A general report and analysis of software engineering measurement and theoretical development

Summary

The purpose of this report is to examine techniques relating to the measurement of software engineering. The reasons why one would want to measure one's work, and what ethics should be considered before doing any of these things. With software engineering being one of the fastest growing fields of work and having an increasing impact on our world the measurement of it will too become increasingly important.

Measurement of work

Since software engineering, or programming, is inherently done on computers, every single thing that is done can be monitored. The main purpose one would do this is most likely economic, as from a companies point of view what is important to it is how much money is being spent per addition to a piece of software. For personal use it could be used for seeing one’s own improvement, in terms of how long it takes someone to solve a programming puzzle or problem.

How might we define the amount of work done. Considering that the other factor to compare to is very standard and obvious that one would use time. A quite obvious measurements one might use might be simply number of lines used by a programmer. This in fact has an acronym of “LOC” or, lines of code.

Within this you can measure two different types of “lines”, literal lines of code, or logical lines of code. While it might seem a tiny difference in terms of semantics it is anything but, and has a great impact on the final count.

```
for (int i = 0; i < 100; i++) printf("Number %d", i);
```

If one measures the physical lines of code, this is only counts as 1 line of code. However if measured by logical lines of code, it counts as two lines of code.

```
for (int i = 0; i < 100; i++)
{
   printf("Number %d", i);
}
```

This is the same piece of code just written differently, and now the literal lines of code count is 4, but the logical lines of code stays the same.

Literal lines of code can be a good measurement in some conditions, such as with a well defined coding standard and inspection of code. They also have the advantage of being inherent with the medium that you can just count the “new lines” of code to get your result, whereas with logical lines of code, since each programming language they need separate utilities.

Both of these methods of measurement have some very large and glaring issues if they are the only thing you judge productivity by. Since LOC cannot verify if code is particularly good or not it can only be used as a metric if there is faith in the programmers ability. It does not account for many other factors people may care about such as efficiency, readability, or even functionality.

There is no way to check that code works with LOC, which is a big problem for those solely using it to measure productivity.

With logical lines of code, there is no standard or baseline as to what is even considered a logical line of code. And the number of instructions ran cannot even be used as maybe another piece of code solves it in fewer instructions. This, coupled with the vast number of programming languages makes it difficult to assess a programmer's performance without accounting for each one they might use.

If a programmer thinks or knows that their efficacy is being measured by LOC, there is not much to stop them besides people inspecting their work from writing extra lines or making things longer just so they seem like they're doing more than they are. Not to mention the stress or worry it may cause someone if they think about it, and the impact that might have on their own work as a result.

Code Effectiveness

Let us examine another technique one may use to try measure productivity. And that is code coverage. Code coverage is a technique to see how much of the code is covered by test cases that are typically also written by the person who wrote the code. These can be surprisingly effective at making programmers check and account for weird cases in which their code might break or be abused.

Most languages come equipped with libraries (either native or foreign) that may be used to write out test cases and analyze code coverage. Typically there are more than 1 to choose from for larger languages.

Code coverage testing is also very effective at large scale projects, in which cases multiple people may be working on different aspects of the project and modifying things, the tests can be run to check if a change made something cease functioning or introduced a new bug.

While very useful some may consider code coverage to be a waste of time as it means programmers work longer on their code, and less overall time writing new code. This can be put down to a fundamental issue of quality vs quantity and should be analyzed on a case by case basis.

This technique also does not entirely measure a programmers efficacy as it only checks that the code they write passes through all the tests that they also write, so while very useful for improving the safety and security of code, it is not a great measuring tool.

Maintainability

One aspect many people may seek is software maintainability, this is how easy code is to manage and maintain over the coming years. As other software changes, in many cases so must our code as not only to use the latest hardware to its maximum ability, but also to ensure that ones code runs on a new operating system or is updated after a vulnerability is discovered.

Typically, a software project with many developers will only take a few years to develop from nothing. After this it must be supported for around 10-20 years if not more, in which time the software engineers must be able to fix the code when issues arise.

An increasingly popular solution to this kind of problem is “DevOps”, which is taken from the words development and operations. The idea of it is to monitor a software project as it is being developed so it can be fixed as its being developed in a greater manner than with a split system. Typically many of the developers for the project will move to the operations side of things while also

helping make fixes in code. An obvious advantage of this approach is that the people most familiar with the code will better understand what the issue is when one arises and can directly see it for themselves.

