Measuring Software Engineering

Brief:

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

Measurable Data:

Software engineering is a complex task. It does not simply consist of writing reams of code and hoping for the best. Software engineering consists of identifying, often highly complicated and intricate problems and solving them in the most efficient manner. Logical solutions often don't present themselves immediately, higher order thinking is often required to identify solutions. Keeping this in mind, how is it possible to quantify the output of a software engineer? The goal of a software engineer is never to produce as many lines of code as possible as fast as possible yet so often companies attempt to put a number or rank beside an engineer to define their contribution to the company.

It is necessary to be aware of the measurement techniques employed by the software engineering industry. While some do provide insight into the productivity and effectiveness of an engineer, no one piece of measurable data can claim to fully encapsulate the contribution of one's output.

It is important to have a specific goal in mind when collecting data. Too often companies invest excessive amounts in data collection technology with no means of quantifying if the investment yielded results. The question of what we are looking to measure must be asked. There are several key attributes a great software engineer should have.

1. Positivity: A great engineer cares about the customer and the end-product. They are a beacon of constant positivity and ensure jobs are done to the best of their ability. They are willing to go the extra mile and bring their best self to work everyone. Teammates work better when with them due to positive atmosphere that radiates from them.
2. Communication: There is a direct correlation between good communication skills and excellent development. Excellent communicators can ask the right questions at the right time ensuring correct product specifications. A great developer can also relay complex information back to other engineers in a coherent manner, excelling at articulating complex ideas.
3. Time Management: Great time management leads to a highly reliable engineer. They have a strong work ethic, showing up to work on time and working efficiently to deliver their workload. Exceptional developers excel managing their clients and teammates instead of expecting to be managed.
4. Self-improvement: Engineers who have the ability, willingness and drive to constantly evaluate their work and look for opportunities for self-improvement tend to outperform their peers. They don't shy away from new methods and technologies instead embracing an opportunity to upskill.
5. Diversified Toolkit: Great engineers are competent with many technologies. An experienced developer is well versed in best practices like agile development, task management software and version control.

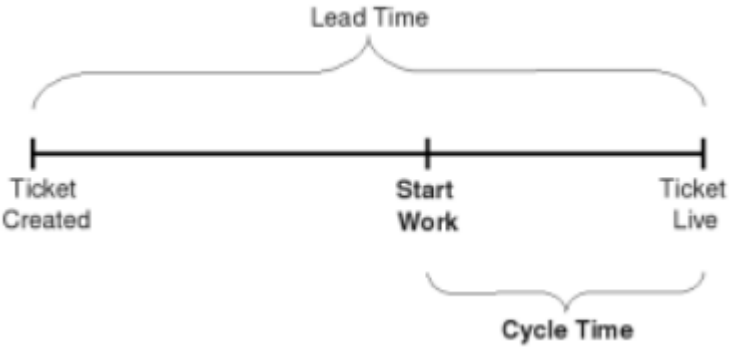
Defining the attributes, you are looking for in a software engineer, now allows you to begin collecting data. At this point, a specific measurement should be decided on. It is not acceptable to want to measure a 'great engineer', this is vague and highly subjective. Consequently, the collected data will lead to obscure and or misleading results. The key thing to decide upon is a specific purpose for the data.

Lead Time

A lead time is the latency between the initiation and execution of a process. In the case of software engineering, this is time expired from when an issue is first logged until work is completed on that issue. This lead time is the time and not the effort. You may have a lead time of 100 days but only take 10 minutes to fix a reported bug. Lead time is relevant and important from a business perspective. The cycle time can show how effective an engineer is at solving a problem once it reaches them. To reduce lead times, cycle times should be reduced but it is only one actor. Often excessive lead times are due to improper business practices where software engineers rarely can influence. (Roock, 2010)

Technical Debt

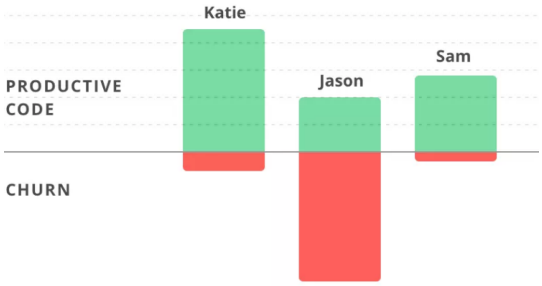
Technical debt (also known as design debt or code debt) is a concept in software development that reflects the implied cost of additional rework caused by choosing easier solutions now instead of alternate approaches that would take longer at the time but would require less work in the future. Not addressing debts as soon as possible leads to more debts, due to difficulty in altering the software and increasing the risk of breaking the software after each update.



