[1]

MEASURING THE SOFTWARE ENGINEERING PROCESS

17332568

Claire McNamara | Software Engineering | 12/11/2019

[1] https://www.r-ccs.riken.jp/en/k-computer/about/

TABLE OF CONTENTS

INTRODUCTION

Why are we so keen on measuring the software engineering process in the first place? According to the 1933 report "Establishing a Software Measurement Process" by Donald R. McAndrews, the "primary purpose of measurement is to provide insight into software processes and products so that an organization is better able to make decisions and manage the achievement of goals". It is in a company's best interest to have a measurable system of working out its' overall productivity level in order to figure out the areas in which it could improve. Why is it so difficult to achieve though? Because "getting more done isn't the same as being productive"[2]. There is simply no one definitive way to measure the productivity of an individual no matter what line of work they are in let alone an individual's productivity in the context of software. Software engineering is an interesting field. It combines clear logic with the fuzziness of the world of creativity and problem solving. There is no one road to take that will carry an engineer toward a solution. Following on from this, as computers become more and more powerful, their capacity to take on jobs previously carried out by software engineers increases. This in turn impacts how productive we view engineers in comparison.

In this essay, I am going to explore the ways in which software engineering is being measured at the moment and why I agree with Martin Fowlers' assessment of the situation

"This is somewhere I think we have to admit to our ignorance."

MEASURABLE DATA

What is measurable data? Measurable means it can be quantified and data is facts and statistics collected together for reference or analysis[3]. In terms of software engineering this is a very wide topic. This is because the term "software engineer" encompasses many different aspects of computing. From proof of concept, overall design, management, implementation, testing, maintenance, etc. there are any number of ways progress could be measured and all of them very different depending on what area the software engineer is in. It is also necessary to remember that, in general, the practice of software engineering is carried out by a human which in itself brings countless more metrics.

Measuring the Productivity of the Individual Software Engineer

An analogy I like to think of is one of the lumberjack. At first glance, it may seem trivial in terms of how to measure the effectiveness of a lumberjack as their job goal is one that is quite straightforward – fell as many trees as possible. However, once one begins to think about it more deeply it is evident that this is not a completely accurate metric by which to go by. Many different factors can affect the amount of trees a lumberjack can fell. External contributors such as terrain, the weather, the species of tree, and the tools available for

[2] https://www.7pace.com/blog/how-to-measure-developer-productivity
[3] https://www.lexico.com/en/definition/data

them to use. Then also internal ones like the experience of said lumberjack, their health, their mood, etc. Just from a surface level overview we have already identified seven other measurable variables that impact the work of the lumberjack and this is for a job in which the end goal is clear. In software engineering, the final result is not so clear cut. Sure, the outcome tends to revolve around having working code that does the job it is supposed to do however, within that, there are countless other components that need to be taken into account. Things like the quality of the code, its readability, the complexity of the job it is carrying out, how well it integrates with existing code (if existing code exists) to say nothing of the process of coming up with the code solution in the first place.

Code Line Count

First and foremost, let us look at what we can definitively measure. Like in the case of the lumberjack, there is the stereotypical way of measuring the productivity of a software engineer which is by measuring them via the amount of lines of code they have written. And, if you ignore the many of the obvious flaws, it is one way to quantify how valuable an engineer is to a company. However, what happens if the software engineer in question is involved in maintaining a code base rather than creating new features. Their line-count would be minimal in comparison to someone who spent their time developing and writing new code; therefore one would be seen as less valuable than the other and yet their individual productivity levels may be equivalent. Not only that, there are differences in the line count of code that gives the same outcome between programming languages. A more verbose language such as Java will yield many more lines than a more concise one such as Python.

Money

Another clear cut measurement is money. For instance, engineer A writes code that brings the company in $10,000 in a year and engineer B writes code that only brings the in $2,000 in that same time period. It is very easy to say that employee A is the one of more value and yes, if one only looks at the current facts, that is does seem to be the case. However, while the software developed by A gives the company an immediate intake of five times that of B, the software developed by B in two years' time accumulates a total of intake $8,000 and due to the cost of maintaining A's software, the company has lost $2,000 on it. Now how do we determine which one was the more productive? Both have technically brought in the same amount of money into the company however, in spite of the fact that A has brought in a large amount of money immediately, as a result of the cost of maintaining said software, in the long run it ends up costing the company money. B's software appears less profitable at first however, thanks to the how easy to maintain the code is, it costs less and therefore ends up bringing in the same amount of money as A in the long run. Knowing this, it is now easy to say that clearly B was the more productive as in spite of the profit in total being the same. A's software will continue to lose money for the company while, due to the high standard of B's code, it will continue to bring profit. As demonstrated, relying solely on money can also be a skewed way of measuring the productivity of employees. Combining code quality with the profit made would yield a less biased result however, how can one measure code quality? The short answer to that is, subjectively. In this scenario, maintainability clearly is the major difference between A's and B's code and that has had a massive impact on its' profitability for the company. However, this is not always the case.

