MEASURING SOFTWARE ENGINEERING

CONTENTS

Requirement ... 1
introduction ... 2
1. Software engineering process ... 2
2. Measurable data ... 3
Lines of code ... 4
Lead time ... 4
Git Commits ... 4
Code churning ... 5
Sprint burndown ... 5
3. Computational Platforms Available ... 5
Personal Software Process ... 6
Leap ... 6
Hackystat ... 6
4. Algorithmic approaches available ... 7
5. Ethical Concerns ... 10
Data collection ... 11
Data usage ... 11
Data sovereignty ... 12
6. Conclusion ... 12
Sources ... 13

REQUIREMENT

To deliver a report that considers the ways 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 ethics concerns surrounding this kind of analytics.

INTRODUCTION

For us to understand how software engineering is measured and assessed, we must first understand what exactly it is. Software engineering is a direct sub field of engineering and involves the usual steps applied by an engineer, to design, develop and maintain software. The exponential growth of software engineering in the last 50 years has brought the need for proper measurement and assessment of this process. In this report I will discuss the different ways in which software engineering processes can be measured and assessed in terms of:

1. Measurable data
2. Computational platforms
3. Algorithmic approaches
4. Ethical concerns surrounding this sort of analysis

I will begin by outlining the phases of the software engineering process, in order to properly understand what each metric is investigating in the stages of development.

1. SOFTWARE ENGINEERING PROCESS

The software development lifecycle describes the phases of the development process and the order in which those phases are executed. Each phase produces deliverables that are required by the next phase in the cycle.

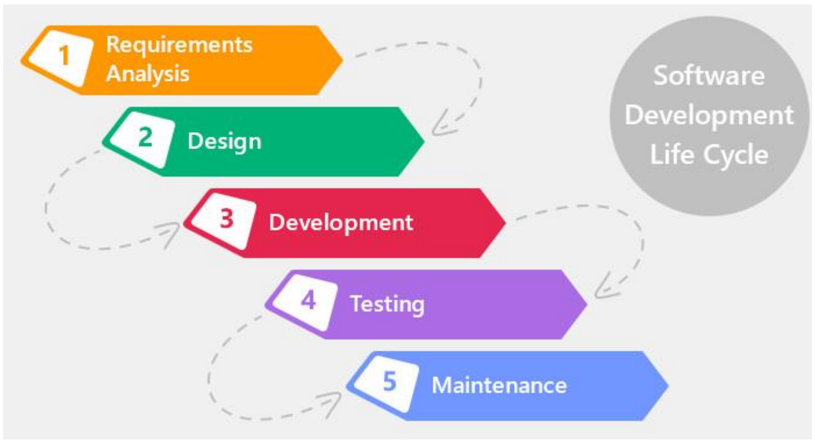


Figure 1 (What are the Software Development Life Cycle (SDLC) phases?, 2020)

The six phases included in the software development process are:

1. **Requirement Analysis:** In this phase business requirements are gathered. The project managers meet with stakeholders and determine the needs of system. The requirement specification document is produced which will act as the guidelines for the next phase of the model.

2. **Design:** In this phase the system and software design are prepared from the requirement specifications which were outlined in the requirement specification document. The system design specifications serve as input for the next phase of the model. In this phase, test strategy is devised, indicating what needs to be tested and how it should be tested.
3. **Implementation:** This stage consists of the most obvious aspect of the software engineering process – the coding. On receiving system design documents, the work is divided up and coding is started. This is the longest phase of the software development life cycle however if the first two phases were done correctly the implementation phase should be a smooth task.
4. **Testing:** This phase of the software engineering process revolves around checking for bugs and errors in programs.
5. **Deployment:** After successful testing the product is delivered to the customer for their use. The first prototype of the system will be beta tested by customers in order to establish any initial bugs and the error handling of the system. These issues are sent to the engineering team who will solve them before the finished product is sent out to the customers.
6. **Maintenance:** This is the process where the developed product is taken care of. Maintenance ensures software can deal with newly discovered problems and requirements in the future after the initial development of the software.

The software engineering process is a long one, consisting of different stages in order to ensure that each aspect of the process can be carried out in an efficient and hassle-free manner. Essentially, every phase of the process is there to make the next phase easier for the relevant team carrying out the task, be it developers, market researchers or strategists. There are many ways in which this process can be assessed in terms of measurable data.

