Measuring Software Engineering

Dominique Meudec

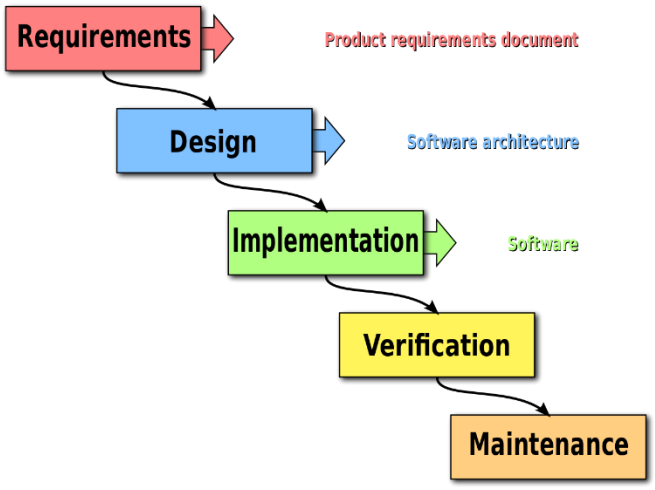
18327666

Introduction

Measuring the productivity of somebody's work can always be tricky, particularly in software engineering where not all work done can be measured in the same way. When measuring anything, there are variables that have to be taken into account and how these variables weigh up against each other in terms of importance and value. What exactly determines the value of somebody's work in software engineering? Is it product requirements, delivery time, or the cost involved with it? A lot of the time, a good worker is an efficient worker, where their positive work far out ways their negative work. For software engineers, the question is what metrics are used to find somebodies efficiency.

Software Engineering Process

To be able to measure software engineering, we have to be able to understand the different types of project structures that could be used by a team or individual. Throughout time, the discipline of software engineering has changed, and the project structure has changed as well. One of the first types of software development is the Waterfall model. In essence it is very simple, where there are many stages to a project, the project doesn't move to the next stage until the current stage is fully complete and working.



The Waterfall model is very simple and functional, its easy to test and to understand where the project is at. Programmers are able to make sure each part of the project is perfect before moving onto the next part. This means that there is no backtracking involved and in essence the project should run smoothly. However, overtime the way software engineering worked as a whole has changed. Now clients are asking for changes to be made in previous stages and such changes could render the latest parts useless. In today's world, clients demand a lot of flexibility and want results quick. The Waterfall model simply doesn't allow for this, as what is planned at the start is what gets finished at the end. Agile development is a development of this model and is the most popular in software engineering.

Instead of betting everything at launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly. In essence, Agile allows for the ability to create and respond to change. While Agile allows for flexibility and quick reviews, there can be uncertainty and lack of predictability in the project.

Measurement and Assessment

After briefly covering two of the most popular methodologies in software engineering, there are clearly still issues in how we measure the efficacy of the work being done. There can be a reliance on looking at the amount of commits and lines of code written in a period of time, but this doesn't tell us of the quality of the code produced. Some problems require more lines of code whereas others may not, despite their difficulty being the same. There is also the issue of a programmer rewriting the same problem again and again and committing these changes because somebody else makes a change to the structure or there is a system update. Should an important metric then be the amount of time solving a problem? The size and the importance of this problem, then has to be taken into consideration to allow a fair evaluation of the programmer's efficacy. These metrics are still used and will always be but are there other metrics which provide a more accurate and truer representation of somebody's work.

