MEASURING THE PRODUCTIVITY OF A SOFTWARE ENGINEER

Conor Gilmartin

16321855 CS3012

MEASURING SOFTWARE ENGINEERING

INTRODUCTION

A major question has arisen for many organisations and companies around the world. How do you measure the productivity of a software engineer or a software development team? It is a much debated topic and it is often controversial. Many people think that software is simply too complex to measure. There is no debate about whether there is data that can be measured from the code of a software engineer. For instance, the amount of code an engineer produces, the amount of bugs they introduce and how many they fix can all easily be measured. However, the debate is whether this analysis can result in meaningful conclusions.

The most common definition of productivity is that of a 'black-box'. It is the total output divided by the total input. Despite being easy to define it is very difficult to measure. As will be discussed, there are many different metrics that you can obtain from the output (the software), but there are no metrics or combination of metrics that will give you a clear value of the software. Similarly it is also difficult to measure the input. It can no longer be counted by labour hours. Money spent, equipment used and worker training are other factors that must be taken into consideration.

QUALITY OF SOFTWARE

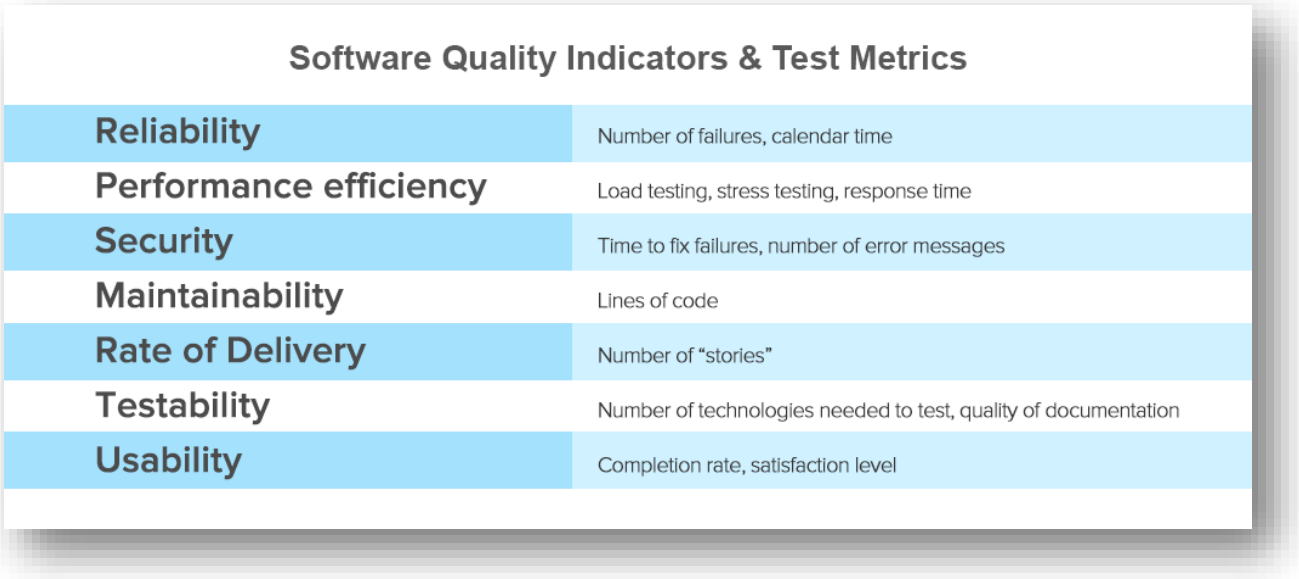
The quality of a piece of software generally comes down to two different aspects:

1. Functional quality – refers to how well the software conforms to the given design. It complies with the specifications of the functional requirements.
2. Structural quality – refers to how the software system meets the non-functional requirements. This can refer to the efficiency or maintainability of the system.

There are several aspects that contribute to the quality of a piece of software but they generally come under the above factors. The CISQ software quality model defines four important indicators of software quality:

- Reliability
- Performance efficiency
- Security
- Maintainability

Reliability refers to the risk of the software failing and losing stability when exposed to unexpected conditions. The software architecture and source code design contribute to the performance efficiency. This affects the program's scalability, customer satisfaction and response times. The security aspect of the program refers to how well the software application protects the sensitive and personal information within the program. The quantity and severity of vulnerabilities found in a software system are indicators of its security level. Poor coding and architectural weaknesses often lead to software vulnerabilities. The maintainability of a software system is the ease with which you can change the software and adapt it for other purposes. It can also refer to hoe easy it is to transfer from one development team to another. Compliance with software architectural rules and use of consistent coding across the application combine to make software maintainable.



