Ciarán Coady
Student Number : 17326951

Measuring Software Engineering

Report Spec: To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, and overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Introduction

A common conception, or misconception, is that software engineering is not measurable, and engineer productivity likewise. For example, if an engineer spends 2 hours researching a given problem, and then creates an efficient and concise solution within another 1 hour. Is this engineer as productive in comparison to someone who may have thrown together a solution within an hour? With code that is hard to read and twice as long?

Examples like this really do beg the question, is it possible to measure engineer productivity, and by extension measure the effectiveness of the engineering process? I believe that it is very much measurable, although I question the fairness and accuracy of such measurements when it comes to decision making.

Consider using lines of code (LOC) as a measurement for productivity. There is some ambiguity when using this metric. Is a line something that ends with a carriage return? Does it include comments and whitespace? Or is it just logical lines of code; actual instructions? At a base level, using LOC as a measure productivity does not give any basis for analysis, there needs to be complementing information. For example, how many of these lines were new? How well does the code solve the posed problem etc. In the words of Bill Gates:

"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs".[1]

However, it does not mean that such measurements are useless, or that software engineering is unmeasurable. In this report I hope to analyse some of the current measurement techniques out there, what platforms are available to analyse these measurements and finally discuss the ethical concerns of such a process.

What is Software Engineering?

"An engineering discipline that is concerned with all aspects of software production." [31] Software engineering is not just the act of writing code, it encompasses the whole process of producing a software system. It is analogous to other industries like construction where it is not possible to just begin work on a building without following a process.

In software engineering we first must gather the requirements of the software system. This includes analysis of the various stakeholders and their needs for a system, designing the system with the requirements in mind. It must be ensured that the design meets the requirements and then consult the stakeholders again. Needless to say, the software engineering process becomes very complicated rather quickly. This is before any technical or implementation details are even considered. What I described above is the requirements gathering phase of the "waterfall" software engineering process but depending on what engineering process a team is following this might not be the first step at all. Maybe they are employing an agile technique with continuous integration.

Without a doubt, the software engineering process is complicated by nature, which would leave many to believe that the efficiency of a process and productivity are not measurable. Due to the range of different techniques available, this seems like a valid argument. Research shows that 75 percent of business and IT executives when surveyed admitted that their projects were either "always" or "usually" doomed right from the start.[30]

Why is it that so many projects fail and what can be done about it? Surely if executives know a project is going to fail, there should be some course of action to prevent it? From here, the measurement of engineering becomes interesting.

Why Measure?

What is the purpose of measuring performance, or in fact measuring any aspect of software engineering? Well that can be answered quite easily by looking at other industries. Let's take manufacturing as an example. In the manufacturing industry we can measure pretty much every aspect of production, the time it takes, progress down the assembly line, the number of people working on it and so on. From this quantitative data it can be determined how the process may be improved. If production is slow, add more people to the assembly line. If there are multiple employees doing the same job, compare them to each other and extract individual efficiency and performance metrics. This measurement and data can then be used when put with the end product to get all kinds of useful information, such as how more products can be produced, or possibly at what point along the assembly line is a flaw being introduced. In this way the entire process can be improved.

Similarly, in software engineering the end goal of measuring and tracking different quantities like commits or lines of code written, is to extract useful information about the quality of the current process. This information can also be used to improve such a process or possibly even scrapping that methodology altogether in place of a more promising one. This could also allow the prediction of the quality of a software product and time it might take for a product to be completed. Thus, allowing managers to give better estimates on delivery time of a working, quality product.

For project teams, this kind of data may be useful if they are evaluating a new development process, such as agile vs waterfall.[2][3]

For managers it boils down to a few major concerns: increasing return on investment, identify areas of improvement, manage workloads, reduce overtime and reduce costs.

It can also be used by management to identify if an employee needs help, motivation, or deserves a promotion opportunity. It can equally identify major contributors to the project, or even identify engineers who may be having a negative effect on progress.

By taking such measurements, we can hope to get an indication of software performance, quality of planning, measurement of productivity and the measurement of the performance of a specific development process.

Measurable data

I would like to address what I understand the measurement of the software engineering process to mean. I see the measurement of software engineering to be: A measurement of a developer’s contribution to a software system, getting it from its current condition to the desired end point. To make this even more concise, I am viewing this as an engineer’s individual productivity and how they impact the progression of a software system. The ideal here is to extract some data from engineers’ activities, and from that be able to reason about this person’s performance. From measuring an individual performance, we can then gain a clear view of the effectiveness of an engineering process. In my view, even a competent engineer or team of engineers will falter if following a poor production process.

