Measuring Software Engineering Report

Jack Keenan – 18324599

Introduction

This aim of this report is to provide an insight on how the Software Engineering process can be measured, describe some of the computation platforms and algorithms available for measuring software engineering and finally discuss some of the ethical concerns relating to the above. The report will highlight how measuring software engineering effectively is a difficult task, but it has its rewards. The results from measurement will allow managers and individuals to analyze their performance and pinpoint the areas that need to be improved. Ultimately this can increase return on investment, reduce project costs and result in efficiency improvements.

Measurement

The measurement of software engineering is not a straightforward task. Although there is a huge amount of data available its reliability, usefulness and relevance can be questioned. It has been argued that software engineering measurement cannot be done effectively and therefore should not be done at all. The development process as a whole has so many variants that it is extremely difficult to pick a single measure or even a mix of measures that effectively analyzes the process. However although there is no silver bullet that analyses software engineering as a whole there are plenty of metrics that will provide us with results that can be used to identify inefficiencies, reduce costs and time and highlight where productivity needs to be improved and therefore measuring software engineering is an important task that should be done. There is a lot of data you can observe and measure that is produced by a team of engineers. This report looks at:

- Lines of code
- Number/Frequency of commits
- Velocity
- Quality

Lines of Code

When thinking of the ways in which an engineer can be measured an obvious data point that many people suggest as an appropriate aspect to measure is the lines of code. Lines of code is debatably the most commonly developed metrics by the software engineering industry due to its convenience to analyze and the ease in which it can be visualized. It can easily be assumed that if an engineer is writing lots of code, they are productive. However, it has been proven over time and for obvious reasons that quantity does not mean quality. This method of measure completely disregards nice and concise solutions that are easy to read as it concentrates on output rather than outcomes. Is 200 lines of code really better than 20 lines of code if they solve the same problem? Of course not! The 200 lines of code is much harder to maintain which results in higher costs. So, rewarding people for writing code that could otherwise be done more concisely is not an effective measure. This would encourage developers to drag out their code in order to appear more productive which is not desirable for code reviews and repairs down the line.

Commits

A second and easy way to measure engineers is by the number and frequency of their commits. Although convenient this is a really ineffective measure as commits do not correspond to achievements. Of course, developers should be committing regularly as a rule of thumb but what needs to be committed is down to the personal preference of the developer and therefore should not be used to compare developers. A commit can be as small as changing a variable name or a adding a comment so measuring productivity by the volume of commits is simply not effective.

Velocity

Velocity is another measure that can be used, it is a measurement of the quantity of work an agile team completes during a software development sprint. It measures the story points finished over time. Story points are an estimate of the amount of effort it will take to complete the product backlog or any other aspect of the project. Using this as a measure of process has issues. The first being that teams have different objectives and circumstances which voids the validity of comparison. The second being if story points are going to be used as a method of evaluation, teams may have the tendency to play with the system and inflate the story points. Velocity as a metric can often result in communication breakdown as teams become more focused on improving their velocity instead of collaborating with other teams. This can result in subpar products being delivered. However, if the team and manager are willing to engage with the metric to analyze their performance by providing accurate estimates of story points it can be useful metric. With covid-19 came a new challenge as many teams were forced to work remotely where they had been operating in an office beforehand. It was important for managers to ensure that their teams maintained their performance and so looking towards measuring velocity in order to ensure their productivity was not affected by the lockdown would have been useful.

