Measuring Software Engineering

Tin Orešković

November 21st, 2018

Contents:

Introduction

There are numerous ways for employers to measure productivity of their employees in software engineering and there are many ways to do so, which will be discussed later. The important question is: Why is measuring the software engineering process so important for a company. The answer to that question is very simple. Every employer wants to know whether their employees are their jobs which is expected of them, if they weren't they wouldn't be doing that job. However, maybe the most important aspect is productivity, how efficient is a developer. That can be measured in a lot of different ways: lines of code, number of commits, code coverage, defect rate, hours worked, accuracy of estimation and many more. However, I will explain only the ones mentioned in this paragraph. Also observing someone's work and measuring it can help employers make their employees more productive in the future.

Measuring Software Engineering process

Lines of Code

This is probably the easiest method to measure one's productivity. However, I believe it's one of the worst ways to do that. It is true that if you write more lines it, more or less means that you wrote more code and some people would think that means that someone was really productive. On the other hand, developers can just write many useless lines of code or just take the simplest approach which extremely non-efficient and slow. That's why I think that this approach is extremely bad.

Number of commits

Keeping track of number of commits would be a better idea. If one commits often you can keep track of their work and many commits essentially means that they spend a considerable amount of time working on the project. That being said, there is a bad side of this measurement process. A person can commit every single time they make a change in their code, not a significant one, it can be just a mistake in spelling, or literally adding a variable. In conclusion a programmer can commit even the smallest change or no change what so ever and then keeping track of number of commits would be a bad idea because it doesn't represent one's productivity.

Code Coverage

Testing a code is generally a good idea in programming, especially if you work on a project in a group. Code coverage means how much code has been covered in those tests, so the higher the percentage is, the higher is the chance that your code works the way it should since it shouldn't have any bugs. All of that is true under the assumption that correct tests were written for a specific code. My opinion is that this approach is by far the best out of the first three I mentioned, although it is flawed as well.

Defect rate

This process is based upon measuring the number of defects a developer produces while working on a project. The method seems reasonable enough since the number of bugs is being tracked. However, there are some issues with this approach. Mainly the problem with this method is that people focus way too much on fixing bugs rather that development of the actual project. There are also some minor problems like there may be several different bug reports that a related to one bug only which would make the defect rate worse for that person. So in general the process is fine, but it has its problems.

Hours worked

This is also one of the most obvious ones, but I believe it's one of the worst ones as well. Obviously if you are going to do more work if you work 10 hours and not 8 hours. However, productivity decreases with time, so if you work 10 hours straight someone's concentration is going to get quite low and they will do almost no work by the end of the day. One's productivity is the best during the first 6 hours of work, 8 hours maximum. This can lead to people focusing way too much on the time they spend working and not on the actual task. That can also ruin the morale of developers.

Accuracy of Estimation

Estimation is one of the least common methods of measuring the process of development and if you asked me not a good one either. Accuracy of estimation is a process where a developer estimates how long it will take them to finish a certain task. That means that, for example someone says that they will finish the project in a week and if they don’t finish it within a week they will probably get in trouble. However, if they finish it ahead of schedule they will be praised for it, which can lead to a lot of problems. Let’s say that you estimated that it will take you a week to finish a task. If that task usually takes 10 hours to do, you can just do 1 or 2 hours of work a day, which extremely decreases productivity.