2. MEASURABLE DATA

There is no denying that data is the most crucial component of the software engineering process. Another vital component is measuring this data. There are many factors we must take into consideration when assessing the ability of a software engineer. One such factor is, of course, the quality of their code for which there are many different assessment methods, with many debates around the effectiveness of these methods. Size-orientated metrics are a very direct approach to measurement and are derived by normalizing the quality and productivity measures by considering the size of the product as a metric. Production metrics attempt to measure how much work is done and determine the efficiency of software development. Agile metrics come from the philosophy of agile development and give us insight into how the process is working so areas of improvement can be identified. In this section of the report, I will discuss examples of these metric types.

LINES OF CODE

Lines of code (LOC) was the first software engineering metric, it involves counting the number of lines of code and is an easy and definitive approach. LOC is a size-orientated metric. The reasoning behind LOC is that more lines equate to more work done, in the same way a lumberjack's productivity can be measured by the number of trees they chop. However, any software engineer will agree that more code certainly doesn't imply a better program. LOC rewards inefficient and lazy coding because it discourages the condensing of lines of code. Although less lines don't necessarily mean better code, it is usually more efficient. Another flaw with using LOC is that it doesn't measure effort or complexity of the work done by a software engineer. For example, a software engineer tasked with maintaining a program will have far fewer lines written than the person who initially developed the program and would therefore be seen as less valuable even though their productivity is equal. LOC also doesn't measure the effort, functionality and complexity of a programme.

LEAD TIME

This agile metric measures the time taken between the identification of a requirement and its fulfilment. Essentially how long does it take for a project to reach completion. Lead time can be used to gauge how responsive a software engineer is to their customers and how quickly can a customer get their product.

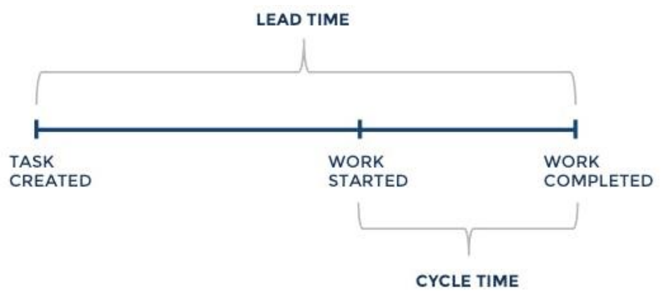
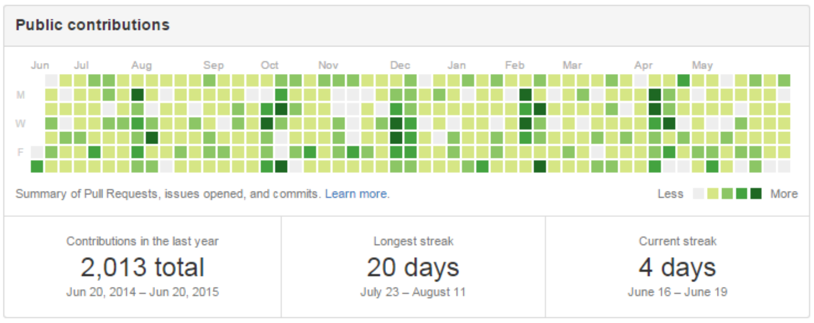


Figure 2 (Tranter, 2020)

GIT COMMITS

If a software engineer or team is using a version control system, the number of commits to a repository by individuals or by the team can used as a metric. Number of commits is a measurement of the amount of time a software engineer spends on their code. This is a good way of measuring a software engineers activity level. However, it is not an acceptable metric to use for productivity as more commits don't equate to better progress.


Public contributions
Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May
M
W
F
Summary of Pull Requests, issues opened, and commits. Learn more.
Less More

Contributions in the last year
2,013 total
Jun 20, 2014 – Jun 20, 2015

Longest streak
20 days
July 23 – August 11

Current streak
4 days
June 16 – June 19

Figure 3 (Franken, 2020)

CODE CHURNING

Code churn is a productivity metric and is defined as the percentage of a developer's code that is an edit to previous code. To compute it, we count the number of lines of code that were added, modified or deleted over a period. Code churn can be considered as non-productive as the same code is just being rewritten. If the code churn on a task increases dramatically it may indicate that the task needs more attention.