Quality

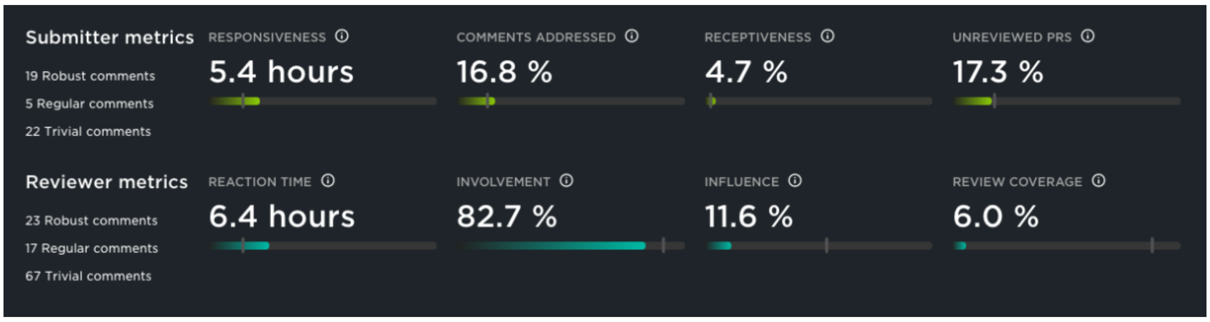
Quality focuses on global measures and outcomes of code that has been already delivered to the production environment and thus is a good measure. Some metrics used to measure the quality of software are 'Mean time between failures' (MTBF) and 'mean time to recover'(MTTR). MTBF measures the average time between breakdowns of the system and looks at the performance, the safety and the design of equipment ultimately examining the reliability of an asset. Teams who use the metric generally perform better as it motivates them to deliver better quality code in order to reduce the MTBF. Predicting how likely a breakdown is within a certain period and how often a particular error may occur can help increases reliability and avoid costly breakdowns. MTTR measures the period of time a service is down. It begins as soon as the service goes down and includes repair time, testing time and the time it takes to complete all other exercises until the service is returned to users. A low MTTR indicates the service or component can be repaired quickly and as a result does not really affect the business. In contrast a high MTTR indicates that a failure could have a significant impact on the business. The difference between MTBF and MTTR is that MTBF tells the organizations how often the system breaks down whereas MTTR indicated how quickly they can get the system back up and running after a failure making these two metrics useful together to improve the engineering process as a whole to maximize performance.

Computational Platforms

There is a large quantity of computational platforms that analyze data providing users with informative reports and visualizations that will help highlight different aspects of the software engineering process that need improvement.

Pluralsight

Pluralsight flow collects historical Git data and processes it into clear and informative reports so that a team's success can be measured. It thinks of Git as a record of how a team works rather than simply just version control. It is a platform that can identify bottlenecks that need to be removed in the workflow leading to the team being more efficient and effective. Instead of just measuring the number of commits or the number of lines of code that a worker has it looks deeper into the commits they make. Flow will identify how long is spent adjusting old code versus writing new code. The platform can be used to upskill the developers as it gives an insight to the effectiveness and performance in the different programming languages by analyzing commit efficiency, skill IQ and it also provides skill development recommendations for the team. It also allows for the management of large-scale pull requests by providing visualizations of the team's code review dynamics by identifying if reviews are productive, by scanning for disagreement triggers in code reviews and by observing patterns in the process. Flow provides clear and concise reports allowing managers to identify bad patterns that need to be improved and aids sprint planning by evaluating who is contributing often and performing well.



