**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.

**Traditional Measurement of the 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.

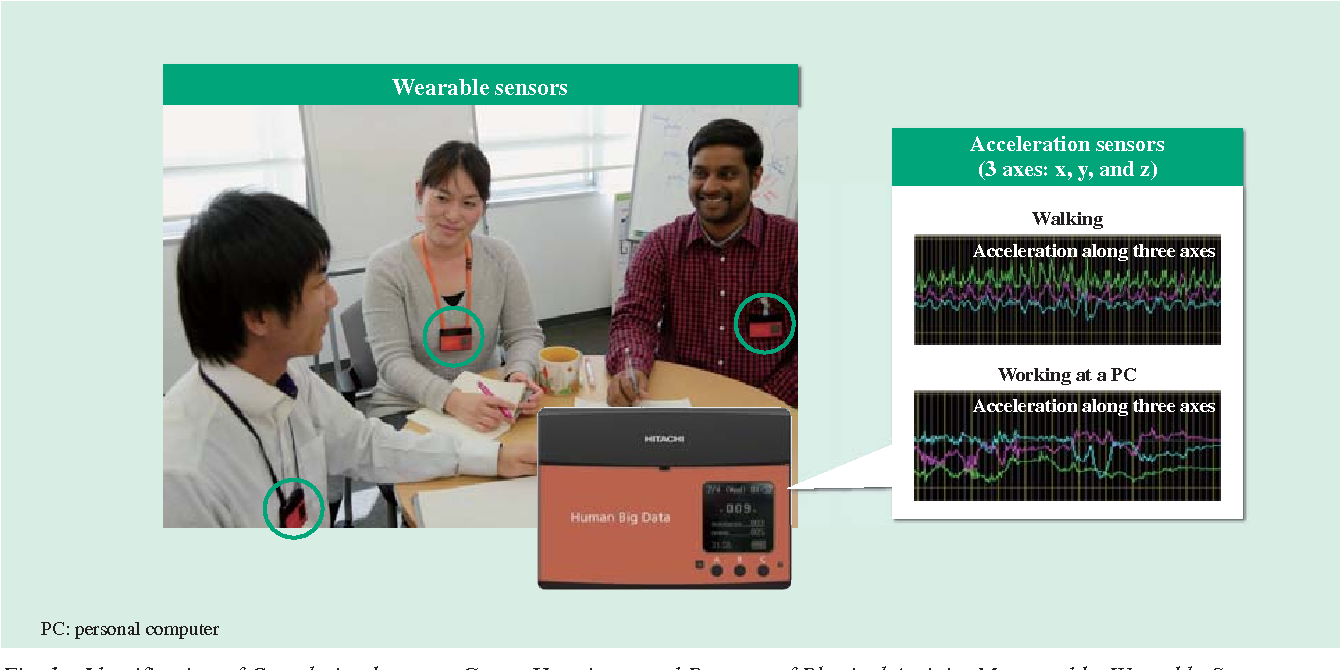
**Accuracy of Estimation**

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.

**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

**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.

**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**

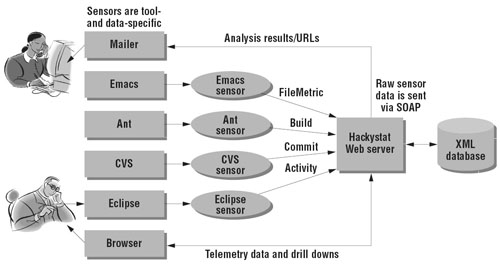
This is the developer data analysis software that was created by the same team as LEAP because of understanding how limited are both PSP and LEAP. This time they focused on different aspects. They focused on collecting data with no overhead for developers and after that they decided what high-level software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

**References**

* [**https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity**](https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity)
* [**https://en.wikipedia.org/wiki/Software\_engineering**](https://en.wikipedia.org/wiki/Software_engineering)
* [**https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams**](https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams)
* [**http://csdl.ics.hawaii.edu/techreports/2012/12-11/12-11.pdf**](http://csdl.ics.hawaii.edu/techreports/2012/12-11/12-11.pdf)