Software Engineering

Measurement and Assessment

1. Abstract

This report will outline the ways in which the software engineering process can be measured and assessed. In particular, it aims to address this subject in terms of measurable data, the computational platforms available, the algorithmic approaches available and the ethics of such analytics.

1. Introduction

Before exploring the measurement and assessment of software engineering, we must first understand what is meant by the term ‘software engineering.’ Software engineering is the application of engineering principles to the development of software; focusing on design, development and maintenance.   
  
The origins of software engineering date back to the 1960s. It came about in response to the drastic advancements in the capabilities of computers that occurred during the 1950s and 60s. These rapid developments were not matched with the necessary research into how best to approach the development of programs that could fully power these new capabilities. This resulted in chronic failures by large software projects to meet schedules and remain within budget. The root of the problem was that the individualistic approach to program development would not work for larger and more complex projects. This catastrophe was known as the ‘software crisis.’

It was at a conference in Germany in 1968, which was being held to discuss the ‘software crisis’, that the concept of ‘software engineering’ was first proposed. Today, the field of software engineering continues to progress at an immense rate, and as stated by Ian Sommerville, is “critically important technology for the future of mankind.”

Since the 1960s, it has become apparent that larger scale projects demand a team that can work fluidly and flexibly. At the outset of a project, it is often unclear as to what exactly the client wants. Being able to adapt and change the project as it progresses is central to success. Moreover, larger projects require more programmers which in turn results in a higher need for coordination and collaboration in the workplace.

1. Data Measurement

As you would expect find in any other industry, managers of software engineers need a clear idea of how their team is performing. However, unlike other industries (such as retail), it can be difficult to objectively measure the output of software engineers. Managers not only need to monitor the output of their team, they also need to answer questions such as “How much effort is required to complete a project?”, “How long will a project take?” and “What is the expected cost of a project?” In order to effectually answer these questions, management need access to the appropriate data.

* 1. Measuring Code

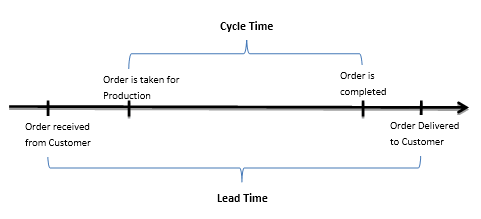
When evaluating the productivity of development teams, what metrics matter most? There are a number of metrics to consider, from time sheets to customer satisfaction, however in this section I will examine some of the most popular metrics in measuring code quality and the productivity of a developer.

Agile Metrics

Agile metrics are used to enhance the software development process. Metrics such as lead time and cycle time are considered.

Lead time determines the time taken to generate ideas, develop and deliver a software product. Naturally, management will strive to have as short a lead time as possible. Partly in an effort to please customers, but also because the longer a project goes on, the more expensive it will be. As we often are told, time is money. However, developers can often be faced with a tradeoff between speed and producing a high quality project. This presents a potential flaw with using lead time as a metric and therefore is should be used alongside other metrics.

Cycle time is the part of the lead time dedicated purely to the production of the software, as illustrated by the diagram below.



Production Metrics

Production metrics assess the scope of projects completed and measures the productivity of software development teams. It has quantifiers such as active days, tasks scope and code churn.

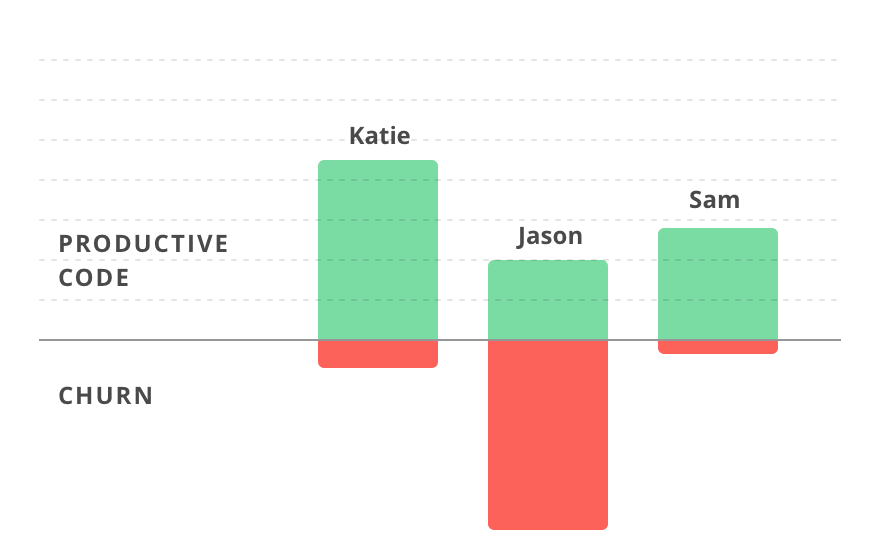
Active days refers to the time dedicated by a developer to developing the code. This excludes time spent planning or on other minor activities. This metric can be especially useful for identifying hidden costs.