One of the key factors in ensuring code scalability is the modularity of the project. This is a concept that applies from everything from small, one person projects to large, multi team ones. The general concept is that you should be writing modules of code and using them, at typically a small scale. It is much easier to understand code when it is represented as a function that only does one thing, rather than a piece of code within other pieces of code. This is especially crucial for large scale projects as not only will it be more efficient for the team to re use other members code rather than writing it themselves, it will also greatly improve readability of the code.

Security

One of the most important factors that has not yet been mentioned that many companies will seek is the security of code. With increasing reliance on software in day to day use, how do we ensure that the code we make is secure and cannot be hacked. This is one of the most complicated problems in software engineering and has a vast number of aspects to it.

Firstly securing code itself, for a closed source project. A very common technique is code obfuscation. Code obfuscation is the process of taking existing code and renaming variables and functions through use of a programs to make it practically unreadable. This makes it very difficult to reverse engineer a software project but makes absolutely no difference to the processor that runs the code, as variable names and function names are basically arbitrary and as long as they are unique will have no impact on it.

Anti forensics can also be used to dissuade employees from leaking and selling code to competitors, however this is not much of a preventative measure as much as it is retaliation for violations for a contract. Anti forensics is the analysis of a person's work hard drive and usage of the computer to see if code has been copied or not. Typically this is done after a person leaves the company and moves to a competitor.

Second way one might think of security, is securing your own systems and databases. How one might do this? Ideally, they wouldn't. It is best practice to not be securing data yourself if at all possible as it is **extremely** easy to mess up in such a way that entire databases can be compromised. In most cases, it will be cheaper and safer for you to store data in the cloud or let a specialised company handle it. However, if it absolutely necessary to store your own data on your own servers, there is a variety of things one must do.

The data should ideally be inaccessible, not only through security checks but put off site in a remote and highly secured area. Data should always be encrypted in such a way that even if someone gets access to it, it is of no use to them. Lastly dedicated penetration testers should but found to check the security of the site and data.

Ethics

A question one might ask at this point, is what are the ethics of using such techniques and should it even be allowed? Considering computer programming is an emerging field, it has not been philosophized about for that long, especially in comparison to fields such as medicine, in which the

Hippocratic Oath and other such ethical standards have existed for over 2,000 years.

Software is typically seen in a virtual sense, that its only there when we open it so it can be very difficult to see the real effects of software. With new Internet of things devices, self driving cars, and increased automation this will change very soon. Because of this software must take into account, along with the way in which to observe the progress on software engineering projects.

Because of this it is essential that developers are given clear information on what the software they are developing will be used for, as with certain projects, some developers may not be willing to participate in. If they are forced to, under contract, this may not be a good option for the company either as they might not do good work or only do the absolute minimum that is required of them.

Psychological effects on quality of work

Since problem solving, and therefore software engineering is highly reliant on the state of mind of the person it is highly important that one does not push a software engineer too hard by imposing overly strict deadlines and hours. At a certain point there are diminishing returns in terms of what you can expect from your engineers and will not be worth it for the small amount of productivity you will receive back. A balance of deadlines on projects and people is necessary and this will be variable from team to team, and project to project.

Luckily since it is presumed that you are already measuring peoples, and teams performance as they code, one can assume you can see if a certain

change in the workplace will have been beneficial or not to workers. This is one of the better uses you can have for the measurement of software engineering is improving the conditions of work.

However this use case is a double edged sword, while it may be worthwhile to improve conditions and productivity, the impact of simply measuring peoples performance is a slight detriment on average to people. This is because people will be slightly more worried as to how they are doing than if they were not being observed. Therefore it is of utmost importance that the workplace uses this data ethically and does not seek to punish under performing programmers in a rash way, and rather tries their best to bring out the best of their programming.

It is difficult to tell if the performance metrics on teams and people should be public to others, for how might one understand their performance numbers without comparison. And is it possible to prevent team members from being overly competitive early on which could lead to burnout.

The best way that I think to do this is by publishing anonymized data, so a person is able to compare their own performance to their team without knowing specifically how any other person is doing.

Sources

https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf
http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf
http://www.citeulike.org/group/3370/article/12458067
http://patentimages.storage.googleapis.com/pdfs/US20130275187.pdf
https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams
http://scet.berkeley.edu/wp-content/uploads/Report-Measuring-SW-team-productivity.pdf
https://pdfs.semanticscholar.org/5b94/00ba2a7184d5dc0a4fa6029ae6a52e7fbb6b.pdf