Technical debt can be caused by a variety of factors such as; poor conception, poor scheduling, bad development practices and outdated technology. Some of the most common types of technical debt being source code formatting, low test coverage and code complexity issues. It is crucial to quantify the levels of technical debt in your code, this should not be done manually. There are a few tools available like 'Coverity', 'SonarQube' and 'Code Climate' which will be discussed in the section on computational platforms. These tools will give an estimate in days or hours needed to fix this debt. Measuring the technical debt gives us an approximation of the time needed to fix the technical debt. Even though the attempt to reduce technical debts can be costly, not addressing them at an early stage will cost more money and time in the future. (Osetskyi, 2018)

Code Churn

Code churn looks at the percentage of a developers own code representing an edit to their own work. This can be measured by looking at the lines of code modified, added or deleted over a short period of time. When churning spikes this can be an indicator that something is off with the development. Code churn has been noted to be the proverbial canary in the coal mine. (Thompson, 2016) There are many causes of code churn, some of the most frequently encountered include; unclear requirements, indecisive stakeholder, difficulty of problem and under-engagement.

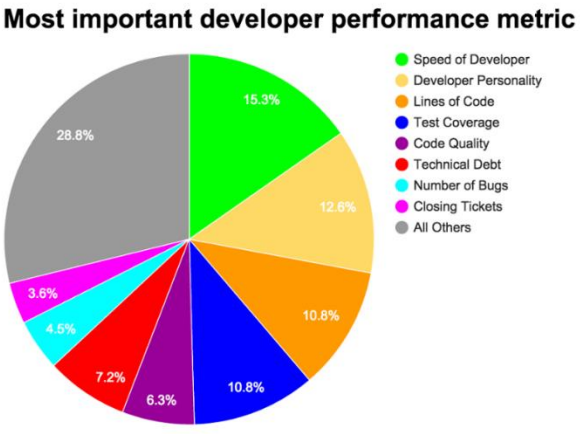


Active Hours /Days

This is an obvious and easy to calculate metric.
It is easy to fall into the pitfall believing the more the better. Someone who works 10 hours instead of 8 hours should complete 125% of the work. This is not the case. There is a diminishing marginal return to labour that leads to drop off in productivity after a certain amount of hours /days is reached. Countless studies have been undertaken analysing the relationship between hours worked and net productivity. The pervasive attitude of work longer, work better needs to end. (Collewet, 2017)

This list is not exhaustive, many other metrics and data exist when attempting to measure software engineering. No one type of data can be objectively defined as superior to others. Each type has a set of pros and cons which must be considered. What should be looked at is which type of data will be most helpful in achieving your goal set out prior to data collection.

An interesting point to note and one that rarely is considered is what developers themselves see as the most helpful in determining performance. A recent campaign lead by renowned entrepreneur and investor Brian York looked to do just this. Over 300 developers were asked what they believed to be the most important performance metric when engineering software. The results were as surprising as they were varied with no metric dominating. The results are displayed below.



This study highlighted there is no consensus when it comes to collecting data for performance evaluation. While it can and is debated which metric should be highlighted above others, it would be wrong to say performance shouldn't be measured at all. It will be a decision each company should take independently incorporating the stage of business they are in and what areas of development need to be highlighted. Tools and metrics are available, and performance can and should be tracked. (York, 2015)

Limitations to data collection:

As noted in the introduction to data collection, many attributes cannot be easily measured. Qualitative aspects are much harder to rank and compare than those that are quantifiable. Consider a group of software engineers working together in a team. If one member spends time supporting and promoting their team mates, how can this be incorporated into a metric? How do you map the progress of an

engineer's contribution to team moral over time? Most would concur a positive environment is conducive to performance. No one has yet devised a strategy for effectively measuring this aspect.