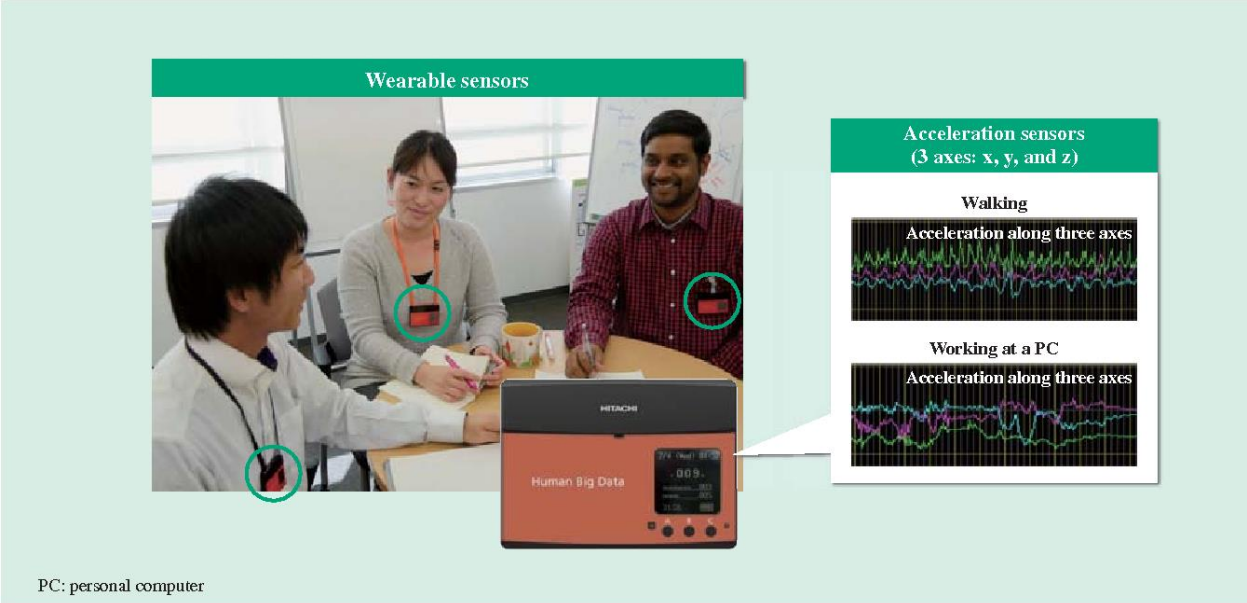
In conclusion, traditional approaches of measuring software engineering process are extremely insufficient and flawed. While some may look better than others which are just horrible, all of them have many disadvantages. However, a combination of methods mentioned would maybe be a good idea, although even then it would be possible to find flaws in that system.

Computational platforms

There are numerous platforms that were developed to make measuring the productivity of software developers easier. The idea was that those traditional measuring methods are not good enough and that there need to be a better way to measure someone's productivity. Every method is different however, they focus on different aspects and they measure different things in the process of development.

Measuring Happiness Using Wearable Technology

Another way of improving developers' performance and productivity is to measure happiness and physical activity. There have been many studies that prove that happier people are more productive as swell as more creative (according to one study happier people are 37% more productive and 300% more creative). That's why scientists invented Wearable sensors that identify the correlation between happiness and physical activity. Which also proved that there is a relationship between physical activity and happiness. In conclusion, physical activity makes people happier and increases their productivity.



Source: https://www.semanticscholar.org/paper/Measuring-Happiness-Using-Wearable-Technology-%E2%80%94-for-Yano-Akitomi/fef7d51d9f7c641631e96ff820e29ab73b50d0d5

Personal Software Process (PSP)

Collaborative Software Development Laboratory (CSDL) at the University of Hawaii at Manoa have look for some kind of analytics that would help software engineers increase productivity, so they started using the PSP that was described in Watts Humphrey's book called "A Discipline for Software Engineering". That version of the PSP uses manual data collection and analysis, which makes it extremely exhausting to manually enter all that data to be analyzed. For example, in one specific version of PSP, developers must fill out 12 forms and manually calculate more than 500 distinct values. All those aspects make PSP quite fragile, but extremely flexible as well. Also PSP had huge data quality problems since it sometimes led to incorrect process conclusions even though it had low error rate.

LEAP

Because of numerous problems that occurred while using PSP, developers from the University of Hawaii created a new toolkit called LEAP (lightweight, empirical, anti-measurement dysfunction, and portable software process measurement) that would address certain problems with PSP by automating and normalizing data analysis. Developers can control their data files, but it won't reference developers' names. It is also portable which means that it creates a repository of personal data that a developer can transfer from project to project or from company to company. LEAP is a good alternative for PSP, but it still has some problems