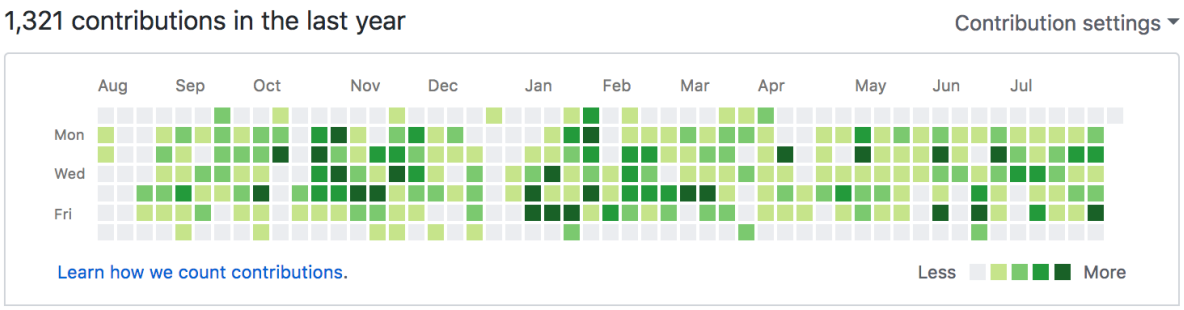
No one project or company a programmer works on is the same as another's, so to measure across other's can be challenging. Within each project as well, everybody has different skill sets which others may not have, which you can't put a price on. The challenge for measuring software engineering is the diversity of each and every programmer. While we me never get a 100% evaluation on a programmer's efficiency, there are ways to get a very good idea of it.

Measurable Data

For Data to be of any benefit to a company or programmer it is quite important that gathering this data is easily collected and doesn't become another task to do. Having data that is accurate and true is one that is automated and doesn't interfere with a programmer's work. If in measuring software engineering the person had to manually record such data, the time taken to do this would simply render any measure of productivity worthless. This is simply

because that software development teams may consider it more important to actually do the work than to measure it. Hence, it becomes imperative to make measurement easy to collect or it won't be done.

Measuring and analysing work shouldn't be anything that is burdensome or halts the creation of code. Software metrics should have several important characteristics to be of use. They should be simple, computable, consistent, adaptable and in general, accurate to the development of the software product. The most important thing about the data collected is how does one use this data to be able to accurately measure somebody's productivity? No two software development teams will be working on the same project. Each and every project is different, so should the way the data used be different every time?



The above image is an example of a user's contributions over a year, but does this truly measure somebody's productivity? This would be far more useful in the sense of measuring how active a programmer is, rather than their efficiency. We can't just look at the number of commits of code, rather we have to look at the quality of code. These 10s or 100s of these commits may be in relation to solving one issue within a project, in such a case the time taken to deliver a fix from start to finish would be far more accurate. The data should also take into consideration the importance or skill required to deliver a product. Some products don't require as much skill as others, and much more time may have to be factored into this. Two

types of metrics that are often used are “Size-related”, indicating the size of outcomes from an activity, and “Function-related”, indicating the amount of useful functionality shipped during a set period of time.

While we have many tools available to analyse code data, there should be a way to take into consideration the other elements involved with the job. Analysing code isn’t the only metric that should be used, other factors like planning, meeting and organising are key factors in delivering a project successfully. Such simple things can save hours of work further into a project if the right organisational calls are made. These metrics often can’t be measure throughout the project however, as a project can only tell truly how efficient it was at completion.

Computational Platforms

There are many ways in which companies measure software engineering, from Git to Fitbits, each measuring different aspects of a programmer’s productivity. Git is by far the most common in practise in many different companies. While Git is primarily used as a version control and repository language, allowing teams of programmers to work together on the same project. Git is really effective at doing this and is by far the most commonly used platform for teams to program together. While Git provides these functions, it does however also provide information on the details on the change’s users make. Number of commits, number of lines added, changed, deleted as well as comments can all be tracked by Git. How is this of any use to a company? Well for instance they could track the frequency of commits a programmer is making, the quantity of code added or the comments a user makes. While

these metrics are useful in analysing the frequency of a programmer's work, they don't reflect the quality of code added.