There is a trade-off between easily obtained data and richer analytics requiring increasing overheads and some privacy issues. There is a common form of observational bias, "The Streetlight Effect", named after a joke where a man drops his phone then proceeds to search for it a distance away where the light is better. This is commented on extensively by Philip M. Johnson in a research paper (Searching under the streetlight for useful software analytics) analysing the relationship between software analytics and its usefulness. Easy obtainability of data usually yields lower impact results in terms of its usefulness. Once a goal has been identified, it is much better to light a candle than use the streetlight already there.

Computational Platforms:

Firms will first decide what they want to measure, then they will collect the relevant data. The next question a firm will ask themselves is how they can leverage this data to gain meaningful insights into the performance of a software engineer. The answer will be a form of computational platform. A computational or digital platform is the environment in which a piece of software is executed. It may be the hardware or the operating system (OS), even the web browser and associated application programming interfaces or other underlying software as long as the program code is executed with it.

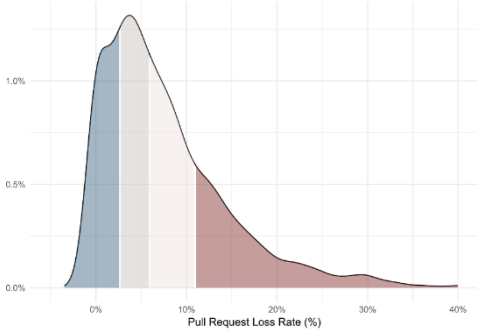
More and more firms are entering the data analytics industry, providing companies with an abundance of choice when it comes to selecting the platform that will best suit the firm's needs.

Personal Software Process (PSP):

One of the first significant developments in the measurement of software engineering performance was brought to us by a man named Watts S. Humphrey. He is known widely as "the father of software quality". Humphrey was the one of the first to recognise the need for a greater understanding of the characteristics that maximised the quality of software engineering. Humphrey was a well-respected professor at a prestigious Pennsylvanian research university during the 1980s. He founded the Software Process Program at the university and served as the director of that program from its inception in 1986 to 1993. This program was aimed at understanding and managing the software engineering process. He believed an extensive understanding of the process could yield significant improvements in the software engineering process. It was work on the program that inspired him to develop the revolutionary personal software process (PSP).

The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. While most industrial organizations had by now adopted modern quality principles, the software industry relied primarily on testing as the principle quality management. The significant step that the PSP took was to extend the improvement process to the software engineers themselves. The PSP concentrated on the practices of individual engineers. The PSP is designed to help software engineers consistently use sound engineering practices. It gives detailed steps on how to plan and track their work, using defined and measured processes, establishing measurable goals, and comparatively analyse performance against these goals. Humphrey laid out the framework for this model in his paper, 'The Personal Software Process' published in 2000.

Planning	Requirements Analysis
	Conceptual Design
	Proxy Based Size Estimation
	Time Estimation
	Schedule Estimation
Development	Design/Review
	Code/Review
	Compile
	Test
Postmortem	



The PSP was centred around self-assessment and management. Humphreys believed that by following a structured framework, a developer could better understand their output and thereby be able to improve it. Due to the time-consuming nature of the PSP, data collection and analytics were carried out manually, it proved too inefficient. Although it was a pioneer in the search to better understand the software engineer process. Many future platforms have built on the groundwork laid by Humphrey.

PRO Metrics (PROM):

There is a prevalent view among software engineers that measuring and collecting data to improve efficiency is not an important activity, especially when compared to coding. The creators of PROM understood this and created an automated tool that bypassed the engineers. PROM could collect and analyse software metrics as well as personal software processes data. The architecture is made up of plug-ins that can automatically collect and distribute data with no interaction with the developer. The infrastructure of PROM is shown to the right. This was an essential improvement on the user-controlled implementation of the PSP. It saved a massive amount of time, effort and was less liable to malpractice. One issue noted with the PROM is the limited producible metrics. A predetermined amount of data was collected, limiting standardization methods. A firm-wide server would receive data collected from individual computers where the PROM had been installed. The server would house a database where the information would be stored and could be analysed and interpreted.

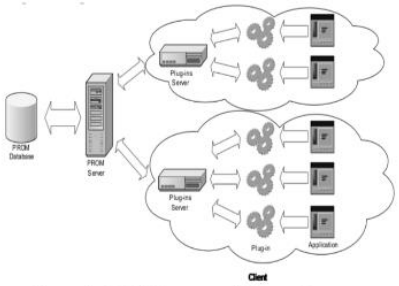


Figure 1: PROM data acquisition architecture

Coverity and SonarQube:

Coverity and SonarQube are two of the computational platforms that can deal with coding issues, they were briefly mentioned earlier in the article.