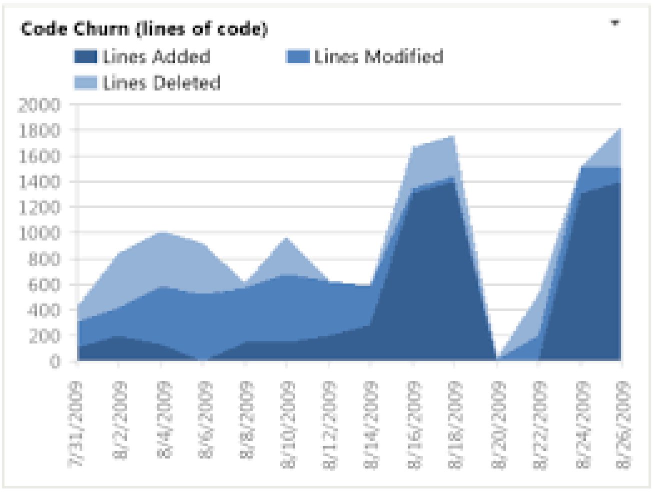


Figure 4 (TFS, 2020)

Code churn will usually fluctuate over time as most coding will be completed in the beginning and decrease closer to deployment. A large code churn could point to unclear requirements, indecisive stakeholders or a difficult problem to solve. Code churn can be valuable metric for evaluating teams and software engineer's workflow.

SPRINT BURNDOWN

A burndown chart is a graphical representation of work to be completed versus the time left to do it in. Burndown charts are often used in agile development methodologies like Scrum. Instead of considering a project as several tasks, the sprint burndown approach uses story points. A story point is an estimation of a software features complexity. Sprint burndown communicates the complexity of the work throughout the development process based on these story points.

3. COMPUTATIONAL PLATFORMS AVAILABLE

After discussing the different types of measurable data in the software engineering process, the next question would be where can work be shared so that it can be accessed and assessed easily. There is now an extensive selection of platforms for companies and software engineers to choose from.

PERSONAL SOFTWARE PROCESS

PSP is a structured software development process designed to improve project estimation and quality assurance by using a disciplined, data driven procedures. This ground-breaking process was created in 1995 by Watts Humphrey, who taught it through practical application. The size, time and defect data of a software project are collected, and a set of analysis are performed. For example, given the system's size estimation, a PSP analysis called PROBE provides an estimation of the time required of a system based on the relationship between size and time on previous projects.

Studies have shown that PSP-style collection and analysis of data by a software engineer can contribute substantial benefits to that individual (Johnson, Kou and Agustin, 2020). While there are many benefits PSP, it is extremely manual in nature requiring developers to fill out forms and fill in checklists regarding tasks they have completed and what went right and wrong. This makes PSP time consuming and prone to human error. Despite these flaws many people believe that PSP is still useful because it can teach useful software engineering skills. However, PSP doesn't have enough of a return outside of teaching and learning.

LEAP

LEAP toolkit was developed in response to the quality problems prevalent in PSP. LEAP stands for Lightweight, Empirical, Automated, and Portable Software Developer Improvement. It combatted these issues by automating and normalising data analysis. LEAP added in regression analysis which was not included in PSP.

LEAP aimed to circumvent the data quality concerns by automating the data analysis. However, since its introduction to the industry it has become abundantly clear that LEAP, just like PSP, is still prone to human error as there is no way to fully automate it since manual input is required.

HACKYSTAT

Hackystat describes itself as "A framework for the collective, analysis, visualisation, interpretation, annotation and dissemination of the software development process and product data." Hackystat is a fully automated computational platform. Software engineers first began using Hackystat by attaching sensors to their development tools. These sensors unobtrusively gather data on the product development process and analyse it on a remote server. Hackystats most interesting feature is its fine-grain data collection which ensures data is collected in real time, tracking developers as they work on the code. This feature is one of the main benefits of Hackystat however many software engineers find it intrusive and are uncomfortable with this level of data being collected.

The nature of this collection allows managers to have a deep understanding of the work that is being done, however, many developers were uncomfortable with the accessibility the platform gave others into their work. (Johnson, 2003).