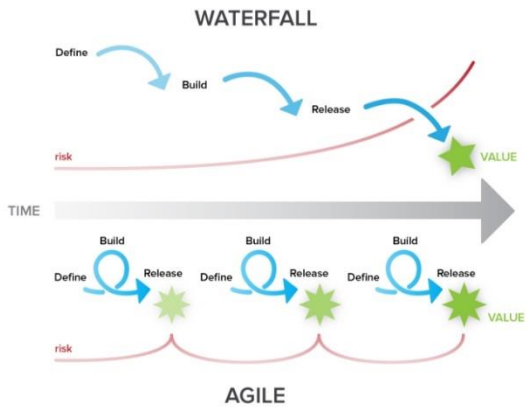
FAILED ATTEMPTS

There have been many attempts to solve the problem of measuring a software engineer's productivity. Some of the methods that were used in the beginning were simple to implement but in the end were not effective in obtaining any useful information. These include measuring the lines of code (LOC) of a worker. This obviously is not a hood idea as some of the most elegant solutions to complex problems are the most succinct and only require a small amount of code. By writing more lines of code, it can often make the solution much more inefficient. Also, human nature will dictate that when you are being measured based on lines of code, you will just write more lines even if it does not improve the solution. Therefore, not only is it a poor way to measure to understand the productivity of a developer, it actually entices them to do worse work.

> "Measuring programming progress by lines of code is like measuring aircraft building progress by weight."
>
> Bill Gates

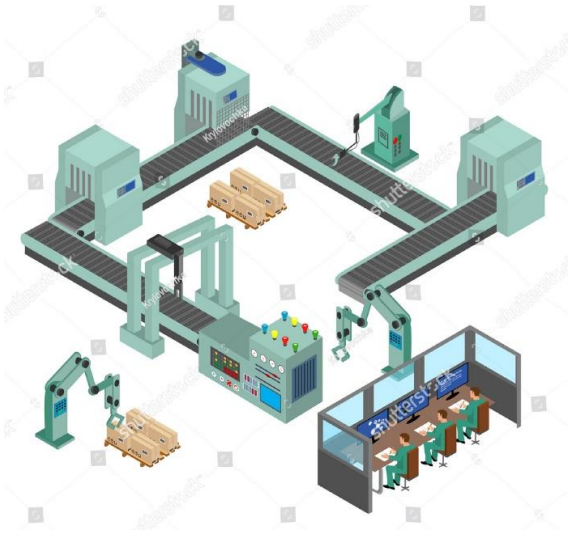
Measuring the productivity of a software developer by the amount of hours they work is obviously not a good idea. It does not take into account the amount of work being done or the quality of the work. Furthermore, workers are then encouraged to work more hours which can actually lead to less productivity. According to a study by Stanford University, employees that work 60 hours per week often accomplish less than employees who only work 40. This is mainly due to an increase in errors or oversights that later need to be corrected. This is especially relevant to software engineers as small mistakes can lead to massive errors that are difficult to fix.

Much of these methods of measuring software development (and others that are still in use) are oriented around the waterfall method. Modern software development however is mainly based on the agile method. Software development is about continuously developing, delivering, integrating, and improving software instead of delivering the final product all at once.



DIFFICULTIES

The main difficulty in the measurement of software development is due to the nature of software. Not only is it very complex, it is also unique. Every piece of software is unique. It is not like manufacturing where the same product is made over and over. You are not able to just measure for defects and reject the products that have them and release the ones that don't. You do not know what the defect might be. The defect in a piece of software could be to do with bugs in the code, or its efficiency. A lot of the time, your software will be used in a variety of ways, most of which you had not envisaged. As a result, it is extremely difficult to tell how defective your software is.



The Snowflake Metaphor

- Each component is a snowflake – You will not need two different pieces of software to have the same functionality. There is no one metric that will tell you when a piece of software is correct.
- Each person is a snowflake – No two people are alike. Each software engineer has different skills, traits, backgrounds and motivations. Each person's performance can only be compared to their past work and not to other individuals.
- Each team is a snowflake – A team is not just a group of individuals. They work together by communicating and cooperating with each other. You cannot compare teams to one another as they are composed of different people.
- Each project is a snowflake – If two projects were the same, there would be no point in doing the second one. Also, each project will have a different value to a business so must be treated differently.