Coverity

Firstly, Coverity is a toolbox of software development products that enable engineers and security teams to find defects and security vulnerabilities in customer source code written in several languages. Coverity has had many famous applications such as on the software used by CERN in the Large Hadron Collider and during the flight software development of the Mars rover Curiosity. (YouTube, 2011)

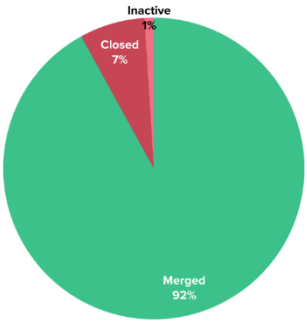
SonarQube

SonarQube on the other hand is a web-based open source platform used to measure and analyse source code quality. SonarQube is known for its extreme versatility, being able to handle more than 20 programming languages. SonarQube can complement other static analysis tools. It covers a wide are of code quality check points which include: Architecture and Design, Complexity, Duplications, Coding Rules, Potential Bugs and Unit Testing. (Shrestha, 2010)

Code Climate:

Code Climate is a privately held software company, founded in 2011 by Bryan Helmkamp and Noah Davis. Its offerings include a plethora of analysis tools to aid software developers in code quality and efficiency. Code Climate is one of if not the most popular computational platforms utilized by software engineers, it is used by over 100,000 projects, and analyses over 2 billion lines of code daily, Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous, and ubiquitous. As with other premier computational platforms, many languages can are compatible with Code Climate including Java, JavaScript, Swift, PHP, Python and Ruby. (Code Climate, 2018)

Some of most useful tools include its synchronization with GitHub, with each Git commit, Code Climate analyses the new code entered and will determine changes in quality and technical debt hotspots.



Drawbacks of the Code Climate platform include the pricing, it is significantly more expensive when compared to its competitors. Commentators have also noted the Code Climate team are unresponsive to customer requests, they show little to no interest in extending tools in the directions customers suggest.

Code Climate really stands out from its peers in its ability to visualize data. This seem less visualization allows for meaningful insights to be derived. Its User Interface is top of the range. Some of the visualizations gathered by the platform are included.

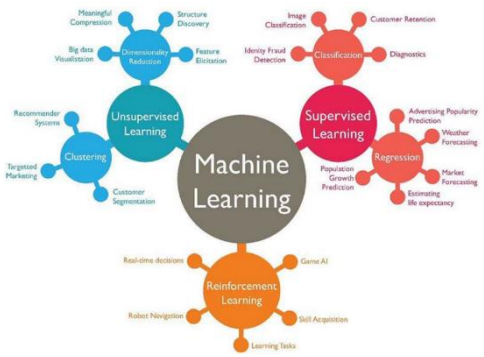
Algorithmic Approaches:

There are an overwhelming number of algorithms and algorithmic approaches available in the world these days. Each one takes a different approach to measuring and assessing the software engineer and their individual processes. When considering which algorithm to use, like which data to collect, a company should identify which algorithm can add value to their business. Which ones are relevant and useful? Some approaches will be considerably more helpful than others.

Machine Learning:

Machine Learning has increasingly gained popularity in the past couple of years. As Big Data is the hottest trend in several different industries now including tech, finance and manufacturing. More and more investment has been exponentially piled into this space, presumably results would follow suit. This has not been the case. Companies must be careful of falling into the 'big data trap' where more and more money is funnelled into their data team, but no quantifiable metrics are used to measure their output.

Machine learning is a difficult to define concept, it is made up of three subcategories; supervised learning, unsupervised learning and reinforcement learning.