Submitter metrics

19 Robust comments
5 Regular comments
22 Trivial comments

RESPONSIVENESS	COMMENTS ADDRESSED	RECEPTIVENESS	UNREVIEWED PRS
5.4 hours	16.8 %	4.7 %	17.3 %

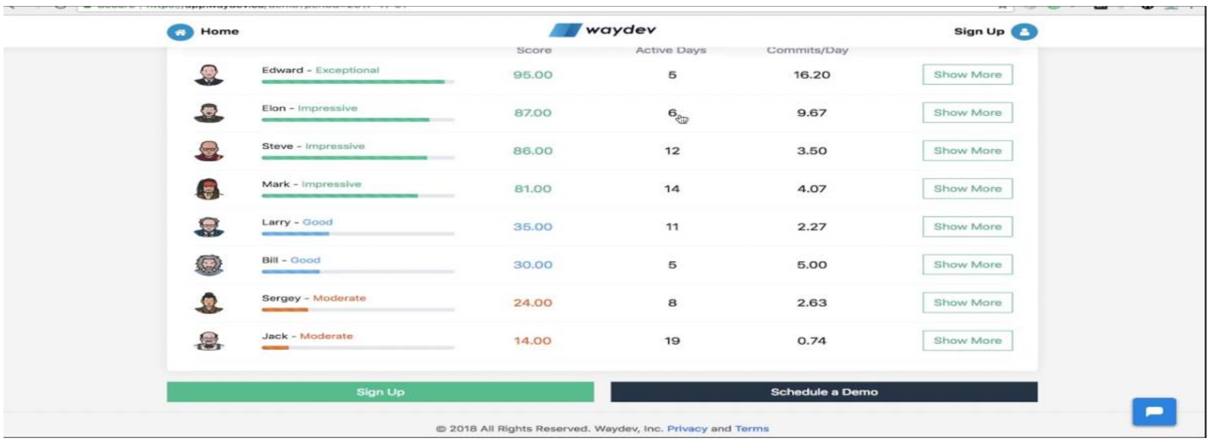
Reviewer metrics

23 Robust comments
17 Regular comments
67 Trivial comments

REACTION TIME	INVOLVEMENT	INFLUENCE	REVIEW COVERAGE
6.4 hours	82.7 %	11.6 %	6.0 %

Waydev

Waydev is another platform that analyzes and computes performance. Waydev is an agile data driven method of tracking engineers. Similar to Flow it unlocks the value of data collected in Git platforms. Waydev generates objective reports without the manual input of the engineers. It analyses the code base, pull requests and tickets to help identify where the engineers can improve. Like flow, Waydev increases the visibility over the team by generating easy to understand visual reports encouraging the acceleration of the team's velocity.

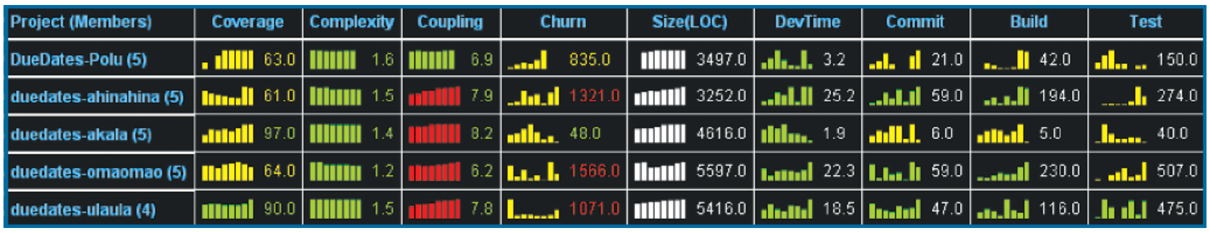


	Score	Active Days	Commits/Day	
Edward - Exceptional	95.00	5	16.20	Show More
Elon - Impressive	87.00	6	9.67	Show More
Steve - Impressive	86.00	12	3.50	Show More
Mark - Impressive	81.00	14	4.07	Show More
Larry - Good	35.00	11	2.27	Show More
Bill - Good	30.00	5	5.00	Show More
Sergey - Moderate	24.00	8	2.63	Show More
Jack - Moderate	14.00	19	0.74	Show More

© 2018 All Rights Reserved. Waydev, Inc. Privacy and Terms

Hackystat

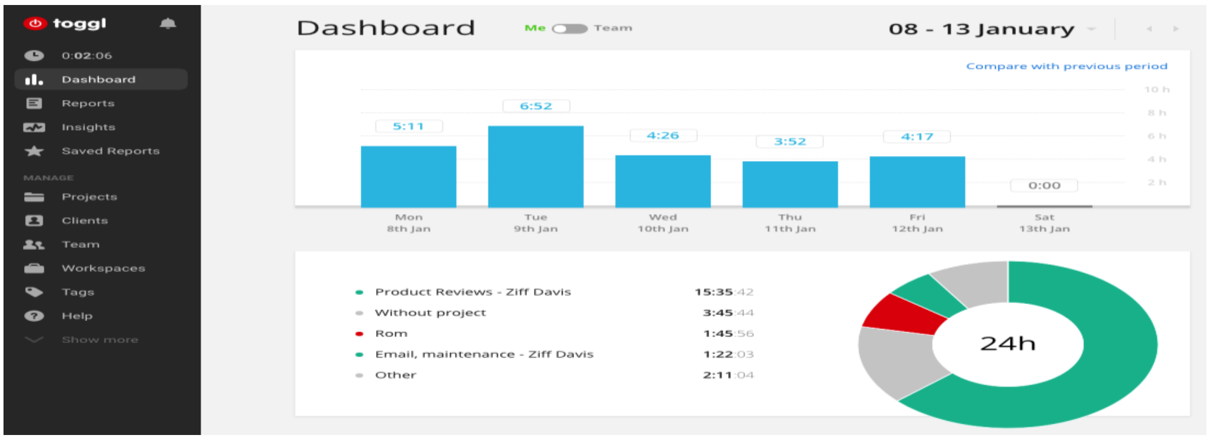
Alternatively if an organisation does not have the resources to pay for a process analysing platform hackystat is an open-source option. Hackystat is a collection of small services that work together to provide collection, analysis, visualisation, annotation, dissemination and interpretation of the data from software engineering activities. For example google charts is a service that is used by hackystat to visualise the data. Software plugins are attached to the development tools which send data to the sensor base. The data in the sensor base is then abstracted and various queries and computations are performed on the data to derive the desired reports.



Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

Toggl

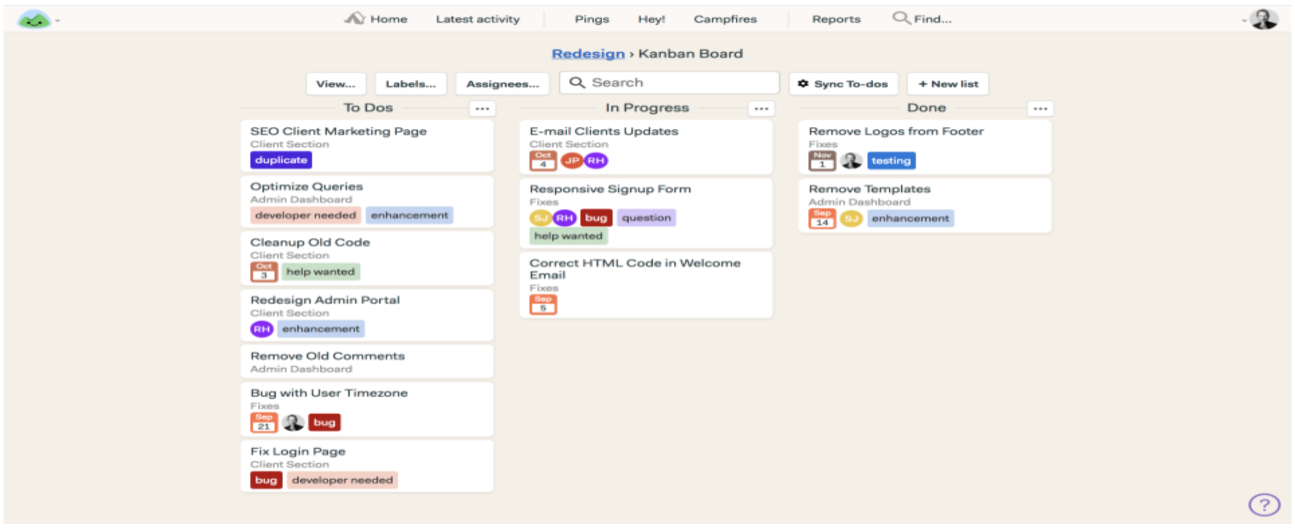
Toggl is a time tracking resource that concentrates on productivity but unlike the other examples the platform is not solely focused on code. It gives users the ability to accurately record and track their time be it checking emails, writing and testing code or just about any other activity they perform while working. This allows users to analyse exactly what they are doing during the day. Summaries like in the image below can be provided so that managers or users themselves can study how they spend their time and see if their time management skills can be improved on. More often than not workers don't realise just how long they are spending on certain areas of their work and so toggl is a great platform to identify inefficiency. The report parameters can be changed to arrive at different types of summaries be it daily, weekly or monthly and the option is given for managers to email workers the summaries so that they too can review their time allocation.



● Product Reviews - Ziff Davis	15:35:42
● Without project	3:45:44
● Rom	1:45:56
● Email, maintenance - Ziff Davis	1:22:03
● Other	2:11:04

Basecamp

Basecamp is a project management tool that allows teams to increase their productivity and efficiency. Basecamp breaks work into different projects that contain everything related to the task at hand including files, to dos lists and due dates. Basecamp is particularly useful at present due to employees working remotely because of Covid-19 as it keeps everyone updated and in the loop. It makes planning much easier because it is clear what tasks have been completed and what tasks are overdue. Making it easy for managers to see what team members are getting through the tasks assigned and what team members need to improve their performance.