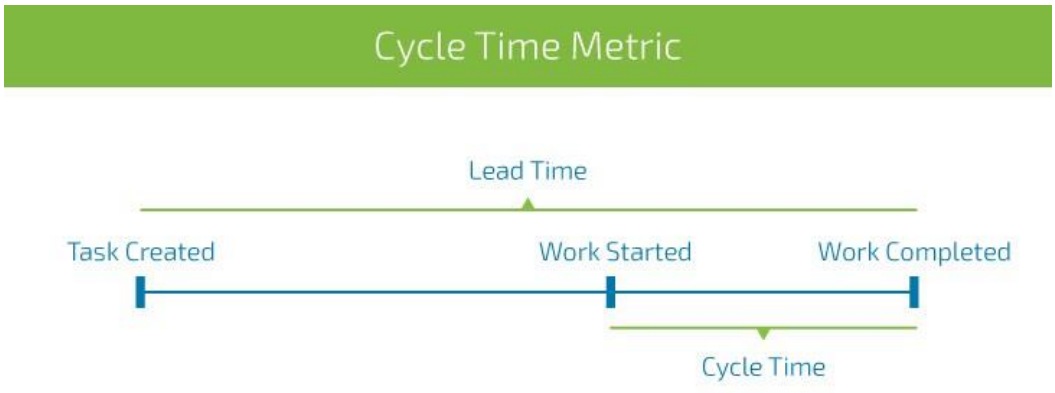
SonarQube is an open source platform used to measure and analyse the source code quality. It can help check that code written is healthy and free from bugs by continuously inspecting code, which can save a lot of time in a project and thus making it far more efficient. With the likes of using git within a project, SonarQube can connect directly to this to analyse branches of repos, code health and helps find and fix any issues. It can also auto issue assignments to various people so that there is a streamline process between finding a problem and resolving it. If a software development team were all using Git and SonarQube on a project then every little change is automatically analysed and reported upon, meaning that issues are raised before they are even found by a person. This streamlines the entire process of being able to measure software engineering, as issues can be easily and quickly assessed and fixed, then in theory all commits by a programmer are of quality rather than quantity, meaning that they are working far more efficiently with the benefits of these platforms.

Another type of platform that seems to be growing increasingly popular, is the use of health trackers such as Fitbits to measure a person health. Particularly in the era of a pandemic, health has become a common topic in the workplace. Often companies incentivise workers with free Fitbits to both help them keep active and to also analyse. It is well known that keeping healthy and fit is a major factor in mood and focus, which all lead to being far more productive. Fitbits could also give the power to managers to view a worker's data, such as daily steps, average heart rate and quality of sleep. A manager could therefore be able to recognise an issue an employee is going through before the employee even brings it up. Tracking a worker's wellbeing could be of use to consider other aspects if for some reason a

programmer is not delivering their part of the project, which other platforms don't offer. How effective this could be in measuring software engineering may not be fully known yet but keeping employees healthy is a major priority in today's world.

Algorithmic Approaches Available

Software product metrics focus on four main areas: Specification, Design, Coding and Testing. By analysing these software metrics, a manager can get a very good insight into the productivity of a software engineer. A popular algorithm used is cycle time, which relates to how much time is spent working on an issue. This could be broken down into separate data types such as bugs or feature requests. This can be quick a good measure of the efficiency of software engineers as it allows to analyse the whole processes, rather than just code written. As with any issue, there has to be some planning and organisation in place before diving into the coding solution. Often this approach can be a true reflection of the time taken from idea to release and can be compared to other issues of similar type. This metric also allows companies to estimate how fast new features or fixes can be delivered to users. It can help pin own the exact bottlenecks that may be affecting a software team's performance and set realistic expectations for the team.



Throughput is also another algorithm that can work very well in measuring a team's productivity. Throughput counts tasks, chores and bugs within a feature that is released. Throughput can give a better idea to the amount of work within a feature that had to be completed for it to be released. No two features will be of the exact same workload, so measuring the different tasks completed within each feature can give a more accurate representation of the productivity of software engineer. It gives a better idea of what their workload was for given time periods.