1. Supervised Machine Learning: An algorithm of this nature maps an input to an output using data for which the input and output are already defined. This set of known data is referred to as the training set. The algorithm makes predictions based on the

training dataset. This process is repeated until deemed acceptable by the user. Regression algorithms are one type of supervised machine learning. Regression allows for the prediction of a continuous outcome variable (y) based on the value of one or multiple predictor variables (x). The goal of the regression model put succinctly is to build a mathematical equation that defines y as a function of the x variables. Classification is the other type of supervised machine learning which will be commented upon in the multivariate data analysis section. (Brownlee, 2017)

2. Unsupervised Machine Learning: Unsupervised algorithms do not require an output set of data. They are used to discover inferred relationships in a given unlabelled dataset. There are two types of techniques, clustering and association. Input data is placed into distinct groups based on similarities and differences.
3. Reinforcement Learning: This area of machine learning is fascinating for the unexpected results it can produce. It operates under a reward-based system where it is rewarded for making corrected decisions and punished for making incorrect decisions. It is concerned with how software agents ought to take actions in an environment to maximize some notion of cumulative reward.

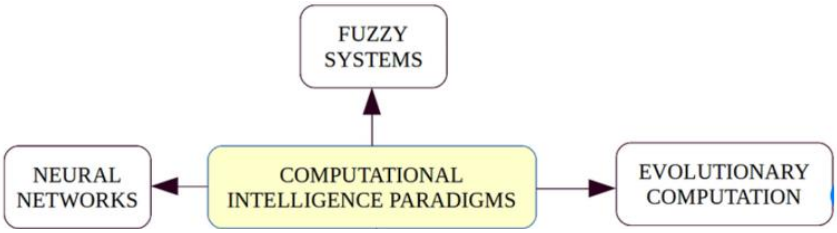
Multivariate Data Analysis:

Sometimes data can be complicated, and each data point could be made up of many different variables. This type of data is known as multivariate data. When in this state, it can be extremely difficult to visualize and interpret. Some techniques that can allow users to extract beneficial information are discussed.

- Principle Component Analysis (PCA): PCA can be thought of as a method for re-expressing the data to reveal its internal structure. It is essentially a dimension reduction technique. The primary benefit of PCA is that it can summarize data through a linear combination consisting of less variables. The most important variables are highlighted in this technique saving valuable time when analysing data.
- K-Nearest Neighbour (KNN): KNN is a classification technique. It is a form of supervised method where the algorithm is 'taught' and once it is taught sufficiently, the algorithm should be able to classify new inputs. It is an easy to implement algorithm. KNN makes no assumption on the spread of the data. Data points are classified based on the K nearest data points. KNN could look at associating different software engineers into groups based on a combination of their attributes.

Computational Intelligence:

Computational Intelligence can be used as a technique to gain better insight into the data provided in the many systems and algorithms outlined previously. It's made up of techniques that truly push the boundaries of what insight we can gain from data. Traditionally the three main pillars of CI have been Neural Networks, Fuzzy Systems and Evolutionary Computation. (CI, 2018)



- Neural Networks: Using the human brain as its source of inspiration, artificial neural networks are massively parallel distributed networks that can learn and generalize from examples. Some of the applications of neural networks that are currently being explored are self-driving cars.
- Fuzzy Systems: Using the human language as its source of inspiration, fuzzy systems model linguistic imprecision and solve uncertain problems based on a generalization of traditional logic, which enables an approximate reasoning performance. Essentially, things that may be 'partially true' can included in problem modelling not possible with the use of traditional logic.
- Evolutionary Computation: Using the biological evolution as a source of inspiration. Evolutionary computation solves optimization problems by generating, evaluating and modifying a population of possible solutions. One of the many developing and fascinating applications of evolutionary computation include the algorithms that will 'evolve' your voice so it sounds the way you want it when you record yourself or speak through technology such as a microphone.

The Future of Algorithms, 'Evolution vs Revolution':

The approaching horizon of algorithms approaching human level intelligence and eventually eclipsing humans has been perennially twenty years away. The advancement of algorithms is an evolutionary process as opposed to the widely touted revolutionary process. If you boil down the aforementioned algorithmic approaches at their heart they are all essentially doing the same thing, searching for patterns in large sets of data. Algorithms continue to advance and improve results by applying more powerful algorithms onto a larger set of data, the order of growth has been of the same order for decades, there is nothing remarkable about the current improvements in algorithmic quality we see nowadays.