Hours Spent Working - "Quality over quantity"

What do the amount of hours spent on a particular project mean? Is it the case to attempt to spend the minimum amount of hours on a project as possible? And then if so, what does that mean for the quality of the code being written? As with the section above, a balance needs to be struck between the amount of time spent on a project and the end quality of said project. There is no point spending five weeks on a project to bring it up to the highest quality when 80% would have satisfied the requirements and could been achieved in two weeks. Equally, it is ineffective to spend days on a project and only bring it to 30% of its' required quality and then spend far more time and money attempting to "fix it up" when it inevitably fails. Alongside this, depending primarily on the skill level of the engineer, the amount of hours spent to produce quality code will vary. Someone with one years' experience in Java will spend more time writing a piece of code in Java over someone with ten years' experience. Therefore, in order to measure the productivity level of an individual engineer, it is important to factor aspects such as that into the equation else risk mistaking a hard working employee with little experience as ineffective and favouring a lazy more experienced employee instead. The inexperienced employee will build up their skill level and progressively become faster while the lazy experienced employee continues to stagnate.

Code Quality

We are now moving a less defined method of determining the productivity of a software engineer. The primary problem here is that there is no black and white way of classifying code. What is "good code" in one context may vary dramatically to "good code" in another likewise with "bad code" –words such as good and bad in themselves are subjective. There are however some high level similarities of what good quality code means that are shared by developers no matter the context. According to an article written by Richard Bellairs for Perforce[4] the five key traits of high quality code are: reliability, maintainability, testability, portability and reusability. In the section about algorithmic approaches, I will discuss how it is possible to measure some of these traits algorithmically however, just in terms of whether or not I believe is a good metric to use, I would argue that is one of the best. It removes the cons of just looking at code line count such as the differences between programming languages, and as mentioned before, when combined with the measurement of profit, can reasonably accurately determine the value of an individual to a company.

[4] https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-it

Measuring Productivity in Teams

In a smaller company, it is possible to measure the productivity of an individual through the methods discussed above. However, as a company scales, it becomes near impossible to constantly just look at the individual and so it is necessary to look at the value of a team instead. This is because, where things which were already complicated before, they become even more so. Take for instance the example of where within a team, one individual is working on a particular aspect of code however another has given them extensive help in the brainstorming process and in the implementation. From an outsiders perspective just looking at the code itself, if will appear as if just one person has worked on the code when in actual fact, there were two. This in itself is quite an extreme example. Through my own internship experience working as part of the Search/Search Intelligence Site Reliability team within Google I found that while as I did have my own particular piece of code to work on, it still remained a very collaborative process. I would visit my partner and chat with him about potential ideas to fix/improve his code and vice versa. All of this contributes to creating a complicated web of productivity spanning across multiple different individuals. Therefore, I would like to explore ways of measuring the productivity of a team as a whole rather than the individuals that make up the team.

Agile Software Development Methodology

"A sprint is a period of time allocated for a particular phase of a project" and "sprints are considered to be complete when the time period expires". This is one method of measuring the productivity of a team as it is easy to see if the work promised by a particular team is finished within the specified time frame. Or, if not, it is possible to tell if the team underestimated the amount of time they needed to complete the work promised in which case, it is clear to see that the team is not working as efficiently as they think they are and therefore need to adjust their expectations or, overestimated the amount, which is indicates the opposite. However, it is necessary to remember that this can be an inconsistent way of measuring progress as it measured solely by human judgement. A team could set a low goal for themselves and then work at a slower pace than their optimal leaving less work complete by the end than what could ultimately be if a more accurate goal was set. Or, set too high of a goal and then submit work that is "almost complete" as complete in order to adhere to the target. This therefore can leave the results gained unreliable.