Tasks scopes refers to the volume of code an individual programmer can deliver annually. This metric can be applied when quantifying the number of developers needed to make up a software development team.

Code churn is the percentage of a software engineer’s own code that represents an edit to their own recent work. It’s typically measured in Lines of Code (LOC) that were modified, added and deleted over a short period of time. Spikes in code churn can help identify problems early on in the lifetime of a project, as it describes how much time a developer spends on a feature while making limited progress.

The metric of code churn is undoubtedly an excellent indicator of performance, provided we look into the data to uncover the reasons of why it is notably high or low. A high churn rate can happen for a number of reasons, but is often an indicator that something is off within the development process. For example, it can be the symptom of a perfectionist needlessly working on a feature incessantly, or could be attributed to indecisive stakeholders, who are constantly asking for features to be slightly changed. Typically, a high level of code churn is a sign of a less stable project. High rates of code churn are typical at the outset of a project, or when many changes have been implemented. One should expect the level of code churn to decrease as the project approaches its release.

The diagram below illustrates an example of code churn. From this basic example, we can see that Jason wrote a lot of code, however the amount of productive code he delivered is significantly less than what Katie delivered.



Technical Debt

The metaphor ‘technical debt’ was devised by Ward Cunningham, and was described by him as follows: “*Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."* (Cunningham, 1992).

In essence, the concept of technical debt describes the additional development work programmers have due to implementing a solution which it is simpler in the short term, but is not the overall optimal solution. Just like financial debt, one must be careful when incurring technical debt as it can be detrimental to a business in the long run. The debt will accumulate interest over time and at some point this will have to be paid off through extra work. Although it is inevitable for projects to hold some level of technical debt, management should endeavour to keep it minimal.

Source Lines of Code

Source Lines of Code (SLOC) is a metric regularly used to measure the size and complexity of a software project. It is commonly used to predict the time and effort required to develop a program. It is also used to estimate the programming productivity once the software is produced. There are two major types of SLOC measures: physical and logical. The most common definition of physical SLOC is a count of lines in the text of the program’s source code including comment lines. Logical SLOC attempts to measure the number of “statements”. Physical SCLOC tends to me much easier to measure than logical SCLOC. SLOC is considered a poor measure of productivity for a number of reasons. One being that SCLOC will vary depending on the formatting conventions and language used. Furthermore, it is understood that well designed code tends to be shorter as it eliminates duplications. Therefore, measuring SCLOC promotes longer, confusing code that is difficult to maintain. In reality, organisations should be aiming to produce code that is neat and concise.

Number of Commits

This refers to the number of times a developer contributes to their code. This can vary greatly depending on the developer. How often a developer commits does not explicitly distinguish a good software engineer from a poor one. It does, however, provide us with a very good idea of if a developer is active, when they are most active and essentially, which developers are doing the most work. This data can be of considerable help to management teams when deciding on promotions, redundancies and progress reviews.

* 1. Taxonomy

Taxonomy originates from the Greek dialect. “Taxis” meaning arrangement of division and “nomos” meaning law. Taxonomy is defined as the science of classification according to a predetermined system, with the resulting catalogue used to provide a conceptual framework for discussion, analysis or information retrieval. The key to taxonomy is ‘semantic architecture,’ as it is all about maintaining a consistent structure.

In order for all programmers in a group to be able to effectively switch between programs, as well as comprehend their capabilities, having a shared taxonomy is vital. For example, there may be a certain style followed when naming variables which every programmer should follow. Taxonomy also improves performance of data collection and analysis.

Another application of taxonomy would be Master Data Management (MDM). MDM is comprised of a set of processes and tools that consistently define and manage the non-transactional data entities of an organization. MDM results in generating more effective business knowledge, as it presents greater control over consistent reference data and the ability to manage the relations between data entities. 