It is easy to measure various things about software development, including things about the actual software, the individuals and the team. However these metrics are only valuable when comparing them to their own past history. In order for a company to receive the benefits of tracking the work its software engineers to gauge their productivity, they must be careful to choose the correct metrics. The metrics chosen should be designed to answer business questions. For example, if your software crashes one percent of the time and recovers almost instantly, not losing any critical information, then this crash rate may be acceptable. However, if this crash rate is on a system that runs 100'000 times a day and each crash loses a 100 euro sale and it costs 50 euro to fix, then improving the crash rate would be a priority.

METRICS

Agile Process Metrics

There are various metrics that companies can track in order to get an idea of the productivity of a software engineer or a team of developers. For agile processes, the main metrics are lead time, cycle time, team velocity and open/close rates. Although they do not measure the quality of the software, they can still be useful.

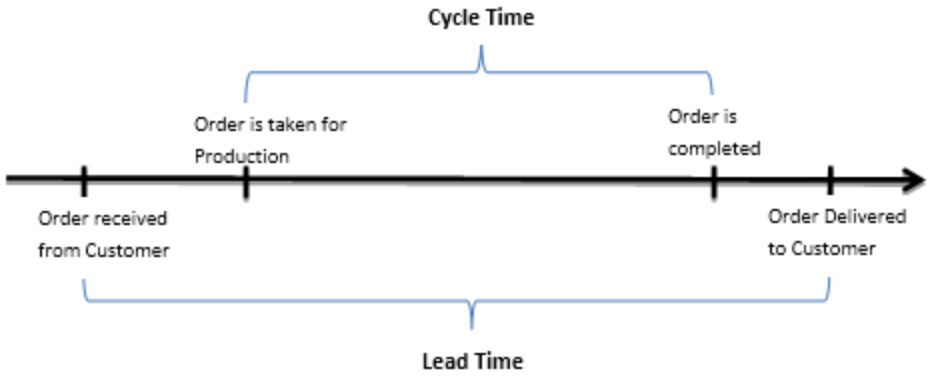
Lead time – the amount of time it takes you to go from an idea to the final piece of software.

Cycle time – how long it takes you to alter the software and deliver the new system into production.

Team velocity – refers to the amount of code typically completes in a certain amount of time.

Open/close rates – how many software issues are reported and closed within a certain amount of time.

Code Churn – represents the number of lines of code that were modified, added or deleted in a specific period of time.



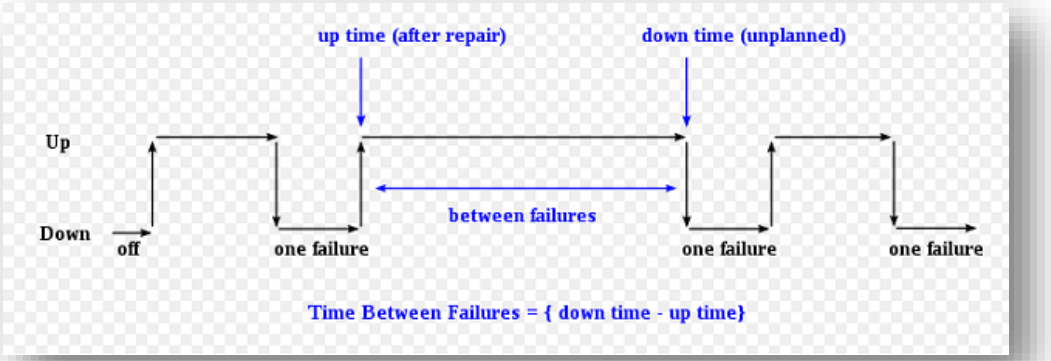
While these metrics are important, it is not necessarily cause for concern if some or all of them are out of the expected range. There should be communication with the individual or team in order to get the whole story to see if there is a problem. For example, a high open rate and a low close rate may mean that fixing production issues is not as high a priority as adding new features at that point in time.

Production Bug Analytics

There are a number of ways to measure the performance of a software system. In an ideal world, our software would never fail but this is highly improbable. When it does fail, it would

ideally retain critical information and recover quickly. This can be difficult to achieve and metrics can aid you in attempting to do this.

