback to the developers and signal when some sort of threshold value has been exceeded. PSP was invented to help developers better understand their performance and not for managers to monitor their employees. So, it is in fact something that developer, manager and client benefit from.

It is important for all of these to meet the ethics conventions that have been agreed on, to ensure a good relationship between all parties. It is understandable, that some engineers are sceptical about the level of monitoring going on, but as long as this does not exceed the measurement of their work and progress on the project, the developers have nothing to worry about, as every party benefits from this in the long run.

References

Anaxi: *Software Engineering Metrics: The Advanced Guide.* Available at:

https://anaxi.com/software-engineering-metrics-an-advanced-guide/ [Accessed 22.10.19]

Austin, R.D. (1996): *Measuring and Managing Performance in Organizations*. Published by Computer Bookshops.

Cook, J.D. (2011): *Some programmers really are 10x more productive*. Available at: https://www.johndcook.com/blog/2011/01/10/some-programmers-really-are-10x-more-productive/ [Accessed 21.10.19]

Dario, J. (2019): *Measuring Developer Productivity.* Available at:

https://hackernoon.com/measure-a-developers-impact-e2e18593ac79 [Accessed: 21.10.19]

Hodges, N. (2018): *Can developer productivity be measured? Available at:* https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo [accessed 24.10.19]

IEEE Computer Society (1999): *Code of Ethics*. Available at: https://www.computer.org/education/code-of-ethics [Accessed 27.10.19]

Johnson, P.M. (1999): *Leap: A "Personal Information Environment" for Software Engineers*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.7514&rep=rep1&type=pdf [Accessed 26.10.19]

Moore, C. and more (2003): *Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined.* Available at: https://www.academia.edu/22196769/Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined [Accessed 27.10.19]

Pasqualis, L. (2017): *A Day in The Life of a Software Engineer*. Available at: https://www.coderhood.com/a-day-in-the-life-of-a-software-engineer/ [Accessed: 21.10.19]

Sillitti, A. and more (2003): *Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data.* Available at:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf [Accessed 27.10.19]

Stringfellow, A. (2017): *What Are Software Metrics and How Can You Track Them?* Available at: https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th [Accessed 23.10.19]

Viking Code School: *A Brief History of Software Engineering.* Available at: https://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering [Accessed 21.10.19]

W., Björn (2018): *What is Personal Software Process? Available at:* https://explainagile.com/agile/personal-software-process/ [Accessed 27.10.19]

Wikipedia: *History of Software Engineering*. Available at : https://en.wikipedia.org/wiki/History_of_software_engineering [Accessed: 21.10.19]

Figure sources

Figure 1: https://anaxi.com/software-engineering-metrics-an-advanced-guide/

Figure 2: https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

Figure 3: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.7514&rep=rep1&type=pdf

Figure 4: https://github.com/hackystat

Figure 5: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf

Figure 6: https://www.slideshare.net/Amjad061/ieee-code-of-conductethics