1. Computational Platforms Available

Code review is a crucial part of the development process as it makes you code stronger and reduces the risk of bugs. The question of if high level quantitative questions, regarding the performance of engineers, can be answered by processing low level digital data is a highly debated topic.

One methodology put forward is the Personal Software Process (PSP). PSP is a structured software development process that was devised to aid understanding and improve the performance of engineers by using a “disciplined, data-driven structure.” PSP was created by Watts Humphrey in the hope of reducing the number of defects in the common developer’s work as well as help them manage the overall quality of projects. However, PSP has been subject to some criticism and is not universally accepted. Some engineers believe PSP is good in theory but not in practice, as it is too inflexible and time consuming.

Thankfully, there are many computational platforms available for conducting code review. Some of the most popular companies that supply software to collect and analyse measurable data are Code Climate, Gitcop, Gitcolony, Codebeat, Semmle, Teamscale, Black Duck, Codebrag and Phabricator. In this section, I will discuss three of the most popular platforms in further detail.

* 1. Code Climate:

Code Climate is a privately held software company, founded by Bryan Helmkamp in 2011. It offers analysis tools to aid software engineers in the improvement of code quality. It is an open, extensible platform and is the most popular (and expensive) option currently available to developers. To illustrate its popularity, consider that Code Climate analyses trillions of lines a code every week and is used by over 100,000 projects. It enables automated analysis of intricate codebases written in an array of languages, such as JavaScript, Python, PHP, Ruby on Rails. Code Climate helps developers control the quality of their code, by combining fully-configurable test coverage and maintainability data throughout the development workflow. In doing so, it makes quality improvements explicit, continuous, and ubiquitous. Having test coverage alongside code quality empowers a team of developers to make more informed decisions early in the development process. This advantage leads to increasingly maintainable code with enhanced usability.

Some of its most impressive features are:

* Automated git updates, which means that Code Climate runs every time you push a new commit.
* Activity Feeds: Code Climate provides up-to-the-minute data enabling developers to see when and how code changes.
* Team Sharing: Code Climate provides direct access to the entire development team. This maximizes code visibility across projects.
* Hotspots: Code Climate correlates code quality against areas of high churn. Therefore your effort can be focused where it is most needed.
* Security Dashboard: A systematized catalogue of your app's weaknesses is provided, including when they were first introduced and how to address them.
* GitHub Integration: Code Climate provides post-receive hooks for instant updates and GitHub drilldown links.
* Test Coverage Integration: Surfacing coverage information at the repo, class, and source listing level.
* “Private, Safe, and Secure.” All data is private by default, with SSL encryption throughout.  
  1. Codacy:

Codacy is another automated code review tool, and is considered to be Code Climate’s closest competitor. Founded in Lisbon in 2012, by Jaime Jorge and Joao Caxaria, Codacy is in many ways similar to Code Climate. Codacy has been used by hundreds of companies, many of which are household names. For example Adobe, Deliveroo and Paypal. The software can be installed on-premise or accessed in the cloud, and according to Jaime Jorge, Codacy “helps developers optimise around 30% of their code review time,” resulting in engineering teams increasing their efficiency by 6%. Codacy strives to centralize the most prevalent problems, alerts and metrics and integrate them into developer’s workflow. It provides review of coding security concerns, programming style violations, as well as providing examples of best practices and many other metrics a business may find useful. It is a provider of many cutting-edge statistics such as churn, complexity, duplication, number of lines of code and even gives an estimation for the time expected to remedy a problem.

* 1. Codebeat:

Codebeat is another popular data analytics tool frequently used by software developers, which has advanced significantly in recent times. Known for taking user’s feedback into consideration, Codebeat is a hosted, affordable, proprietary tool. Codebeat is focused on the needs of individual developers as well as development teams, striving to provide real business value. Their toolchain is tailored to help developers focus on the most important aspects of code review, such as architecture and business logic. Rather than using open source linters by combining existing projects, Codebeat opted to generate their own algorithms from scratch. Codebeat is currently supports ten languages including Swift, Go, Ruby, Python and Java. Codebeat is recognized for providing excellent support systems and has a smart but well-documented API, which facilitates management. Although there are many upsides to Codebeat, it is still associated with a few drawbacks, such as lack of explanation of code duplication.