To do this we first need to understand what exactly to collect in order to measure productivity and other metrics. When I say this, the word data comes to mind. Data is facts and statistics collected together for reference or analysis.[5]

There are two main types of data:

1. Quantitative Data – This is data that deals with quantities, values or numbers. Making them measurable.[24]
2. Qualitative Data – This is data that deals with quality, so they are descriptive rather than numerical in nature. This means it is not measurable. [24]

To paint a full picture of a software system, its contributors, working environment and so on, we must collect both quantitative data and qualitative data. Quantitative data can include number of commits, or lines changed. Qualitative data can include the impact of a developer on their peers, or the value someone adds to a project.

The collection of quantitative data is quite straight forward, for example we can gather most of the required information from a git repo or a messaging platform. However, qualitative data is much more difficult to acquire. To measure someone’s impact on a working environment, you would need to interview staff and co-workers to see what kind of an impact someone is making. Maybe they do not have huge contribution to a code base but are an impactful presence when it comes to pair programming. Maybe they boost team moral when deadlines are due. These are both pieces of data you simply cannot measure from looking at git logs.

Currently platforms available such as GitPrime will give you all kinds of information, e.g. “how well do we share knowledge in code reviews” or “What did engineering accomplish last week”. [26]

The real question then becomes can we extract the unmeasurable information by only collecting quantitative data? And should important decisions be determined solely on metrics and analytics of quantitative data?

Algorithmic and Heuristic approaches

An algorithm is a defined set of step-by-step procedures that provides the correct answer to a problem.[10] Using a good algorithm to solve a problem or make a decision, is useful as it yields the best possible answer every time. The upside to developing an algorithm is that it can be automated using computers and thus has the potential to be much faster and more accurate than other decision-making processes.

The caveat here, is that for an algorithm to work effectively it must have all the data that is necessary to solve the problem. What happens when all the data is not available? How do you qualify what data is needed to make a fair and accurate decision?

It is fair to say that an algorithmic approach may be a viable solution, if we can boil down productivity into several steps that can be satisfied, and we can measure all the required data for such an algorithm to come to a conclusion. Unfortunately, it is not that simple due to the inherent complexity of the software engineering process. Often managers will make snap decisions with very little information, which may not be possible to replicate algorithmically. A manager can do this because they have built up knowledge from experience, part of this knowledge is known as heuristics.

Heuristics are simple strategies to form judgments and make decisions by focusing on the most relevant aspects of a complex problem.[8] Heuristics are quite useful when it comes to making quick decisions; there is no need to analyse all possible options as you would in an algorithmic approach. An example of this would be choosing what clothes you might wear today. Using heuristics this decision becomes a matter of seconds, if you were to algorithmically work out what clothes to wear based on your entire wardrobe, weather conditions, what matches and other criteria, you may never be dressed at all by the end of the day.

Where heuristics becomes useful in the setting of software engineering productivity lies with management. Let’s take for example an engineer who has been working well for several years, always makes good decisions and has a reputation to be a team player. A manager would have formed a heuristic about this engineer, about their work ethic, ability to adapt and impact on a team. This data would not be obtained by an algorithmic approach in the same fashion.

Now let's say this engineer is dealing with some family issues which spans over several months, if being managed using qualitative data collected by a manager, the manager would understand the situation and using heuristics may give the engineer the benefit of the doubt based on past positive performance. However, would an algorithmic approach give such allowances? If an engineer's monthly performance for several periods is subpar, will an algorithm provide any sympathy, or just recommend that this person is no longer effective at their job and should be replaced?

Humans also use heuristics in situations where we don't have all the required information to make an informed decision. We can't use an algorithm if all the required information to come to a solution is not available. So, is all the information needed to make an informed decision on an engineer's performance captured in measured data, or do we need a certain element of heuristics and qualitative data?

By relying on algorithms rather than human judgement we can eliminate some prejudice based on heuristics that a manager may have formed based on experience with a certain employee, which could lead to a fairer judgement of a situation.

In my own opinion, I believe that these algorithmic approaches make a good approximation of an engineer's performance. However, I do not see them as a sole method of measuring performance. With the current available technology, I do not think it is possible to obtain all the necessary data to make a final decision on someone's "performance." I do see these algorithmic approaches as tools to help engineers evaluate their own performance and as an aid for managers to gain an understanding and insights into a project. I believe that the qualitative data that is missed by such algorithmic approaches is vital, and thus needs to be included when making any decisions.