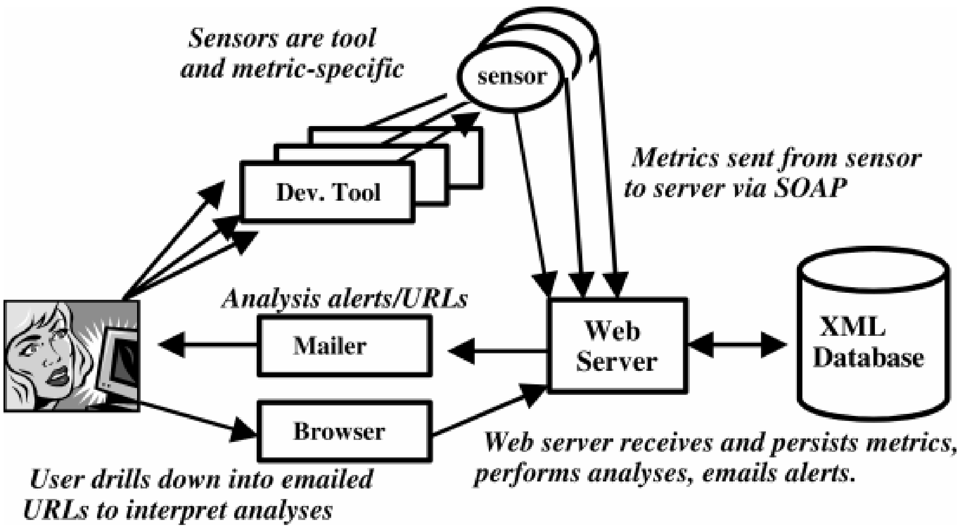

Sensors are tool and metric-specific
sensor
Dev. Tool
Metrics sent from sensor to server via SOAP
Analysis alerts/URLs
Mailer
Web Server
XML Database
Browser
Web server receives and persists metrics, performs analyses, emails alerts.
User drills down into emailed URLs to interpret analyses

Figure 5 (Johnson 2003).

Focusing on what Hackystat does for the developer, it can be used as infrastructure to support professional development, either proprietary or open source, by facilitating the collection and analysis of information useful for quality assurance, project planning and resource management.

What was enticing about Hackystat was the automation. Both PSP and Leap require the software engineer to manually enter the data themselves leading to human error. But Hackystat was met with a whole new set of issues regarding ethics. The push back on this new system was mainly from developers who were being asked to use it. As a result, Hackystat quickly went from being a terrific development in terms of time saving into an unethical platform which software engineers were unhappy with.

These three platforms paved the way for many more after them. Some notable examples include Code Climate, GitHub and GitPrime. This wide selection of platforms has led many software engineers to struggle when making the decision of which one to use.

4. ALGORITHMIC APPROACHES AVAILABLE

There are various algorithmic approaches to analyse data collected during the software engineering process. These range from computational intelligence to artificial intelligence. Machine learning is an element of artificial intelligence. But what exactly is it? Machine learning can be defined as the science of getting computers to learn and act as humans do. It is divided into three sections – unsupervised learning, supervised learning and reinforcement learning.

UNSUPERVISED LEARNING

Unsupervised learning is carried out on data that is unsupervised, meaning the data is purely input data with no corresponding output variables. Its purpose is to uncover previously unknown patterns and the underlying structure of the data. The best use of unsupervised learning is when there is no data on the desired outcomes like determining a target market for a brand-new product that a business has never sold before.

K-Means Clustering

Clustering is considered one of the simplest and most popular unsupervised algorithms. The objective of clustering is to identify subgroups in the data set such that data points in the subgroup are very similar while data points are very different from other subgroups, in order to discover underlying patterns. To do this K-means must find a fixed number of clusters in a dataset that are distinct and don't overlap with one another.

To implement K-Means clustering you assign K random centroids throughout the data. For every data point, assign it to the nearest centroid. For each centroid move it to the average of its assigned data points. Repeat these three steps until the centroid assignment no longer changes. When there are no more changes the algorithm is said to have "converged".

SUPERVISED LEARNING

Supervised learning is a type of machine learning where you have a set of input variables and a set of output variables and you use an algorithm to learn the mapping function from the input to the output. The goal of supervised learning is to approximate the mapping function so well that when you have new input data, you can predict the output for that data. It is called supervised learning because the process the algorithm learning from the dataset is likened the learning process being supervised by a teacher. Unlike unsupervised machine learning, supervised machine learning methods can be directly applied to a regression or classification problem because you know what the values for the output might be, making it possible for the algorithm to be trained.