1. Algorithmic Approaches Available:

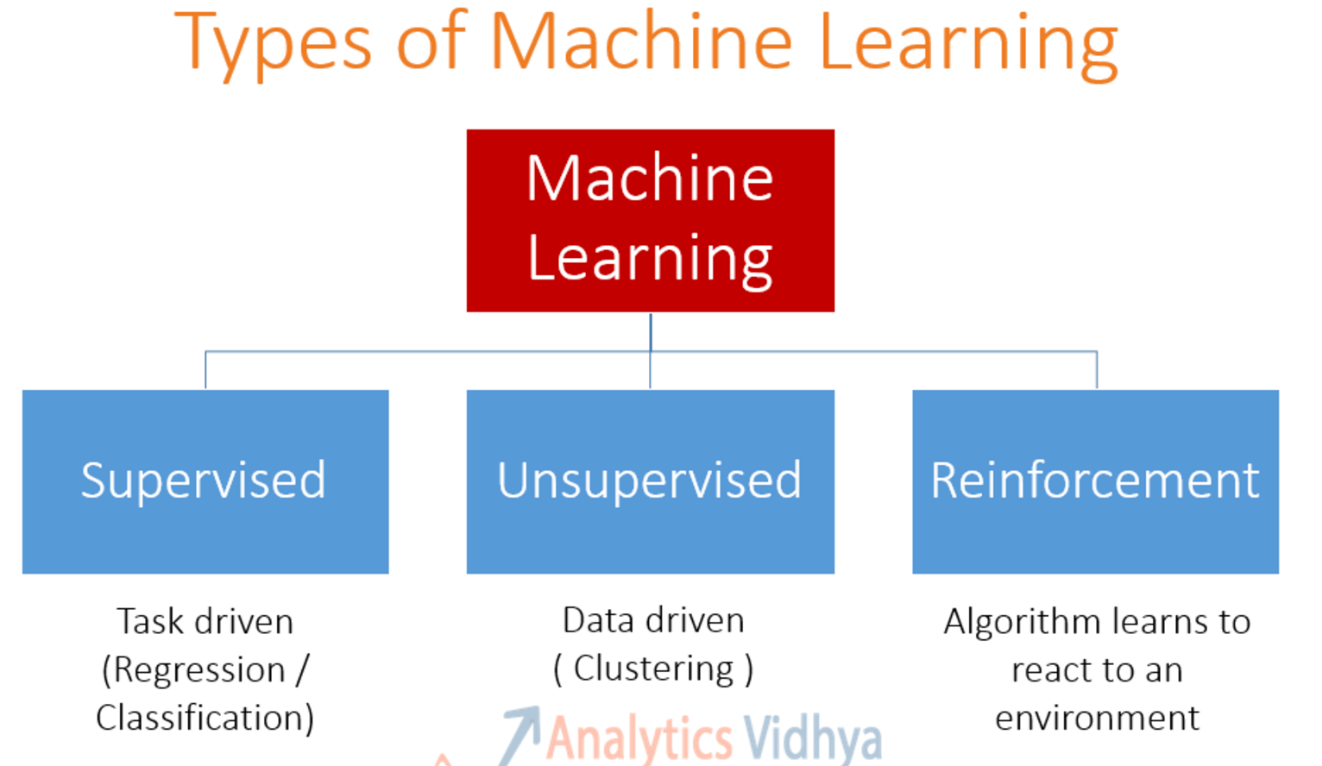
In this section of the report, I will discuss a selection of algorithms available for analyzing the data we have collected for the purpose of measuring software.

5.1 Machine Learning Algorithms

Machine learning was derived from the concept of pattern recognition and the theory that computers can learn without being programmed to carry out a specific task. This led to the question of whether computers could learn from data, which resulted in what we now refer to as machine learning.

Machine learning is fueling revolutionary innovations in artificial intelligence (AI). There are endless commercial applications of machine learning. For example, it can be applied to personalizing marketing campaigns and is frequently used for predicting recommendations to customers based on big data. The public interact with machine learning on a daily basis, however, many of the applications of machine learning go on unknown to the average person. For example, Twitter is now displaying algorithmically curated feeds. This is done through AI evaluating tweets and ‘scoring’ them according to various metrics. Tweets are then displayed in order of score rather than chronologically. Machine learning presents many advantages to commercial organizations as it can greatly enhance an organization’s chance of detecting lucrative opportunities.

Machine learning algorithms can be split into three general categories: supervised learning, unsupervised learning and reinforcement learning.