It is important to distinguish between computational power which has been astronomically improving, and algorithms. Computational power is subject to Moore's law and has been for decades. Alternatively, advancements in algorithmic complexity have been oscillating between a linear and sub-linear progression rate over the last 30 years. Take for example neural networks, which while gaining complexity and usefulness, the fundamental technology has existed since the 1980's. Ultimately to achieve a step-change in the richness of algorithmic output, breakthroughs in both computational power and algorithmic complexity are necessary which look unlikely to occur any time soon. (FT Podcast Tech Tonic, 2018) (No Silver Bullet)

Ethics:

Ethics are a complicated and controversial subject matter when it comes to any topic. Software engineering is no different. Recent scandals such as Facebook's involvement with Cambridge Analytica have pushed the software engineering ethical conversation into the mainstream news. Now everyone has an opinion when it comes to ethics. It is becoming imperative for companies to take a stand with regards to their ethics. There are a wide variety of approaches companies can take. At one end of the spectrum are companies who are led by the behaviour of those at the top and others are expected to follow suit. At other companies, the ethical conversation is an open one, where employees of all levels can feel free to share their views on the direction their company should take. (Gino, 2018)

In May 2018, the European Union released the General Data Protection Regulation (GDPR) to modernize previous data protection laws. GDPR is seen as more aggressive act and strengthens individual's data protection from corporations. All companies now require consent for mining and storing any persons' data. This can be seen in the wave of pop-ups requests permission when entering a web page. Companies are now required to be transparent about the data they hold and how they exploit

this data. Tough penalties will be issued should companies breach these new regulations. A company could be fined up to 4% of their revenue. GDPR applies to inhouse data collection, employees will have to consent to the use of their data. (Citizens Information, 2018)

The GDPR lists 8 data subjects rights but there are three which are most relevant to software engineers:

1. Right to Access: Data subjects have the authority to access any and all data an organization may be holding relating to you.
2. Right to Rectification: Data subjects must be able to amend any incorrect information an organization may be holding relating to you.
3. Right to Erasure: Data subjects have the right to request deletion either in its entirety or in part of the data held in relation to them.

When software engineers begin ideating, they must ensure all the features mentioned above are incorporated into the system. Data must be stored in a way which allows easy and frequent access by the consumer. This will require a change in the structure of a lot of software and related systems.

There are undoubtedly many benefits which come with the increased capabilities of software engineers. There is a price though. We must ask ourselves at what cost are we willing to advance? Misinformation is becoming increasingly common. Who bears responsibility for this? The individual or the company. Developers will be met with constant trade-offs between rich analytics yielded from troves of data and the autonomy of an individual. There is a lot of disagreement on how best to hold data collectors accountable. While the GDPR will go some way to easing these concerns, it doesn't answer the question of how much is too much data, it solely provides increased protection against the misuse of data.

Software engineering companies and individuals developing at these companies will no longer be tasked with the single goal of creating software. They must now act as moral safeguards, adding to the complexity of an already challenging profession. Integrity will become an intrinsic metric in the measuring of a software engineer. (Somerville, 2011)

Conclusion:

Software engineering is an extremely complex process and one that poses an infinite number of questions. Not all these questions have been solved and definite answers are never likely to be reached. Identifying goals and structuring a specific system prior to implementation. Firms must identify which of the many computational platforms are most synchronized with their own company.

This report has enabled a consideration of the place software engineering will hold in the future. As developed countries continue to plateau, how do they unlock future economic growth? The key to this will be enhancing the productivity of knowledge work and service business. Technological advances will be the driver. These advances will be led by software engineers.

Bibliography

(Roock, 2010)

https://www.scalablepath.com/blog/7-qualities-that-differentiate-a-good-programmer-from-a-great-programmer/

(Osetskyi, 2018)

https://dzone.com/articles/what-technical-debt-it-and-how-to-calculate-it

(Collewet, 2017) IZA Institute of Labour Economics

http://ftp.iza.org/dp10722.pdf

(Humphrey, 2000)

https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf

https://ieeexplore.ieee.org/document/1231611

(YOU TUBE, 2011)

https://www.youtube.com/watch?v=0g477Mhif0k

(Shrestha, 2010)

https://www.sourceallies.com/2010/02/sonar-code-quality-analysis-tool/

(CODE CLIMATE, 2018)

https://codeclimate.com/about/

(Brownlee, 2017)

https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

(CI, 2018)

https://cis.ieee.org/about/what-is-ci

(Gino, 2018)

https://hbr.org/2011/03/talking-about-ethics-how-we

(Citizens Information, 2018)

http://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_of_general_data_protection_regulation.html