Computational Intelligence

Computational Intelligence (CI) is the theory, design, application and development of biologically and linguistically motivated computational paradigms.[6] Essentially computational intelligence is trying to mimic the capability of the human mind using computational techniques. It is a constantly evolving field with many new approaches emerging on a regular basis. A few examples of these approaches include: Neural Networks and Fuzzy Systems.

Neural Networks

The term "Neural Network" is used to describe a massively parallel distributed network that tries to mimic the functionality of neurons in the human brain. As a result of this neural networks have that ability to "learn" and generalise from examples.

Fuzzy Systems

A "Fuzzy System" models' linguistic imprecision and solves uncertain problems based on a generalisation of traditional logic. This allows for approximate reasoning and thus emulate heuristics like I spoke about previously.

These computational techniques have been employed in several different fields, from video games to self-driving cars. The success of such methods in other fields has promoted its use in situations where both computational power, e.g. with a large data set, but also human-like intelligence is necessary to garner some meaningful output.

Both approaches I have mentioned involve clustering groups of data based on a training set of data, then make decisions based on this information. This begs the question how intelligent are these approaches in reality? I personally think these systems are only as effective as the training data they get provided with. If these systems are implemented and then trained poorly, what is the validity of the results it returns, and should we trust them? This is a question I have still to answer for myself, but possibly as systems mature and are exposed to a higher volume of data, they could indeed produce results similar, if not identical, to that of human judgement. For me this remains to be seen.