The Five Developer Metrics[5]

According to Travis Kimmel who is Co-Founder and CEO at GitPrime there are five metrics that should be used to measure the productivity of a software development team and they are as follows – Lead Time, Churn, Impact, Active Days and Efficiency. In the following short paragraphs I will attempt to explain what each of this key words mean.

[5] https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/

Lead Time

This refers to "the time period between the beginning of a project's development and its delivery to the customer". By measuring this you can predict how long it is going to take to complete a particular project to a high degree of accuracy. It can give the manager of the team and the team itself a "better understanding of *the* team's work capacity"[6].

Churn

Churn can refer to many different things however in this instance it is in relation to code churn. Code churn is the "percentage of a developer's own code representing an edit to their own recent work". Having a lot of code churn can mean that a team is not working productively as it indicates that the engineers keep rewriting the same piece of code over and over without getting a final finished result. It can be a result of vague project requirements, a difficult project, a misunderstanding of what "ready" entails for that team, or under-engagement. Coming up with a solution for this very much depends on the cause in the first place but measuring this through looking at code insertion vs deletion can be a good way of measuring team productivity.

Impact

This word in a way speaks for itself. It is a more difficult metric to measure as again, it is a metric that can be viewed as subjective. Typically it involves looking at how code change is implemented. The example in the article looks at how adding 100 lines of code to one file can have a lesser impact than a change with fewer lines of code insertions and deletions that spans over a number of files in multiple locations. This way of measuring a team's performance can be seen as an "improvement on the LoC *(Lines of Code)*"[7] method.

Active Days

This refers to the number of days a software engineer is writing code for the project. Having a high number of active days relative to the amount of days a software engineering team spends in meetings etc. can indicate a higher level of productivity.

Efficiency

High efficiency "generally involves balancing coding output against the code's longevity". This means writing low churn, high impact code that lasts a long time.

[6] https://kanbanize.com/kanban-resources/kanban-software/kanban-lead-cycle-time/
[7] https://anaxi.com/software-engineering-metrics-an-advanced-guide/

OVERVIEW OF COMPUTATIONAL PLATFORMS

Introduction

Ironically, some of the factors mentioned above can be measured via software. Different pieces of software can measure different aspects of productivity and depending on the type of work being measured, the skill level of the employees, the culture that is strived for within the company and the size of the company, the software used will vary. An article written by Tom Gorski[8] in 2018 clearly runs through the process in which companies can go through to decide which software is best for them to measure their employees productivity. The following section will go through some of the highlights from this article along with examples of existing software.

Primary Objectives

First and foremost it is important to consider the primary objective that is being strived toward. The article mentions three of the most important factors that should be taken into account when choosing the software – a tool that helps inform the companies decisions, a tool that takes "subjectivity out of productivity" and "is based on accurate and measurable data" and finally, a tool that "motivates employees and improves performance". In an ideal world, it would be useful to have a tool that encompasses all three however, this is near impossible. Something that takes the "subjectivity out of productivity" may not be motivational for employees and vice versa.

Types of Employee Measurement Tools

Project Management

Project management tools such as Basecamp, Monday, Trello and Asana can be very effective when it comes to tracking the progress of projects. They work well for project orientated businesses that do not tend to have too much variation in the type of work being done and have already a predefined process for how things should be carried out. Where they fall down is when none of those things are the case. A lot of time can be spent simply setting up the tasks and managing them and employees can end up spending "more time fiddling with them than getting work done".

Time Tracking

Time tracking tools for example Toggl, GetHarvest and TimeDoctor have a lot of benefits when it comes to measuring the accuracy and effectiveness of employees. They enable management to better predict the amount of time future projects should take based on past readings and in the case where remote employees are paid by the hour, time tracking tools

[8] http://www.saasgenius.com/blog/how-find-best-software-measuring-employee-productivity

can be invaluable. There are two methods of implementing a time tracking tool. One being manual where the employees themselves input the start and stop times and the other being automatic where the employee just presses a start button when they begin work on a particular project and then pressing stop when finished. The main common downside to both is human error. Numbers can be inputted inaccurately throwing estimates off and likewise, employees can forget to press start/stop which again, skews future estimates. The other problem that occurs with automatic time tracking is that it is only possible to track one task at a time. Finally, a tool that monitors the amount of hours an employee spends working that stringently can negatively impact morale as it sends "a message to employees that they can't be trusted".