- **Mean time between failures** (MTBF)
- **Mean time to recover/repair** (MTTR)
- **Application crash Rate** – related to MTBF and MTTR, how many times an application fails divided by how many times it was used.

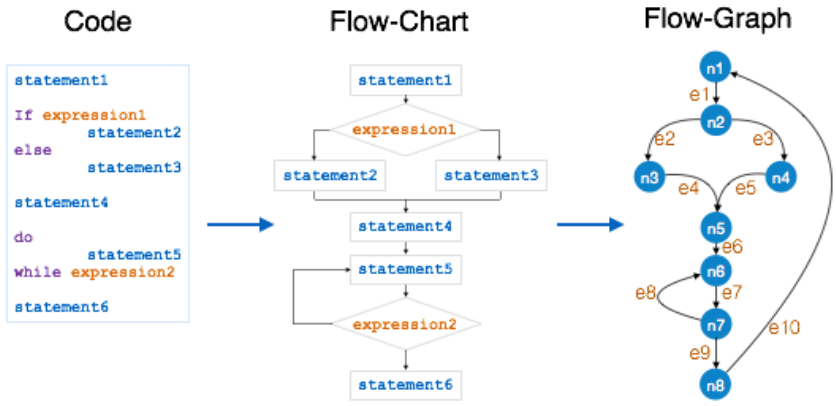


None of these metrics tells you about the actual error or the effect of it on the system or the user. Still, the smaller the numbers, the better. There are more detailed measurements than MTBF and MTTR which are based on the individual software system or application.

ALGORITHMS USED

Cyclomatic Complexity

This algorithm is used to quantify the complexity of a program. It was developed by Thomas J. McCabe, Sr. in 1976. It is computed using the control flow graph of the program with different sections of code being represented by nodes. Each node refers corresponds to indivisible groups of commands in the program. A directed edge connects to node if the second command is executed directly after the first.



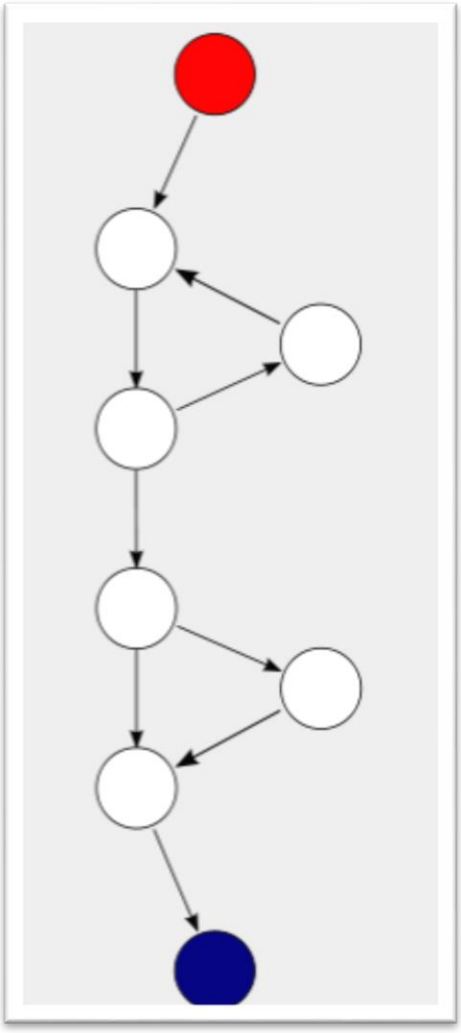
The cyclomatic complexity of a program is the number of linearly independent paths within it. For example, if the code did not contain any conditional statements (if, while, etc.) the complexity would be one. If there was one 'if' statement in the code, there would be two paths through the code. One where the statement evaluates to true and another where it is false. Therefore the cyclomatic complexity would be two.

M = E – N + 2P

Where:

- E = the number of edges of the graph.
- N = the number of nodes of the graph.
- P = the number of connected components.

The control flow graph to the right represents a program. Execution begins at the red node. It then enters a loop. On exiting the loop, there is then a conditional statement where one of two paths can be taken. Finally, the program exits at the blue node. In this case, the graph has nine edges, eight nodes and one connected component. Thus, the cyclomtic complexity of the program is given by 9 – 8 + (2*1) which is equal to three.