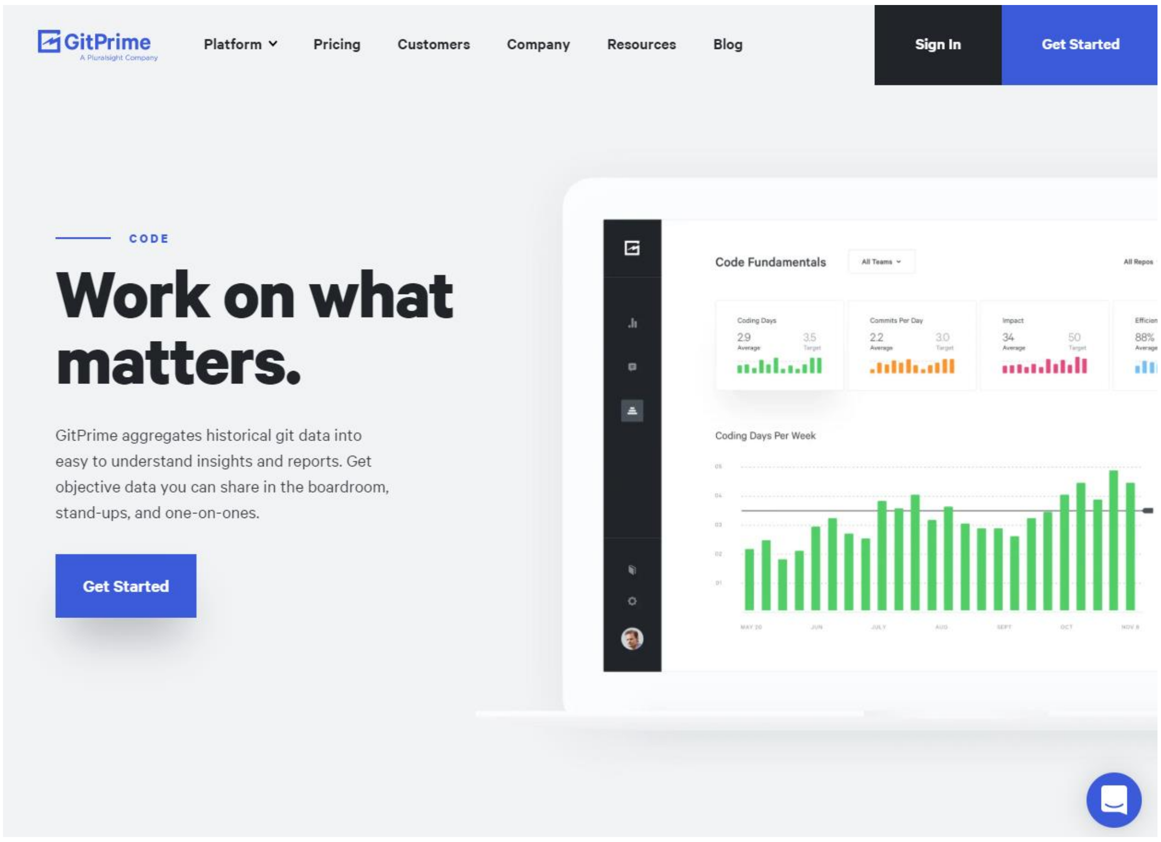
Computational Platforms

After gathering quantitative data in the hopes of measuring software engineering, we must try to give some context to the data collected and extract some meaningful insights from it. This is the role of computational platforms. Platforms like GitPrime try to extract some sort of meaning and useful metrics from the collection of measured data. They do this using a combination of algorithms and computational intelligence.

As I mentioned previously, algorithmic approaches are quite appealing as they can be automated and performed by computers. This means that large quantities of data can be collected and processed, resulting in some sort of conclusion. When coupled with computational intelligence, which essentially is a very clever way of grouping large quantities of data and pattern matching, these platforms can provide simple metrics, numbers and percentages that represent quantities, such as performance.

GitPrime

One such computational platform is GitPrime. GitPrime mines data from git and using techniques as described previously to provide data like “Identify areas to give concrete feedback” or “How much of your team's burn is going to refactoring old code?”



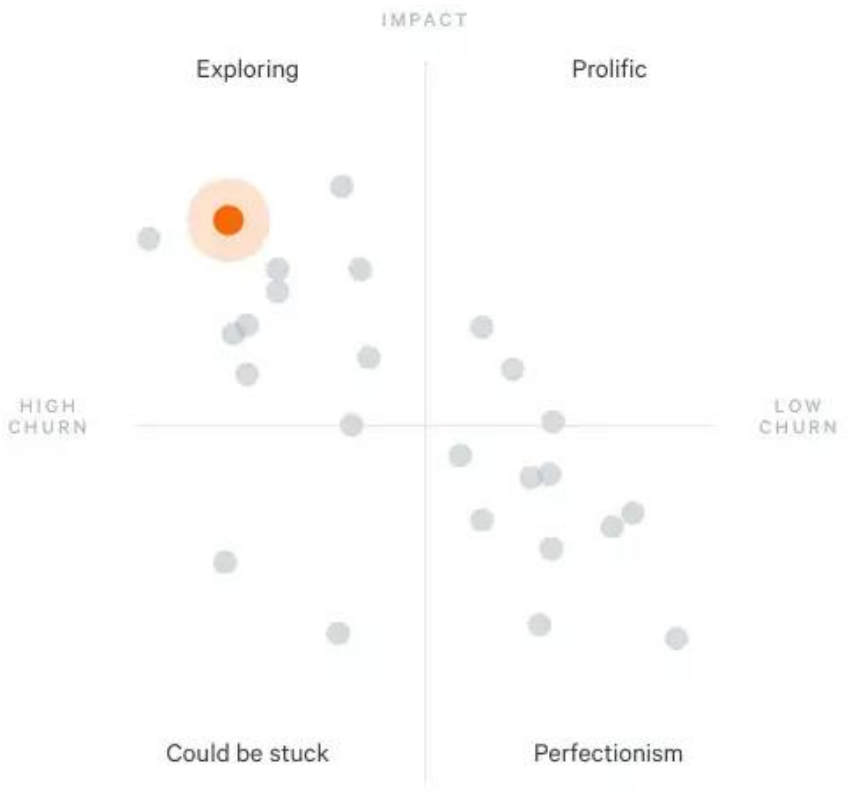
CODE

Work on what matters.

GitPrime aggregates historical git data into easy to understand insights and reports. Get objective data you can share in the boardroom, stand-ups, and one-on-ones.