Hackystat

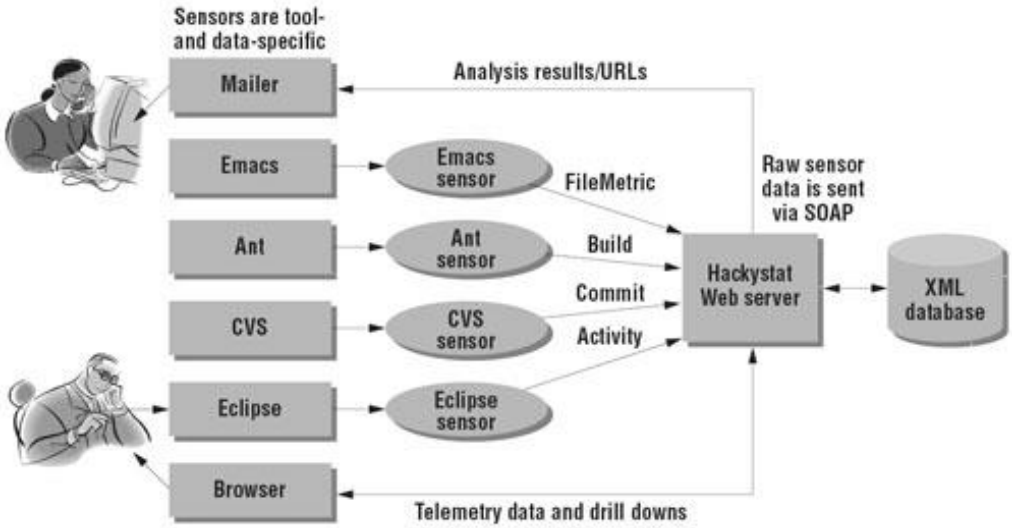
This is the developer data analysis software that was created by the same team as LEAP because of understanding how limited are both PSP and LEAP. This time they focused on different aspects. They focused on collecting data with no overhead for developers and after that they decided what high-level software goals could be supported by analyses on that data.

Hackystat has four main features:

- It has a client and a server-side data collection, which means they provide IDE's or build tools, but they also provide server-side tools.
- Obtrusive data collection. That means that it is really hard to get interrupted while using Hackystat, because its client-side caches any data collected when a developer works offline, so that when they go online cached data is being send to a repository.
- Fine-grained data collection
- It also supports both personal and group development. It collects personal and group development data as well as it tracks the interplay among developers.

All these new innovations were paid off because Sixth Sense Analytics incorporated Hackystats into a commercial offering. It keeps track of four different data:

- DevTime – how much time a developer spends on their IDE
- Commit – how often a developer commits and how many lines they commited each time
- Build – how many times a developer built and were they successful
- Test – how often a developer tests his project



Source: http://141.44.17.27/cms/index.php/ja/home/forschung/128-smha

GitPrime

This computational platform was founded by Ben Thompson and Travis Kimmel in 2015. It is a software engineering analytics company which is currently based in Durango, Colorado. We already know that Git-based solutions such as GitHub, BitBucket, GitLab and many more help developers when they work on their projects as a group. GitPrime was created to keep track of team progress and collects data related to the project development using Git-based repositories. After that it provides engineering metrics which helps developers see their progress and help them with problems they encounter.

Codacy

GirPrime is the probably the most popular solutions for data analysis, however there are some alternatives which are somewhat different, but still idea is the same. One of those platforms is Codacy. It is fully automated code analysis tool, using Codacy developers get static analysis, cyclomatic complexity and code coverage from unit tests for every single commit and pull request. It can track your overall code quality as well as track the quality over time, which means it compares quality of the code from the beginning of the project till when the project development is finished. Codacy also supports more than 20 languages.

Source: https://github.com/marketplace/codacy

Algorithmic approaches

Agile

Agile software development is an algorithmic approach to software development in which problems are being solve in the group through self-organizing and cross-functional teams. It advocates planning, step-by-step development and continuous improvement. It also encourages fast response to change. Agile software development has many different methods, but most of them break development work into smaller sub-parts. They essentially use iterations (or sprints) which are small time frames that are usually from one to four weeks long.

In each of those time frames a cross-functional team works on each function such as coding, testing, planning, analyzing and more. An interesting aspect in this approach is that testing is done in every time frame, that's why it is an iterative approach.