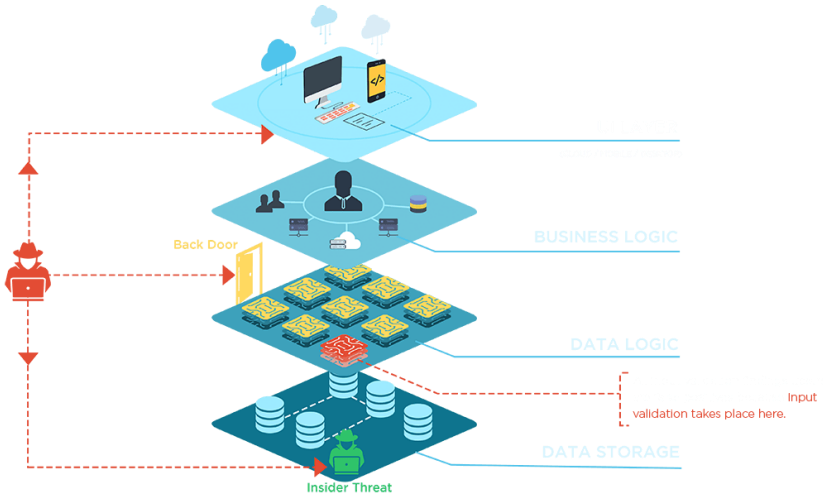
If there is only one program (or subroutine or method) then the number of connected components, P, is always one. However, cyclomatic complexity may be applied to many different programs or subroutines at the same time. In these cases, P will be equal to the number of programs or subroutines in question.

McCabe showed that the cyclomatic complexity of a program with only one entry point and one exit point is equal to the amount of decision points in the program plus one. A decision point is an if statement or a conditional loop. In this case, if there are compound predicates for the decision (if cond1 and cond2 then), they should be counted based on the amount of conditions involved. This is because this is how they are viewed at the lower machine-level instructions.

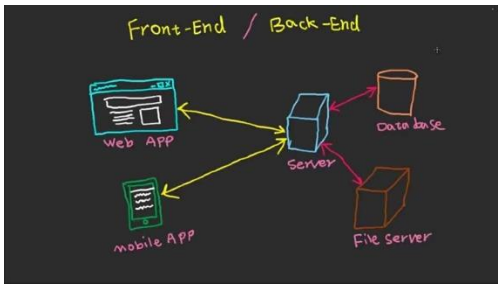
COMPUTATIONAL PLATFORMS

CAST AIP

There are various platforms available that will measure software development. One of these is CAST Application Intelligence Platform (AIP) which is an automated solution for completing the software measurement process. It is capable of assessing multiple languages within a complex infrastructure and it is enterprise driven. It is able to look at an application and break it down to its source code level. Then it can assess the application based on its quality, security, productivity and additional factors concerning the business goals it is supposed to accomplish. The automated process ensures the results are accurate for the given goals, and they are repeatable. The base score it produces acts as a continuous measurement of every aspect of the application.



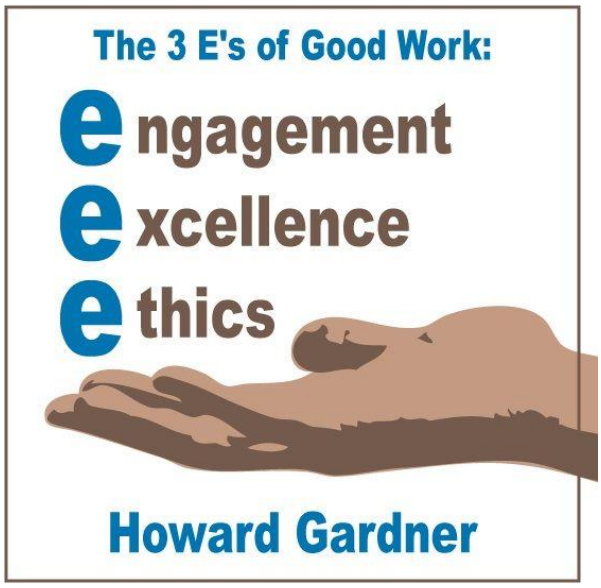
This platform is able to analyze complex, multi-layer applications and thus needs an understanding of the numerous technologies and frameworks used. It needs to be able to distinguish between the mechanisms used in procedural code like that found in C and those used in object oriented languages such as Java or C#. This requires more than just syntactic analysis. For example, when dealing with constructs such as call-backs in C or virtual calls in object oriented programming, the flow of data across the different programming elements must also be understood as if the program were actually being executed. The biggest challenge is to combine the understanding of each technical layer into an overall understanding of the complete system. How does the



Javascript front-end talk to the business logic in C#? How does the C# communicate with the Cobol program? You need a deep understanding of the entire system to answer these questions. Simply aggregating the quality of each component would be quicker, but those analytics wouldn't reflect the true value of the system and architectural defects wouldn't be found. To get a real insight into the value of the overall application, code analyzers must speak to each other and understand each other.