Supervised Learning:

Supervised methods assume a given structure within the data, for example Decision Tree Analysis, K Nearest Neighbors, Regression or Logistic Regression. In supervised learning, there is an output variable and a set of input variables. Using the input variables, a function is generated to map inputs to desired outputs. The goal being that when there is new input data, its corresponding output variables can be predicted.

A training data set is used to generate the mapping function. The training process only ends when the algorithm reaches a specified level of accuracy. This is why we refer to these methods as supervised learning.

Example: Decision Tree Analysis

Decision tree analysis uses a tree-like graph or model of decisions and their possible outcomes, including the chance-event outcomes, resource costs and utility. Decision tree analysis facilitates a structured and systematic approach to establishing logical and rational conclusions.

Unsupervised Learning:

Unsupervised learning is when there is only input data, and no known output data. Therefore we must discover a structure from the data alone. The goal of unsupervised learning is to learn more about the data by modeling the underlying distribution in the data. We refer to this technique as unsupervised because there is no “correct answer” and we completely depend on the algorithm to uncover the structure of the data.

Examples of unsupervised learning include K-means clustering, the Apriori algorithm and Principle Component Analysis.

Example: Principle Component Analysis (PCA)

PCA is a data-reduction technique which is used to represent complex data sets in a reduced number of dimensions so that the structure of the data can be more easily visualized. It converts a set of correlated variables into a reduced set of uncorrelated variables.

Reinforcement Learning:

Reinforcement Learning adopts a trial and error approach to learning by using feedback from its own actions. As humans, we learn through interaction. Reinforcement Learning is simply a computational approach of learning through actions.

The learner is not instructed as to what actions to take, but instead must discern which actions return the most reward by testing them. Simple reward feedback is required for the agent to learn its behavior; this is known as its reinforcement signal. This behavior can be learned as a once off, or can continue updating as time goes by. The ideal behavior (i.e. the behavior that maximizes reward) will occur if a Reinforcement Learning algorithm converges to the global optimum. This is possible but is not always achieved.

A limitation of Reinforcement Learning is that often, it is too expensive in terms of memory. The more complex the problem, the more memory that will be required. Examples of reinforcement machine learning algorithms are Markov decision process, Dynamic Programming and Monte Carlo Methods.

Example: Markov Decision Process (MDP)

MDP is a framework used for making decisions in a stochastic environment. There are four components of an MDP model: states, actions, effects of the actions and the immediate value of the actions. The algorithm is positioned in a particular state and is provided with the actions it can take, along with the relevant consequences. It must then make decisions and define the new state it is in. Each time the algorithm makes a poor decision, it learns from it and will not repeat the mistake. This process repeats until the solution is found. The solution to an MDP is referred to as a policy. It outlines the optimal action for each of the states.

1. Ethical concerns:

Ethics is defined as the “moral principles that govern a person’s behavior or the conducting of an activity,” (Oxford Living Dictionaries, 2018). As society’s dependency on computers continues to rise, software engineers have the opportunity to either contribute to society in a lasting way or cause great harm. Therefore, it is fundamental to the reputation and influences of software engineering that developers commit themselves to making it a respected and beneficial profession. Due to the nature of software development, it is accepted that the practicing software engineer acquires obligations to users, clients, colleagues, the organization for which he/she works for as well as the discipline of software engineering.

In light of this, the Association for Computing Machinery (ACM) created a code of professional practices within the industry during the 1990s, which was updated as recently as July 2018. However, this is merely a company-specific code and is not legally enforced. This code contains eight core principles regarding the behavior and decisions of software engineers, as well as those being trained. Despite these efforts, new research from North Carolina State University suggests that this code is not affecting the decisions made by software developers.

From a legal perspective, organizations must be increasingly cautious regarding data protection. Organizations across Europe have felt the effects of General Data Protection Regulations (GDPR) which came into effect in May this year. These changes in regulation not only impose stricter laws on organizations that hold personal data, but also acted as a reminder to society as a whole of the importance of protecting our own privacy. As a result of GDPR, organizations seeking to measure their software engineering process must do so in a judicious and legal fashion, or else face being fined the larger of €20 million or 4% of annual turnover.