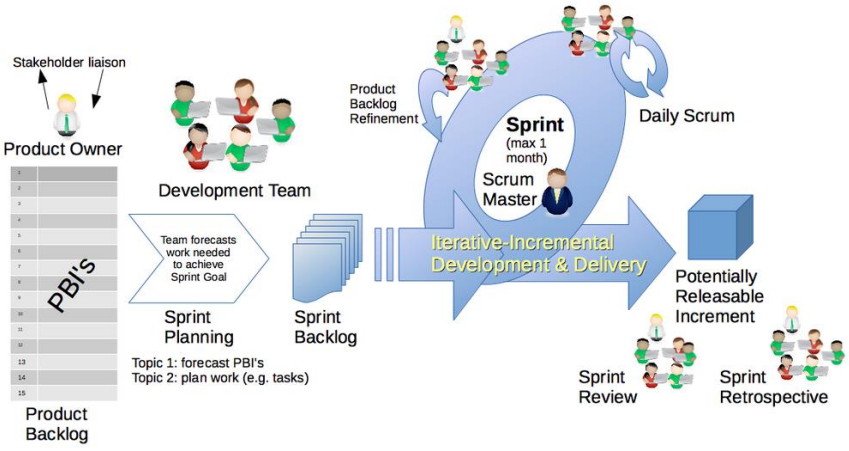
The "Manifesto for Agile Software Development" is based on twelve principles:

1. *Customer satisfaction by early and continuous delivery of valuable software*
2. *Welcome changing requirements, even in late development*
3. *Working software is delivered frequently (weeks rather than months)*
4. *Close, daily cooperation between business people and developers*
5. *Projects are built around motivated individuals, who should be trusted*
6. *Face-to-face conversation is the best form of communication (co-location)*
7. *Working software is the primary measure of progress*
8. *Sustainable development, able to maintain a constant pace*
9. *Continuous attention to technical excellence and good design*
10. *Simplicity—the art of maximizing the amount of work not done—is essential*
11. *Best architectures, requirements, and designs emerge from self-organizing teams*
12. *Regularly, the team reflects on how to become more effective, and adjusts accordingly*

Source: https://en.wikipedia.org/wiki/Agile_software_development

Scrum

Scrum is an Agile framework for managing work in software development. It is made for three to nine members in a group and just like in Agile development method, the work is divided into small iterations (or sprints) which are no longer then one month, usually they are two weeks long. Group members track their progress and plan in daily scrums, which are stand-up meetings that last about 15 minutes. The most important principle of Scrum is the recognition that a customer will probably change their mind at one point about what they want in a project, that's why predictive approach is not possible since there would be numerous problems. Because of that, in Scrum, the focus is on the team's ability to respond quickly to emerging requirements.



Source: https://en.wikipedia.org/wiki/Scrum_(software_development)

Ethics Concerns

I believe that by far the biggest ethical concern is manipulating with someone's personal data. That is a big problem because, as developers from the University of Hawaii said, the easier it is to collect and less controversial it to use someone's data, the more limited is its usefulness. I still believe that each developer needs to consent and after that the platform can start collecting data.

For example, there was a problem with Hackystat, made by developers from the University of Hawaii. One of its features is that it is unobtrusive, which means that the program will never interrupt you when you are working on your project. It will keep gather data even if you are offline. Some people didn't like that since they didn't want the client-side to collect data without telling them about it, so they didn't want to install that feature at all. Even though developers found it extremely useful, numerous developers were not satisfied with that aspect of Hackystat.

Conclusion

In this report I mentioned some traditional ways of measuring software engineering and how they are being executed. I listed some computational platforms and described them. Finally, I mention two algorithmic approaches and gave an opinion on ethical concerns in collecting personal data.

I believe that data analysis in software engineering is of great importance. It can provide an employer some information on their employees. That makes helping developers much easier since it is obvious what are the main problems. However, every single approach has problems, there is no perfect method of measuring software development.

References

- https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity
- https://en.wikipedia.org/wiki/Software_engineering
- https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams
- http://csdl.ics.hawaii.edu/techreports/2012/12-11/12-11.pdf
- https://en.everybodywiki.com/GitPrime
- https://www.gitprime.com/
- https://github.com/marketplace/codacy
- https://en.wikipedia.org/wiki/Agile_software_development
- https://en.wikipedia.org/wiki/Scrum_(software_development)