Get Started

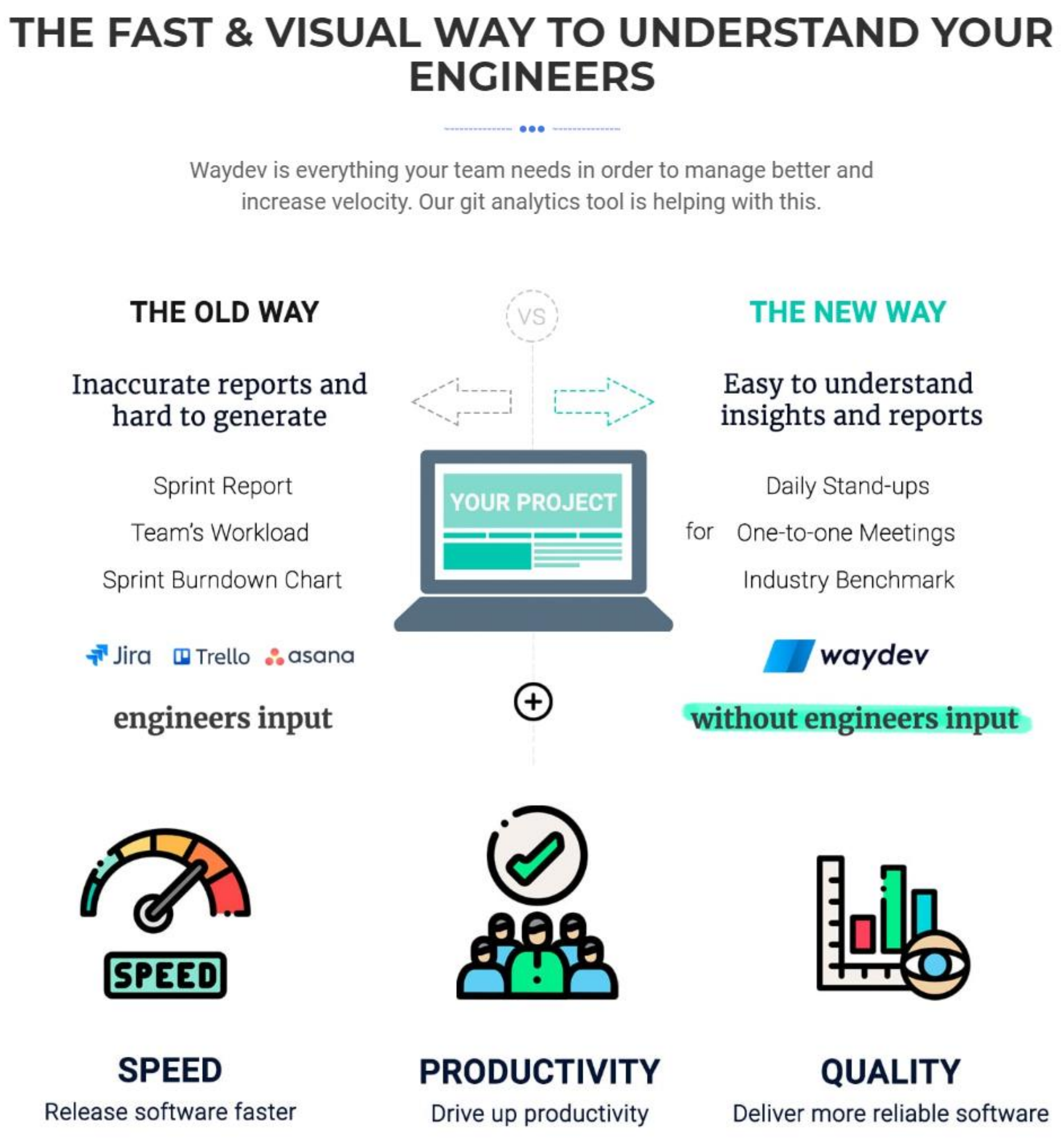
Above: An example visualisation dashboard from the GitPrime website, displaying how different metrics are displayed.



Above: An example visualisation of code churn from GitPrime.

WayDev

WayDev is another computational platform which provides analytics of developer data. Their key focuses are similar to that of GitPrime.



THE FAST & VISUAL WAY TO UNDERSTAND YOUR ENGINEERS

Waydev is everything your team needs in order to manage better and increase velocity. Our git analytics tool is helping with this.

THE OLD WAY

Inaccurate reports and hard to generate

Sprint Report
Team's Workload
Sprint Burndown Chart

Jira Trello asana

engineers input

VS

YOUR PROJECT

THE NEW WAY

Easy to understand insights and reports

for Daily Stand-ups
One-to-one Meetings
Industry Benchmark

waydev

without engineers input

SPEED
Release software faster

PRODUCTIVITY
Drive up productivity

QUALITY
Deliver more reliable software

These platforms are extremely attractive to project managers as the information they can view not only will make their job less ambiguous but also helps them have concrete information about their employees work rate, how they can motivate them, see who is working well and others who are not pulling their weight etc.