K-Nearest neighbour (KNN)

Using classification algorithms like K-Nearest Neighbours, we can classify the new data points into the dataset with high accuracy. KNN is one of the simpler techniques used in machine learning. It is commonly used in the industry due to its simplicity and low calculation time. KNN will classify similar data points together in a test set, it will then use this test data to make an "educated guess" on what an unclassified point should be classed as.

KNN is used in image recognition technology and decision-making models. Companies like Netflix and Amazon use this approach in order to make recommendations to their customers. Netflix even offered a prize to the team that could generate the most accurate recommendations from their algorithm. For the most accurate prediction it is important to choose an optimal value for K. The lower the value of K the lower the value of bias -the difference between the true label and our prediction- in the model. If K increases, the bias will decrease, but this will result in a higher chance of error in the test data, causing higher variance. As K increases, the number of neighbours and the complexity of the model also increases.

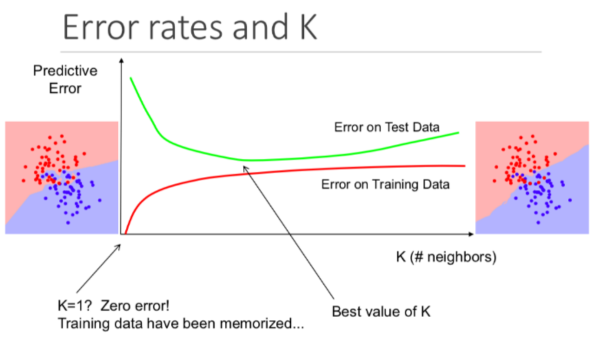


Figure 6 (Singh, 2020)

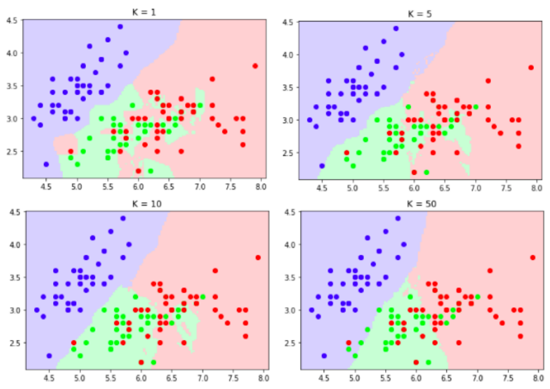


Figure 7 (Lin, 2020)

From this diagram we can see that when K is small there are some outliers of green label are still green and outliers of red label are still red. When K is larger the boundary is more constant.

REINFORCEMENT LEARNING

Reinforcement learning is the training of machine learning models to make a series of decisions. The rational agent learns to achieve a goal in an uncertain, possibly complex environment. The artificial intelligence faces a game-like scenario and then uses the approach of trial and error to solve the problem. The machine is rewarded or penalised by the programmer for the actions it performs, the goal being to maximise the total reward. This continues until the system reaches a final state. In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure. An example of reinforcement learning in practice is the training of models that control autonomous cars. The programmer cannot predict everything that could happen on the road. Instead of incorporating lengthy "if-then" instructions, the programmer prepares the reinforcement learning agent to be capable of learning from the system of rewards and penalties.

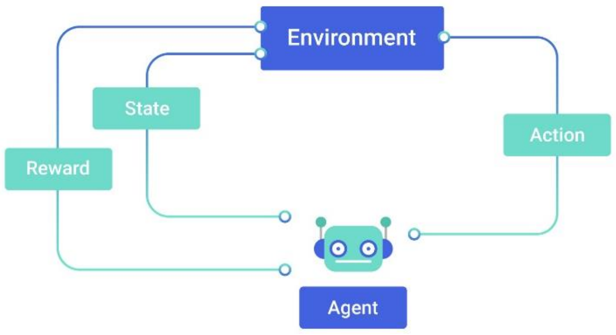


Figure 8 (Demush, 2020)

5. ETHICAL CONCERNS

Many ethical concerns arise from the area of data collection. Collecting and analysing data is a new and complex issue, one that is growing faster than protective frameworks can be put in place. As trends like machine learning, big data and data analytics become more and more prevalent in our society, demand intensifies for laws and regulations to be put in place to protect our privacy. But these regulations are struggling to keep up with the rate that data capabilities are moving.