Team Productivity and Collaboration

Team productivity and collaboration tools are composed of tools such as Actioned, Teamweek, Teambook and iDoneThis. The reason why tools such as these exist is to make sure "that everyone involved on the project is on the "same page" and has the latest version of the current project"[9]. It allows for an overview of progress even if there is more than one project on the go at one time. Teamweek for instance provides simple yet effective visualisation of each member of a team's progress while Teambook provides that along with a comprehensive set of metrics such as availability, utilization and productivity. Tools such as these promote unity within a team even if the team is one that has members that work remotely or, is comprised completely of remote workers. They use "peer accountability" which encourages "high performance in teams" and yet is also flexible enough to include all types of work". They allow for everyone to get an insight into what everyone else is doing and through that, get a sense for how their work fits into the bigger picture. The downside to this is as mentioned in the article, these kind of tools are based fully on the input of employees and lack hard data. This means an increase in subjectivity between different teams.

Results Dashboard

Results dashboard tools include tools like Zoho, Klipfolio, Tableau and Sisense. Tools such as these are information management tools that can "visually track, analyse, and display key performance indicators (KPI), metrics and key data points to monitor the health of a business, department or specific process"[10]. These tools are centered around displaying factual operational and analytical information rather than inputted data by team members. They can help managers "spot trends" and when teams are small, help motivate employees as they are able to see exactly how their work has contributed to the end result. However, as the team scales, this becomes far more difficult and combined with this, it is the nature of software development for new features or better reliability to not relate directly to "new

[9] https://financesonline.com/collaboration-software-analysis-features-types-benefits-pricing/
[10] https://www.klipfolio.com/resources/articles/what-is-data-dashboard

sales" or "better user retention" for a long time. Contrary to the previous point of boosting team members morale it can likewise have the opposite effect of demoralizing them as it is not until after a many months that the benefit of the work they have done shines through.

ALGORITHMIC APPROACHES

Software Quality

As mentioned above in the section named *Code Quality*, the quality of software can be measured algorithmically in a number of different ways.

Reliability

Reliability "is usually defined as the probability of failure-free operation for a specified time in a specified environment for a specific purpose"[11] and it relates to the number of defects and the availability of the software. The number of defects in the software can be measured by running a static analysis tool. Static code analysis refers to, before the code is executed, debugging it by analyzing it against set(s) of coding rules. It can be done by humans however, it is generally done by an automated tool. Software availability is measured using MTBF (Mean Time Between Failures). It is comprised of:

MTTF (Mean Time To Failure) + MTTR (Mean Time To Repair) = MTBF

Software reliability itself is measured in terms of failure rate (λ):

$$\lambda = \frac{1}{MTTF} \quad \text{->} \quad R(t) = e^{-\lambda.t}$$

The number gained from this equation is between 0 and 1. The higher the number, the greater the reliability.

Maintainability

This can be looked at algorithmically using Halstead complexity measures. The more complex the program, the more difficult it is going to be to maintain. Halsted provides a number of formulas that are considered a quantitative way of looking at a programs complexity. The more complex the program i.e. the higher the number gained by filling in the formulas below, the less maintainable the software.

[11] https://hackernoon.com/software-reliability-pt-1-what-is-it-why-developers-should-care-f2a71274772c

Parameter	**Meaning**
n_1	Number of distinct operators
n_2	Number of distinct operands
N_1	Number of operator instances
N_2	Number of operand instances

Metric	**Meaning**	**Formula**
n	Vocabulary	$n_1 + n_2$
N	Size	$N_2 + N_2$
V	Volume	N * log2 n
D	Difficulty	$n_1/2 * N_2/n_2$
E	Effort	$V * D$
B	Errors	V / 3000
T	Testing time	E / k

Table of Formulas[12] (k generally being 18)

Testability

Cyclomatic complexity (CYC) was developed in 1976 by Thomas J. McCabe Sr. and in essence is "a count of the number of decisions in the source code"[13]. It is based on a control flow representation of a program which depicts a program as a graph consisting of Nodes and Edges. Basis Path testing is one of White Box techniques. It checks each linearly independent path through a program which means that the "number *of* test cases will be equivalent to the cyclomatic complexity of the program"[14]. The higher the number, the more

[12] https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.2/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

[13] https://www.perforce.com/blog/qac/what-cyclomatic-complexity

[14] https://www.guru99.com/cyclomatic-complexity.html

complex the code and the more difficult it is to test effectively. The optimal number to get for this is between 1-10. Anything above 40 is considered not testable at all.