Algorithmic approach

On top of computational platforms there are also a number of algorithmic approaches at hand that can be used to measure the software engineering process. This report takes a look at Halstead complexity measures, cyclomatic complexity and computational intelligence.

Halstead Complexity Measures

Halstead complexity measures are a set of metrics used to measure the complexity of software by going through each line of code and determining which tokens are operands and which tokens are operators. The the following can be collected:

n1 = the number of distinct operators
n2 = the number of distinct operands
N1 = the total number of operators
N2 = the total number of operands

The above numbers can be used to calculate program length, program vocabulary (*total number of unique operator and operand occurrences),* volume (*size in bits of space needed to store the program*), difficulty (*how difficult the program is to write and understand in a code review*) and Effort (*Time required*).

Halstead metrics have many advantages such as they are easy to calculate, can predict the rate of error and maintenance effort and can be used on any programming language. However, the metrics depend on completed code and are not commonly used as they are seen to be inaccurate due to the fact there is no specific rule for the identification of operators and operands which can lead to different calculation results on the same code.

Cyclomatic Complexity

Cyclomatic complexity is another measure used to determine the complexity of an algorithm. It is a measure of the linear dependent paths through a programs source code. The control flow graph of a program is used to calculate the cyclomatic complexity. The nodes in the graph are used to denote the smallest group of commands in the program and the directed edge is used to connect two nodes. The formula of cyclomatic complexity is:

*Cyclomatic Complexity = E – N + (2*P)*

Where: *E = Number of Edges in the flow graph*

N = Number of nodes in the flow graph

P = Number of nodes that have exit points

Cyclomatic complexity evaluates the risk associated with the program and helps determine the independent path executions which is helpful for developers and testers. It is easy to apply, can help guide the testing process and can be computed faster than the Halstead metrics. Programs with a lower cyclomatic complexity are easier to understand and have less risk associated with modification to the program. Code with high complexity is difficult to test and is likely to have errors.

Computational Intelligence

“Computational Intelligence is the design, theory, application and development of biologically and linguistically motivated computational paradigms”. Computational Intelligence (CI) has come to the forefront in recent times as more and more companies and organizations look towards it for examining data. Given some inputs it can be used to generate feedback and make decisions. CI allows problems that cannot be translated into binary in order for the computer to be able to process, so essentially is a way of performing like humans. CI has three main pillars – neural networks, fuzzy systems and evolutionary

computation. In addition to the three main pillars it extends to other principles such as Learning Theory and Probabilistic Methods.
Neural Networks uses the human brain as inspiration creating artificial neural networks. These artificial neural networks are given examples which they can learn and generalize from in order to make decisions in a similar manner to a human. Neural networks are used for data analysis and classification and therefore are useful in examining data related to the software development process.
Fuzzy Systems can face incompleteness in data as opposed to Artificial Intelligence which needs exact values and knowledge by using Fuzzy Logic. It uses approximate reasoning through traditional logic to solve uncertain problems.
Evolutionary Computation solves optimization problems by using biological evolution as a source of inspiration. To do this it generates, evaluates and modifies a group of possible solutions.

Ethical concerns

The measurement and analysis of software developers raises many ethical concerns. It is important for a company to know that their workers are working productively but at the same time workers can be measured inaccurately and their privacy can easily be invaded which can lead to stress resulting in reduced productivity.

Time Theft

Time theft occurs when an employee receives pay for time they did not actually work and is considered stealing of company time. Employees can commit time theft by directly stealing time for example getting another employee to clock them out, employees may take longer breaks than they are supposed to or employees may take on personal tasks while at work. It includes doing just about anything to kill time when the worker should be working. It can be argued that the companies should have the right to monitor employees to ensure they are working as they should be. Particularly with the large number of workers now working remote due to covid-19 employers can no longer directly see what they are doing during working hours and employees could be doing just about anything but working while at home. As a result, organizations should be able to ensure workers are working while on the clock as they are paying for their labour.