I believe that success in software development relies almost exclusively on the quality of the engineers involved. By quality, I do not only refer to the standard of the software they produce, but also the quality of their judgement and the decisions they make. There is no ‘right answer’ as to how the ethics of software engineering can be handled, as it is an obscure and complex subject. However, it is clear that if we treat this subject with complacency and fail to respond to the challenges it is presenting, we will face problems of increasing scale and impact.

A high profile example of when ethics were left by the wayside is the Volkswagen emissions scandal in 2015. Car owners across the globe were stunned when it emerged that Volkswagen had programmed 11 million cars globally to cheat emission standards. Another international example of the immoral use of data analytics is the Cambridge Analytica scandal. Earlier this year it hit headlines that the political data firm Cambridge Analytica had harvested the personal data of millions of Facebook users without consent and used this data for the manipulation of election votes. Cases like these are detrimental to the reputation and credibility of the field of software engineering. Therefore, there is a pressing need for clearer and stricter laws and legislation surrounding data analytics. In creating these laws and legislation, the subject of ethics will be lifted from the grey area in which it currently lies.

There is a need for a code of practice which is enforced by a professional body. Admittedly, a one-size-fits-all code of practice may be difficult to produce, as the field of software engineering is powered by diverse thinking. Moreover, the ethical approach to data measurement needs to be a core subject in studying the profession of software engineering.

1. Conclusion:

To surmise, measuring and assessing software engineering is a complex process around which there exists many questions and debates. This report has outlined how software engineering can be measured and assessed through a number of metrics, while also providing an insight into existing platforms and algorithmic approaches. Finally, the ethical implications associated with the field of software engineering have been addressed.

# References

Artificial Intelligence Depot., 2002. *Reinforcement Learning.* [Online]   
Available at: http://reinforcementlearning.ai-depot.com/  
[Accessed 17 11 2018].

Arvind Narayanan, S. V., 2014. *Why Software Engineering Courses Should Include Ethics Coverage.* [Online]   
Available at: https://cacm.acm.org/magazines/2014/3/172501-why-software-engineering-courses-should-include-ethics-coverage/fulltext  
[Accessed 17 11 2018].

Baron, J., 2018. *Forbes: We Need To Work Harder To Make Software Engineering More Ethical.* [Online]   
Available at: https://www.forbes.com/sites/jessicabaron/2018/10/17/we-need-to-work-harder-to-make-software-engineering-more-ethical/#4221f4a750cc  
[Accessed 16 11 2018].

Bhatt, S., 2018. *5 Things You Need to Know about Reinforcement Learning.* [Online]   
Available at: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html  
[Accessed 17 11 2018].

Brooks, F. P., 1987. *No Silver Bullet - Essence and Accident in Software Engineering.* University of North Carolina: s.n.

Brownlee, J., 2016. *Supervised and Unsupervised Machine Learning Algorithms.* [Online]   
Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/  
[Accessed 17 11 2018`].

Code Climate, 2018. *About Us.* [Online]   
Available at: https://codeclimate.com/about/  
[Accessed 16 11 2018].

Codebeat, 2018. *How is Codebeat Different.* [Online]   
Available at: https://hub.codebeat.co/docs/how-is-codebeat-different  
[Accessed 16 11 2018].

Codebeat, 2018. *What languages does codebeat support?.* [Online]   
Available at: https://hub.codebeat.co/docs/language-supported  
[Accessed 16 11 2018].

Crunchbase, 2018. *Codacy.* [Online]   
Available at: https://www.crunchbase.com/organization/codacy#section-overview  
[Accessed 16 11 2018].

Earley, S., 2011. *Why Taxonomy is Critical to Master Data Management (MDM).* [Online]   
Available at: http://www.earley.com/blog/why-taxonomy-critical-master-data-management-mdm  
[Accessed 13 11 2018].

Erickson, D. & Staheli, D., 2016. *Code Churn Excel Report.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/devops/report/excel/code-churn-excel-report?view=tfs-2018  
[Accessed 13 11 2018].

Gibson, P., 2018. *Software quality, metrics, tests, processes.* [Online]   
Available at: http://www-public.imtbs-tsp.eu/~gibson/Teaching/CSC5524/CSC5524-PSP.pdf  
[Accessed 20 11 2018].

Gotterbarn, D., 2018. *Ethical Considerations in Software Engineering.* [Online]   
Available at: http://csciwww.etsu.edu/gotterbarn/artge2.htm  
[Accessed 17 11 2018].