Computational Intelligence[15]

Up until now, this essay has been focused primarily on having humans do the majority of the work. However, as the demands placed on software become more complex, there has been a gradual shift toward integration with the field of computational intelligence. Having computers instead carry out the work previously executed by humans What is computational intelligence (CI)? "Computational Intelligence is the theory, design, application and development of biologically and linguistically motivated computational paradigms"[16]. In layman's terms, it means using methods that are close to the way in which a human reasons namely, using inexact and incomplete knowledge to produce control actions in an adaptive way. Typically it is comprised of neural networks, fuzzy systems and evolutionary computation. CI exists for studying problems "for which there are no effective algorithms, either because it is not possible to formulate them or because they are NP-hard and thus not effective in real life applications".

Having read through a number of paper abstracts (e.g. Computational intelligence in software engineering by IEEE), book descriptions (e.g. Computational Intelligence in Software Engineering by W. Pedrycz and J. F. Peters and Software Engineering with Computational Intelligence edited by T. M. Khoshgoftaar) and two papers that were free (What is Computational Intelligence and what could it become? By W. Duch and Computational Intelligence and Quantitative Software Engineering by W. Pedrycz, Alberto Sillitti and Giancarlo Succi), I have come to the conclusion that while the field is exceptionally interesting and has much potential, we need not fear computers taking all the software engineering jobs just yet. Just one of the problems facing CI currently is the fact that "different applications – recognition of images, handwritten characters, faces, analysis of signals, mutimedia streams, texts, or various biomedical data – usually require highly specialized methods to achieve top performance". In other words, we do currently have the capability to carry out many of the skills we would wish CI to do for us however the "specialization" part has lead to "compartmentalization of different CI branches" and so therefore creation of "meta-learning systems" will be a great challenge as it will require the fusion of such branches. Following on from this, in order for "computers" to be able to take over the job currently undertaken by a software engineer, it will require the fusion of Artificial Intelligence (AI) and CI. In broad terms, AI is concerned with higher level cognitive functions and CI with lower level ones. The high level functions concerning things like systematic thinking, reasoning, episodic memory, planning and understanding of symbolic knowledge and the lower level functions centering around perception, object recognition, signal analysis, discovery of structures in data, simple associations and control.

[15] http://cogprints.org/5358/1/06-CIdef.pdf
[16] https://cis.ieee.org/about/what-is-ci

It is currently beyond our capacity to merge these two areas in such a way that we can mimic a humans problem solving ability.

Why look at CI at all then in the context of measuring the software engineering process? Firstly, in its current state, it does provide the ability to analyse large amounts of data be it data relating to people as discussed below, or data relating to the code itself. It is possible therefore to use that data to improve the quality of software and look at an individual's productivity level. Secondly, in an essay relating to measuring software engineering, I would argue that it is necessary to consider the next evolution of the practice of software engineering. Why measure something if not to improve upon it?

People Analytics

"The basic premise of the "people analytics" approach is that accurate people management decisions are the most important and impactful decisions that a firm can make."[17]

The article referenced through the above quote is one that has come from an article written in 2013 titled "How Google is using People Analytics to Completely Reinvent HR". It is primarily dealing with the change in human resources from a more traditional approach of "gut feeling" to one driven by data – "All people decisions at Google are based on data and analytics". This takes helps them more accurately keep track of the type of employees they have and in doing so create teams and environments that promote productivity which in turn leads to better quality software. People analytics has the capability to help business' better understand "their workforce as a whole, as departments or work groups, and as individuals, by making data about employee attributes, behaviour and performance more accessible, interpretable and actionable (Pape, 2016). This includes the use of information systems, visualisation tools and predictive analytics, underpinned by employee profiling and performance data."[18]

Measuring the software engineering process through the people carrying it out seems intuitive however, there does remain ethical concern surrounding it. An example of a company that specializes in people analytics is Sapience. It is a company that provides automated work pattern reporting and real-time analytics to help businesses better understand how people work. Having been described as "like a Fitbit for your job, only your bosses are using your results to evaluate you"[19], companies such as this bring up the question of how far are we willing to go to automate processes traditionally carried out by human judgement?