Panopticons Disciplinary Concept

Panopticons disciplinary concept was derived from occurrences in prison. The idea of the concept is built on the fact that in a prison from a guard tower the guard can see all the prisoners, but the prisoners cannot see the guard and so cannot determine whether they are being watched at a particular moment in time or not. In the modern day it can be argued that this concept can be applied to workers whose computers are the guard in the watch tower and they are the prisoner. The employee can be monitored but cannot see exactly when or what is being monitored which can be considered unethical as it may result in stress for the employee. Tracking employee's performance has extended from the data generated from their computer to data derived from on-person devices and devices around the office. For example, humanyse the employee tracking badge discussed in class is a device that monitors how an employee interacts by measuring the tone of voice they use in their conversations. This can lead to uncertainty and ethical concerns as workers may fear that the devices tracks more than what is described such as the fear that humanyse is tracking the content of the conversations a worker has rather than just the tone the worker uses in the conversation and thus violating their privacy and linking into the panopticons concept.

Accuracy

We must also question how accurate the measures of the employees are. Can the workers productivity solely be measured on their performance in the metrics we use to analyze them? or does the worker do work that goes unnoticed in the calculations? More often than not developers collaborate on projects and the metrics outlined above do not account for the time, effort and effectiveness of help offered to each other. On a team there may be a developer that keeps to themselves and just completes the work assigned to them and on the other hand there may be a developer who offers help to other developers in order for the team to progress as a whole. Is it really fair to compare the performance of these two developers solely based on the data collected by the analyzing tools? It is quite likely the developer that communicates with the team is having a vital impact on the software engineering process that they are not being credited for as there is no measure in place to evaluate this aspect of teamwork. Building on this, measurement can encourage workers to become more self-centered as they may see it as a sense of competition which could lead to communication breakdown and less effective results in projects and tasks. Thus, is can be considered unethical to measure performance as it cannot always be evaluated accurately, and workers could be criticized for the positive impact they have outside the metrics.

GDPR

There are now laws in place to protect employee's data, in May 2018 the General Data Protection Regulation (GDPR) was introduced to the EU. The introduction of these regulations meant that employees have the right to information regarding the collection and processing of their personal data, access to the data held about them, have their personal data erased and question the data control about collecting their data if they consider it unlawful. Although the data is protected from being processed by the company in an unlawful manner or in a manner non consistent with the regulations this does not mean that the data is safe. This leads us to ask the question what data is absolutely necessary to store. We have seen in the past that companies have had data breaches and attacks so is it really ethical to be holding vast amounts of non-vital employee data if it puts them at risk in the event of an attack?

Conclusion

In conclusion, there is no one size fits all way of measuring the software engineering process. Although there are plenty of ways to measure software each way provides different insights that can be used to improve a team. The report provides some examples of computational platforms and algorithmic approaches that can help better analyze the process so that improvements can be made ultimately having a positive effect on the time, costs and performance of a project. Finally, the report discusses ethical issues raised by the measuring of developers. Although ethical concerns arise there are now laws in place to protect the worker and in my opinion the positive impact that can be made by learning from the results of measurements outweighs the ethical concerns.

Sources

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

https://www.infoq.com/articles/measuring-tech-performance-wrong/

https://about.gitlab.com/blog/2016/03/08/commits-do-not-equal-productivity/

https://www.fiixsoftware.com/mean-time-between-fail-maintenance/

https://www.splunk.com/en_us/data-insider/what-is-mean-time-to-repair.html

https://www.pluralsight.com/product/flow

https://waydev.co/about-us/?gclid=CjwKCAiA7939BRBMEiwA-hX5J1fNZjzHkEcpPSMx9CUP-lzyr2dzBNNPq49QRDaakik5eKqn5laH0RoC_zIQAvD_BwE

https://philipmjohnson.org/projects/hackystat

https://toggl.com/track/features/

https://basecamp.com/how-it-works

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

https://www.geeksforgeeks.org/cyclomatic-complexity/

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

https://www.patriotsoftware.com/blog/payroll/time-theft-what-is-prevention/#:~:text=Time%20theft%20is%20when%20an,is%20considered%20stealing%20company%20time.&text=Time%20theft%20primarily%20applies%20to,there%20are%20for%20salary%20employees.

https://ethics.org.au/ethics-explainer-panopticon-what-is-the-panopticon-effect/#:~:text=The%20panopticon%20is%20a%20disciplinary,not%20they%20are%20being%20watched.

https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/data_protection_in_the_workplace.html#