However, this data should not be taken as gospel. These metrics are by no means fair and representative of a person's contributions to a project. In fact, sometimes analytical data can be skewed if the subjects being analysed know the metrics by which they are being measured. For example, if we decide to use lines of code as a metric for productivity, then we will reward employees who write many lines of code and penalise those who do not. This will have the negative effect of promoting long convoluted solutions that could be solved in a much more concise manner using less lines of code. This could result in code that is harder to maintain and understand, and reward arguably poor solutions to problems.

I think that the computational platforms available today do not provide a “solution” to the measurement of software engineering. This is my opinion as I feel they do not gather enough information to obtain a clear overview of a development environment, as most platforms seem to mine their data from only one or two sources such as version control or a messaging platform. However, I do see them as an aid in reducing any prejudice or preconceived ideas we may have about a person or development process. These computational platforms act as a guide for management, possibly highlighting areas of production that need to be changed or investigated. For engineers these insights can be useful to instil a competitive element within teams, and indicate the shortcomings of their processes, both things which can increase productivity and have a net positive effect on production.

Ethics

With such processing ability and availability of data it poses the question: Should we measure engineers’ activities?

In my opinion, I would say we should measure engineers’ activities. In many other professions every action taken by workers is measured. Or in a more general sense, most things in everyday life are measured. With the increase in computing power and advancement in various computational techniques, it is not surprising that every aspect of life is being measured. Nobody wants to feel inferior when compared to a co-worker, but the pros of measurement fair outweigh any cons, in my opinion. Using the feedback data analytics can provide, there is the potential to not only improve the software engineering process, but also allow engineers to identify their own strengths and weaknesses, thus giving the opportunity for self-improvement.

Is it fair to make life changing decisions such as firing someone based on measured data?

As mentioned above, I feel that with the current technology it is not sufficient for decision-making to consist solely of these computational platforms, as I believe that they lack in subtle information that can prove to be vital in decision-making. But when used in collaboration with other information that may have been collected through techniques such as interviewing, or heuristics a manager may have accumulated over time, I feel that the results provided by such computational platforms are a useful insight and provide the basis for fair judgement.

However, when using such third-party platforms, one must be careful of the GDPR implications, as personal data of engineers must be provided. The engineers must be made aware and consent to the processing of their personal data.

Conclusion

In conclusion, I believe that the software engineering process can be measured and assessed in terms of measurable data, but I do not agree with it being the sole indicator of the performance of engineers or the engineering process as a whole. From my research, I think that the current computational platforms available do not accurately reflect the subtleties of a teamworking environment and therefore, should not be used in isolation.

1. http://wiki.c2.com/?LinesOfCode
2. https://agilemanifesto.org/
3. https://en.wikipedia.org/wiki/Waterfall_model
4. https://en.wikipedia.org/wiki/Software_engineering
5. https://www.lexico.com/en/definition/data
6. https://cis.ieee.org/about/what-is-ci
7. https://www.youtube.com/watch?v=ReFqFPJHLhA
8. https://en.wikipedia.org/wiki/Heuristics_in_judgment_and_decision-making
9. https://en.wikipedia.org/wiki/Algorithm
10. https://www.verywellmind.com/what-is-an-algorithm-2794807
11. http://www.citeulike.org/group/3370/article/12458067
12. https://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf
13. https://www.johndcook.com/blog/2011/01/10/some-programmers-really-are-10x-more-productive/
14. https://www.youtube.com/user/SoftwareEngBook/videos
15. https://www.youtube.com/watch?v=0KmimDq4cSU
16. http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf
17. http://patentimages.storage.googleapis.com/pdfs/US20130275187.pdf
18. https://www.geeksforgeeks.org/software-measurement-and-metrics/
19. https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451
20. https://en.wikipedia.org/wiki/Artifact_(software_development)
21. https://www.scrum.org/resources/what-is-scrum
22. https://help.gitprime.com/general/what-is-gitprime-exactly
23. https://stackify.com/track-software-metrics/
24. https://www.cleverism.com/qualitative-and-quantitative-data-collection-methods/
25. https://nortal.com/blog/the-myth-of-developer-productivity/
26. https://help.gitprime.com/general/what-is-gitprime-exactly
27. https://semmle.com/assets/papers/measuring-software-development.pdf
28. https://ieeetv.ieee.org/conference-highlights/artificial_neural_networks_intro?rf=channels|56|Selected_Videos&
29. https://www.youtube.com/watch?v=J_Q5X0nTmrA
30. https://www.entrepreneur.com/article/329019
31. https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited/

32. https://pdfs.semanticscholar.org/4935/410e22756a262e41614747e0d94291cf860b.pdf