In May of 2018, the EU released a ground-breaking General Data Protection Regulation (GDPR), the toughest privacy and security law in the world. GDPR imposes regulations onto organizations worldwide not just in Europe, because if the data being collected is related to people in the EU, then it is covered by this law. Harsh fines are levied against companies who disregard privacy and security standards, some of which can reach tens of millions of euros. GDPR has strengthened the protections around data collection and requires companies to be more transparent about all the data they hold, how the data is used, and how it is stored. The right to privacy which is part of the 1950 European Convention on Human Rights, states,

"Everyone has the right to respect for his private and family life, his home and his correspondence."

This was the basis on which GDPR was established as the EU sought to protect this right through legislation. However, the legislation it replaced was written in 1995 which shows how slow we are to adapt to rapidly evolving technology.

Although GDPR has made strides in the right direction, we are still left with many questions surround data protection. What kind of data should be allowed for use and what is off limits? When and where is it acceptable to measure the activity of a developer? And how can the engineer be made aware of this monitoring and be allowed to give consent or dissent? I will address some of the main ethical concerns below.

DATA COLLECTION

One ethical concern regarding data collection is that there is no boundary to how much data a company can gather about their staff. When a company is measuring the productivity of their developers it is vital that they find the balance between collecting enough useful information while also ensuring that the employees' privacy is prioritised. As discussed earlier, software engineering metrics can be used to improve productivity, but some employers have pushed beyond using only these tools, recording personal information like health records, search history and financial information. Understandably, employees would not be comfortable with this much information being accessible to employers. The employees lack of awareness of the extent of the data collected adds to the unease they feel, a prime example of this is the development of Hackystat and employee's unhappiness with the fine-grained data collection involved with the platform.

It is evident that there needs to be transparency between the employees and employer as to what data is being gathered. This is where GDPR comes in to ensure the transparency of data collection. Employers should also limit themselves to collecting data that is directly linked to the work the employee is doing in order to improve the software engineering process, make better business decisions or improve employee productivity.

DATA USAGE

The main concern for developers is how the data gathered about them is analysed and what information can the employer derive from this analysis about them. The large amount of data collected can be used for predictive people analytics, which is when certain values or events can be identified and then changed in order to forecast a better outcome. In this context employers may use this as a recruitment tool to identify high performing employees and employees likely to leave the company. There are many ethical questions that must be asked of organisations that utilize these tools. Are employers entitled to know this information? Is it fair to use these tools to predict an employees' performance and then value them based on this forecast? We must also recognise that these analyses are not fact and can mislead employers.

DATA SOVEREIGNTY

Data sovereignty is the concept that data is subjected to the laws and governance within the country that its located. But when information is converted and stored in binary digital form in a foreign country, concerns arise like enforcing privacy regulations and preventing the data from being subpoenaed by the host country. Widespread use of cloud computing has enhanced these issues. Employees not only have to trust their employers with their data but also trust that the cloud provider is honest about where their server is located and comply with the terms of agreement.

CONCLUSION

There are a vast number of approaches to measuring software engineering today. The metrics, platforms and algorithms covered in this report only scratch the surface of what is available to both companies and individual developers. It is understandable than companies want to measure, quantify and improve the productivity of its employees as the value of a software engineer is high and in modern competitive markets, companies want to get value for their money. So long as the boundaries of privacy are not crossed when collecting a developers' data and the public and employees continue to scrutinize the unethical organisations who abuse the data collection process, I think we don't have too much to worry about. The measurement of software engineering goes hand-in-hand with the job of a software engineer and rather than try to resist it, I believe that time would be better trying to make as many ethical improvements as possible.

SOURCES

CITATIONS

Linkedin.com. 2020. *What Are The Software Development Life Cycle (SDLC) Phases?*. [online] Available at: <https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited/> [Accessed 24 November 2020].

Tranter, L., 2020. *Cycle Time And Lead Time - Extreme Uncertainty - Practical Agile*. [online] Extreme Uncertainty. Available at: <https://www.extremeuncertainty.com/cycle-time-ead-time/> [Accessed 19 November 2020].