ETHICS

The Good Project defined three elements that are necessary to achieve good work. These are Excellence, Engagement and Ethics. According to them, good work is only possible if the three of these strands are in unison. Software engineers and all workers are now being measured for their productivity. Often workers are set goals and tasks which are tracked to see how quickly they are achieved. This is done in an effort to increase enthusiasm and efficiency. In the context of *The Good Project's* three strands of god work, measurement of worker productivity attempts to increase the Excellence and the Engagement of workers. However Ethics is not a part of the design of these measurement techniques. Ethics is quickly forgotten by the employees when they are judged solely based by their ability to deliver on a particular measurement. For example, if a software developer, stops to think about the ethical considerations of their software, it will likely come at the expense of their productivity. Even though the employee would be doing a better job by considering the ethics involved, the metrics will show that this worker is less productive. This will encourage workers to give little or no thought to the ethics involved with the software they develop.



One problem with measuring the productivity of a software engineer is that once that worker knows what is being measured, they will take actions, intentionally or not, to improve that metric. They will not necessarily improve their overall production or quality but they will probably find some way to improve on the metric. One solution to this problem involves not telling the workers of what is actually being measured. This is where there is a question of

ethics. It is not fair to hold an employee accountable for an arbitrary metric that very likely not reflect their actual work.

This leads onto the next ethical dilemma relating to measuring the productivity of a software engineer. The one thing that is generally agreed upon about measuring a software engineer's productivity is that it is extremely difficult. There is no clear metrics that will tell you how productive somebody is. Is it then okay to judge their work based off these inexact metrics? These metrics cannot reflect many things regarding the work of a software engineer. For example, a worker could be helping another employee to understand a piece of software or assist them in finding a solution to a problem. This will not be reflected in the metrics that measure his productivity. It is not ethical to assess and potentially dismiss employees that can only measure part of their value to the company.

CONCLUSION

Measuring the productivity of your workers is a goal for every employer. It helps them track their employees based on their performance and also helps to increase their productivity. You can identify areas of improvement. It can also determine the quality of the product and can be used to predict how long it will take to develop later pieces of software and how much it will cost. Although there are many different aspects of software development that can be measured, it is difficult to know which ones to choose to get a clear idea the development of the software system. The metrics chosen should be linked to business goals. If you are aiming to add a lot of new functionality quickly, the number of bugs reported should not be of high importance. Also, the metrics should be tracked to get an idea of the productivity of a software engineer or development team but you should speak to the individual or team to get the full picture. Employers must also ask themselves if the metrics they are tracking are ethical. Is it ethical to actually track them? Do they encourage your employees to think less about the ethical concerns of the systems they create? Overall, you can measure almost anything, but you can't pay attention to everything. You should measure only what matters now.

BIBILIOGRAPHY

"Measuring Work and Productivity: What Role Does Ethics Play?", Daniel Mucinskas, http://thegoodproject.org, March 31, 2015

https://www.castsoftware.com

"We Can't Measure Anything in Software Development", John Sonmez, https://simpleprogrammer.com, February 11, 2013

"What Are Software Metrics and How Can You Track Them?", https://stackify.com, September 16, 2017

"Development Leaders Reveal the Best Metrics for Measuring Software Development Productivity", https://stackify.com, April 18, 2017

"Why metrics don't matter in software development (unless you pair them with business goals)", https://techbeacon.com

"9 metrics that can make a difference to today's software development teams", https://techbeacon.com

"3 Key Metrics to Measure Developer Productivity", Gwen Schlefer, https://medium.com, September 28, 2017

"How to Measure Developer Productivity", https://www.7pace.com, March 20, 2018

"The IT Productivity Paradox", Ethan Dreyfuss, Andrew Gadson, Tyler Riding, Arthur Wang, https://cs.stanford.edu