Any metric that claims lines of code as a data source has to be approached with caution. "Measuring programming progress by lines of code is like measuring aircraft building by weight" – Bill Gates. It can lead to bad practises, where code is copied and pasted, bulking out code and writing meaningless comments. Logical Lines of Code over time is a far more meaningful measure of someone's coding efficiency. For instance, a function to calculate a result could be wrote in 5 lines or 10 lines, but they both provide the same functionality. Measuring logical lines provides a far clearer understanding of somebody's work and allows for the avoidance of bad habits and cleaner code.

In general, all these calculations share the same base theory, inputs divided by outputs. This is the basic cost analysis form used in all of the algorithms. This creates a system where often times, analysis can only be done after product has been completed. Because of this, there can only ever be approximations of how long a task will take to be completed, it's a question of how accurate we can get these estimations.

Ethics

These types of measurements raise many concerns, which can resonate across many aspects of not just the production pipeline, but in software in general. As discussed, data is the most important factor in being able to gauge software engineering, but at what point does this data become damaging. Collecting any data can be invasive to both the project and the individual, to a point where every minute detail is analysed. The fear of having every aspect of your job being analysed can lead to fear and anxiety in a team, which can have damaging effects on the project and on an individual's stress levels. Many a time, when under pressure the best most useful thing is to take a step back and look at the bigger picture, often times taking an issue back can lead to the issue being resolved in a better manner. If an individual is struggling with an issue and knowing that big brother is gazing down upon them does not fix the issue, it only heightens it to the point where it can cause unnecessary damage to a project leading to productivity to be worse and allowing the project to stall. To what point is lots of data too much data? How lenient are these metrics in the sense of scrutinising a programmer's productivity? What are these metrics used for? Are they only used to calculate times to deliver a product or too keep an eye on employee behaviour?

The concern to a programmer is how invasive and critical are these metrics being taken and interpreted. These types of data collections can cause impacts on the company culture where groups of people work as individuals rather than a team to outperform their colleagues. Its not unknown that there have been issues raised over toxic workplace conditions in the design of these teach giant offices. There has been a serge in the culture of tech giants to have their employee lives orientated around the workplace, furthering the aspect of invasion of privacy and feeling constantly under watch.

There are clearly many unanswered ethical questions on the aspect of measuring software engineering these days, particularly with it only coming into the for front in recent times. How this will be in five or ten years, but until proper discussions are had on what these measurements are used for, they could cause a lot of strain in the development of products.

Conclusion

In conclusion, there are many different ways of measuring software engineering, but there are many problems that come with it too. How data is used to measure productivity has to be taken with care to accurately be able to measure the progress of an individual. With the correct implementation of this data and tools, the benefits can be reaped and lead to a far more efficient software engineer. However, its important to actually utilize this data in a meaningful manner, whereby the goal of it is not call out an individual's flaws, but rather to help improve them. Once, these cases are taken into consideration, then one can be able to correctly measure software engineering in the best possible way.

References:

https://www.atlassian.com/agile

https://community.uservoice.com/blog/the-pros-and-cons-of-agile-product-development/

https://library.ahima.org/doc?oid=57643#.X7vndWj7RhE

https://stackify.com/track-software-metrics/

https://medium.com/scope-ink/measuring-creativity-dfd1168d7c91

https://www.sonarqube.org/github-integration/

https://www.fitbit.com/content/assets/group-health/FitbitWellness_InfoSheet.pdf

https://core.ac.uk/download/pdf/77239902.pdf

https://www.jamasoftware.com/blog/the-9-best-metrics-to-measure-software-team-productivity/

https://www.gitclear.com/popular_software_engineering_metrics_and_how_they_are_gamed

https://blog.ndepend.com/how-measure-lines-code-lets-count-ways/

https://dzone.com/articles/how-to-use-software-productivity-metrics-the-right

https://www.sciencedirect.com/science/article/abs/pii/095058499290168O

http://www.umsl.edu/~sauterv/analysis/488_f02_papers/SoftwareProductivity.html

https://stackify.com/measuring-software-development-productivity/