Franken, L., 2020. *Github Contribution Calendar Considered Whatever You Make Of It - Logan Franken*. [online] Loganfranken.com. Available at: <https://www.loganfranken.com/blog/1167/github-contribution-calendar-considered-whatever-you-make-of-it/> [Accessed 27 November 2020].

Docs.microsoft.com. 2020. *Analyze And Report On Code Churn And Code Coverage - TFS*. [online] Available at: <https://docs.microsoft.com/en-us/azure/devops/report/sql-reports/perspective-code-analyze-report-code-churn-coverage?view=azure-devops-2020> [Accessed 19 November 2020].

Johnson, P., Kou, H. and Agustin, J., 2003. *Beyond The Personal Software Process: Metrics Collection And Analysis For The Differently Disciplined - IEEE Conference Publication*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/1201249/> [Accessed 22 November 2020].

Singh, S., 2020. *CS 273A: Machine Learning (Fall 2017)*. [online] Sameersingh.org. Available at: <http://sameersingh.org/courses/gml/fa17/sched.html> [Accessed 25 November 2020].

Lin, T., 2020. *Day 3—K-Nearest Neighbors And Bias–Variance Tradeoff*. [online] Medium. Available at: <https://medium.com/30-days-of-machine-learning/day-3-k-nearest-neighbors-and-bias-variance-tradeoff-75f84d515bdb> [Accessed 25 November 2020].

Demush, R., 2020. *ᐅApplications Of Reinforcement Learning - Perfectial*. [online] Perfectial. Available at: <https://perfectial.com/blog/reinforcement-learning-applications/> [Accessed 26 November 2020].

BIBLIOGRAPHY

J. Liu, Y. Zhou, Y. Yang, H. Lu and B. Xu, "Code Churn: A Neglected Metric in Effort-Aware Just-in-Time Defect Prediction," 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, ON, 2017, pp. 11-19, doi: 10.1109/ESEM.2017.8.

(Anastasios), T., 2020. *High Code Churn Is Not Your Friend | Drinkbird*. [online] DrinkBird. Available at: <https://blog.drinkbird.com/code-churn> [Accessed 23 November 2020].

McKeown, T., 2018. *Council Post: The Top Five Predictive Models For People Analytics*. [online] Forbes. Available at: <https://www.forbes.com/sites/forbestechcouncil/2018/08/27/the-top-five-predictive-models-for-people-analytics/?sh=70de88944573> [Accessed 27 November 2020].

GDPR.eu. 2020. *What Is GDPR, The EU'S New Data Protection Law? - GDPR.Eu*. [online] Available at: <https://gdpr.eu/what-is-gdpr/> [Accessed 27 November 2020].

Cloud Management Insider. 2020. *What Does Data Sovereignty Mean In Cloud World?*. [online] Available at: <https://www.cloudmanagementinsider.com/what-does-data-sovereignty-mean-in-cloud-world/> [Accessed 27 November 2020].

Medium. 2020. *K-Means Clustering: Algorithm, Applications, Evaluation Methods, And Drawbacks*. [online] Available at: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a> [Accessed 27 November 2020].

Osiński, B. and Budek, K., 2020. *What Is Reinforcement Learning? The Complete Guide - Deepsense.Ai*. [online] deepsense.ai. Available at: <https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/#:~:text=Reinforcement%20learning%20is%20the%20training,faces%20a%20game%2Dlike%20situation.&text=Its%20goal%20is%20to%20maximize%20the%20total%20reward.> [Accessed 27 November 2020].

Learning, U., 2020. *What Is Unsupervised Machine Learning?* | *Datarobot*. [online] DataRobot. Available at: <https://www.datarobot.com/wiki/unsupervised-machine-learning/> [Accessed 27 November 2020].

Paiva, D., Freire, A. and de Mattos Fortes, R., 2020. Accessibility and Software Engineering Processes: A Systematic Literature Review. *Journal of Systems and Software*, 171.

Fenton, N. and Neil, M., 1999. Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3), pp.149-157.

Johnson, P., Kou, H. and Agustin, J., 2003. *Beyond The Personal Software Process: Metrics Collection And Analysis For The Differently Disciplined - IEEE Conference Publication*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/1201249/> [Accessed 22 November 2020].