[17] https://www.tlnt.com/how-google-is-using-people-analytics-to-completely-reinvent-hr/
[18] https://www.sciencedirect.com/science/article/pii/S0268401218301750
[19] https://www.thestar.com.my/tech/tech-news/2018/05/21/want-a-fitbit-that-measures-how-productive-you-are-at-work-this-company-is-betting-your-boss-does

ETHICAL CONCERNS

Using Computational Intelligence/Artificial Intelligence

There have been, and as I imagine, will always be, ethical concerns surrounding the interaction between humans and machines. This is at a fundamental level due to our lack of trust of anything that is not human when it relates to decisions relating to us. How ethical is it to rely on computer driven concepts such as CI, AI and within that, machine learning (ML) to make decisions for us based on data given about us? I believe that, at the end of the day, it should always be the human making the final decision however, using the technology available to us can be beneficial when it comes to making the right one. It appears that through technology we currently have the ability to collect and measure the software engineering process spanning over countless different metrics. Right from information gained about our step count, dietary habits, how long we spend on self-care, and how long we spend in meetings, to the frequency we produce code and its subsequent quality. Computers can make a cold and calculated decision based on all of these factors while discounting the fact that as humans, we do have our off days, weeks and even months due to the fact that we do not spend all of our life working. That begs the question – if computers lack the capacity to have "off days", why not automate everything instead? As discussed above, there is an increase in the move toward automating jobs that were previously carried out by humans in order to increase productivity and we are certainly heading in a direction where, in the future, computers could take on more and more complex tasks leaving humans with fewer and fewer jobs. There are many ethical issues surrounding that in itself - how much can we rely on algorithms to make the best decisions for us? looking at the kind of jobs that should be automated and not just looking at the jobs that can be, and finally, is it really fair to depend on inanimate objects to pass judgement on living, sentient beings? (This argument I am sure will take on a whole new light once scientists can come up with a way to construct a machine that is conscious)

The 'Fitbit on the Dog' conundrum

How far should we go when measuring the software engineering process? How much is too much? It is important to remember that, while all this innovation surrounding increasing the capacity for companies to measure their employees productivity levels certainly has huge potential, as Behrand a professor from the University of Washington says, "human behaviour is complicated". It is a well known fact that people act differently when they know they are being surveilled and in particular, if this surveillance can be used to create a leaderboard on which they are placed. Without access to the data that they are being measured upon, surveillance to that degree can create an atmosphere of fear and uncertainty. This in turn breeds a culture that is counter to the one that was initially set out to be achieved – one of productivity. In conjunction with this, when faced with competition, people go to "sometimes unproductive lengths to increase their standing". The example in this case being, competing with friends over who is taking the most steps in a day and putting the Fitbit on the dog to "win". In practice, this is not winning at all as the primary objective is to become a healthier person. By putting the Fitbit on the dog yes, according to the data you have taken the most steps however, there has been no progress at all toward the end goal i.e. productivity has been drastically reduced. If people are competing in their way they tend to become more ruthless, less willing to help their colleagues. This once again reduces productivity as the best work tends to be done in a collaborative process. One way a lot of this an be remedied is giving employees access to both the metrics they're being measured by and the data that has been collected about them. Then, rather using it as a means to pitch companies against one another, use the data collected as a means of self-help so that employees can see what they need to work on rather than have the whole thing carried out as a black box affair.

READING MATERIAL

https://financesonline.com/collaboration-software-analysis-features-types-benefits-pricing/

https://collaboration-software.financesonline.com/top-10-collaboration-software-services-you-should-try/

https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-it

https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo

https://www.martinfowler.com/bliki/CannotMeasureProductivity.html

https://kilthub.cmu.edu/articles/Establishing_a_Software_Measurement_Process/6573467/files/12059171.pdf

https://www.geeksforgeeks.org/software-measurement-and-metrics/

https://dzone.com/articles/measuring-productivity-of-your-software-developmen

https://go.tasktop.com/rs/246-WDG-185/images/Understanding%20Software%20Development%20Productivity%20from%20the%20Ground%20Up.pdf

https://www.wrike.com/project-management-guide/faq/why-should-i-use-time-tracking-in-project-management-software/

https://teambookapp.com/features/

https://teamweek.com

https://linchpinseo.com/the-agile-method/

https://blog.gitprime.com/6-causes-of-code-churn-and-what-you-should-do-about-them/

https://en.wikipedia.org/wiki/Software_reliability_testing

https://www.personneltoday.com/hr/people-analytics-data-can-help-boost-performance-productivity-retention/

https://www.humanresourcestoday.com/analytics/productivity/?open-article-id=10800730&article-title=how-technology-can-democratize-people-analytics&blog-domain=deloitte.com&blog-title=bersin-with-deloitte