Gwyn Topham, S. C. C. L. P. S. a. M. F., 2015. *The Volkswagen emissions scandal explained.* [Online]   
Available at: https://www.theguardian.com/business/ng-interactive/2015/sep/23/volkswagen-emissions-scandal-explained-diesel-cars  
[Accessed 17 11 2018].

Hatsenko, O., 2018. *Top 7 Software Quality Metrics That Matter.* [Online]   
Available at: https://diceus.com/top-7-software-quality-metrics-matter/  
[Accessed 13 11 2018].

Helmkamp, B., 2013. *Test Coverage and Code Quality, Better Together.* [Online]   
Available at: https://codeclimate.com/blog/test-coverage-and-code-quality-better-together/  
[Accessed 16 11 2018].

IEEE, 2018. *Software Engineering Code of Ethics.* [Online]   
Available at: https://www.computer.org/web/education/code-of-ethics  
[Accessed 17 11 2018].

Karck, E., 2011. *The Software Crisis: A Brief Look at How Rework Shaped the Evolution of Software Methodolgies.* [Online]   
Available at: https://blogs.msdn.microsoft.com/karchworld\_identity/2011/04/04/the-software-crisis-a-brief-look-at-how-rework-shaped-the-evolution-of-software-methodolgies/  
[Accessed 13 11 2018].

Oxford Living Dictionaries, 2018. *Oxford Living Dictionaries.* [Online]   
Available at: https://en.oxforddictionaries.com/definition/ethics  
[Accessed 16 11 2018].

Ozimek, Ł., 2017. *Comparison of Automated Code Review Tools: Codebeat, Codacy, Codeclimate and Scrutinizer.* [Online]   
Available at: https://www.netguru.co/blog/comparison-automated-code-review-tools-codebeat-codacy-codeclimate-scrutinizer  
[Accessed 16 11 2018].

Pfleeger, S. L., Jeffery, R., Curtis, B. & Kitchenham, B., 1997. *Status Report on Software Engineering.* [Online]   
Available at: https://pdfs.semanticscholar.org/169b/6b0fffa99ed5115f970f61701ded2c7611b0.pdf  
[Accessed 13 11 2018].

POMDP, 2018. *Brief Introduction to MDPs.* [Online]   
Available at: http://www.pomdp.org/tutorial/mdp.html  
[Accessed 18 11 2018].

Psaila, S. B., 2018. *Cambridge Analytica explained: The facts, implications, and open questions.* [Online]   
Available at: https://dig.watch/trends/cambridge-analytica  
[Accessed 17 11 2018].

PVS-Studio, 2018. *Source Lines of Code.* [Online]   
Available at: https://www.viva64.com/en/t/0086/  
[Accessed 14 11 2018].

Ray, S., 2017. *Essentials of Machine Learning Algorithms (with Python and R Codes).* [Online]   
Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/  
[Accessed 17 11 2018].

Shewan, D., 2018. *WordStream: 10 Companies Using Machine Learning in Cool Ways.* [Online]   
Available at: https://www.wordstream.com/blog/ws/2017/07/28/machine-learning-applications  
[Accessed 17 11 2018].

Simonini, T., 2018. *An introduction to Reinforcement Learning.* [Online]   
Available at: https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419  
[Accessed 17 11 2018].

Stackshare, 2018. *Code Climate.* [Online]   
Available at: https://stackshare.io/code-climate  
[Accessed 16 11 2018].

Sutton, R. S. & Barto, A. G., 2017. *Reinforcement Learning:,* s.l.: s.n.

TechCentral, 2018. *Should software developers have a code of ethics?.* [Online]   
Available at: https://www.techcentral.ie/software-developers-code-ethics/  
[Accessed 17 11 2018].

techopedia, 2018. *Software Engineering.* [Online]   
Available at: https://www.techopedia.com/definition/13296/software-engineering  
[Accessed 13 11 2018].

techopedia, 2018. *Technical Debt.* [Online]   
Available at: https://www.techopedia.com/definition/27913/technical-debt  
[Accessed 14 11 2018].

The Standish Group, n.d. *Standish: Project Success Rates Improved Over 10 Years.* [Online]   
Available at: http://www.softwaremag.com/L.cfm?doc=newsletter/2004-01-15/Standish  
[Accessed 13 11 2018].

Thompson, B., 2018. *Why Code Churn Matters.* [Online]   
Available at: https://blog.gitprime.com/why-code-churn-matters/  
[Accessed 13 11 2018].