MEASURING SOFTWARE ENGINEERING

By Conor Power - 16322195

Abstract

In this report I will explore the idea of measuring the software engineering process. I will firstly give an overview of the software engineering process and discuss various models of the process used in practice. I will then explore the motivations for and metrics used in measuring the work of a software engineer. I will also examine numerous computational platforms and algorithmic approaches for carrying out these measurements. Finally, I will conclude the report by discussing the various ethical issues that arise when attempting to collect and analyse data such as the data used in measuring a software engineer's work.

The Software Engineering Process

The software engineering process refers to the set of related activities involved in developing and maintaining software. Although there are various software engineering process models, all models incorporate some form of the following activities:

1. **Software Specification**
   At this stage, engineers and/or customers specify the main functionalities and constraints of the software being designed.
2. **Software Design and Implementation**
   The actual designing and programming of the software occurs at the design and implementation stage.
3. **Software Verification and Validation**
   During this stage, the software is tested, and the engineering team makes sure that the software conforms to its specifications and meets the needs of the customer.
4. **Software Evolution**
   This refers to the maintenance of software. Software maintenance ensures that the software evolves over time to meet the changing requirements of the user.

In practice, the software engineering process is far more complex than just the culmination of these four activities. Each activity mentioned above has a number of associated sub-activities, such as requirements validation, architectural design, and unit testing. There are also many supporting activities involved in the software engineering process, which include change management, quality assurance, and user experience [1].

Software Engineering Process Models

There are numerous software engineering process models that exist. In the real world, the development of large systems often incorporates elements of numerous models. One such model is the Waterfall Model, which is a sequential and rigid plan-driven approach in which activity planning is completed before work begins. However, use of this process is limited, as it can only be applied for projects in which the requirements are well-understood beforehand and are unlikely to change [1].

Another major approach to software engineering is the agile development model (figure 1). This is a development process characterised by regular and continuous iterations of testing and development. With agile development, the requirements of the software continuously evolve over time and are updated through the collaboration of many cross-functional engineering teams and end-users [2].

Agile development processes allow organisations "design and build features quickly, test them with customers, and refine and refresh them in rapid iterations" [3]. According to a McKinsey report on agile practices, companies who have deployed agile methods have seen their innovation accelerate by up to 80%. However, many traditional organisations often find it difficult to scale agile methods into their development approach. This is mainly due to their existing organisational structure, which is often characterised by many fragmented development teams. These teams have traditionally been responsible for a single segment of the organisation's overall software division, and thus find it difficult to adapt to a more collaborative agile approach [3].

There are numerous frameworks based on the principles of agile development used in practice. These frameworks include Extreme Programming and Scrum Methods. Extreme Programming is an agile development approach encapsulated by five core values; communication, simplicity, feedback, courage, and respect [2]. The Scrum Method is another agile approach which involves 'sprint cycles' which are short boxed-off periods of time in which specific work is to be completed and made ready for review [4].

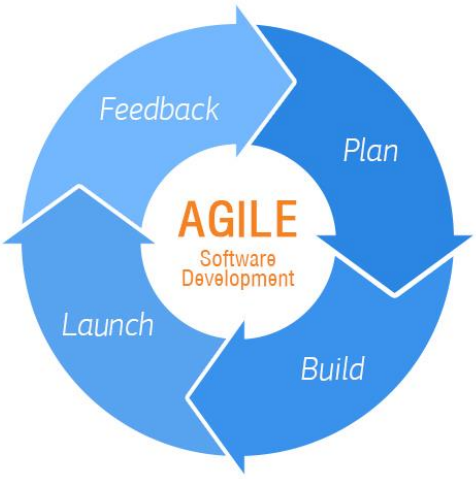

Figure 1 – Agile Development

The Personal Software Process (PSP) is another approach. The PSP is used in conjunction with other process models and allows individual software engineers to plan, measure, and manage their work [5]. I will discuss the PSP later in this report, as it provides a framework for measuring the work of a software engineer.

Measurable Data

Why Measure the Software Engineering Process?

There are many reasons for measuring the work of a software engineer, although opinions differ on whether this is a valuable and/or ethical task. Firstly, it must be noted that the work of a software engineer is complex and involves many tasks and sub-tasks, such as those discussed in the first section of this report. There are many reasons why people believe that measuring a software engineer's work is useful. These reasons include helping to "identify and propagate best practices by comparing productivity across teams, projects, companies and sectors", whilst also helping to "avoid waste and reduce costs to consumers…" and allowing people and companies to "verify whether new tools or methodologies actually increase productivity" [6]. Other motivations include profit boosting and efficiency maximisation.

Those who believe that the work of a software engineer should not be measured often argue that the complex nature of software engineering renders a software engineer's work unquantifiable. Many who hold these beliefs often use examples such as LOC (lines of code), number of bug fixes, and story points to point out the fact that such measurements often reward inefficiency and therefore an engineer's work should not be measured directly. Instead, they argue that things such as job satisfaction [7] and business outcomes [8] should be used to measure an engineer's work. However, these forms of measurement fail to distinguish between the work of a software engineer and the work of any other type of employee within a company, and hence are effectively useless for gaining any insights into the unique profession of software engineering. In reality, there are ways of quantifying the complex nature of a software engineer's work.

What metrics are useful?

It remains clear that the work of a software engineer is not akin to that of a blue-collar assembly-line worker, and as such cannot be measured in such simple terms as gross output or total hours worked. Instead, the complex work of a software engineer requires more nuanced metrics, where relevant comparisons and trends are more important than the raw data. In their paper titled "Collecting, Integrating and Analysing Software Metrics and Personal Software Process Data", Sillitti et al. state that there are two main types of software engineering metrics; Product Metrics and Process Metrics [9].

Process Measurement

One useful type of metric regarding the work of a software engineer is a process metric. For the agile development process, there are a number of key metrics that can be used to analyse the software engineering process and aid with planning and decision-making. These metrics include:

1. **Leadtime**
   This metric measures the time taken to go from idea to software delivery. In general, short lead times are good for both the development team(s) and the customer. However, project quality should not be sacrificed to improve this metric. Ways of improving lead time include simplifying the decision-making process and reducing wait time.

2. **Cycle Time**
   This metric measures the length of time taken to make a change to the software system and actually implement that change. This metric will vary depending on the delivery method of the development team, and hence cross-team and cross-project comparisons can be misleading.
3. **Team Velocity**
   This refers to how many "units" of software a team completes in an iteration (e.g. a sprint in Scrum Methodology). This metric is useful for planning iterations but should not be used as a comparison against other software engineering teams or previous projects.
4. **Open/Close Rates**
   This metric refers to the number of production issues that are reported and closed during a specific time period. With this metric, the trend is more important than the raw data. If open rates are consistently higher than close rates, it could suggest that a team is not giving enough priority to product issues. However, further examination would be required before coming to such a conclusion.

Product Measurement

Another useful type of metric for measuring the work of a software engineer is a product metric. Product metrics refer to measurements relating to the actual software product. These metrics include:

1. **Mean Time Between Failures (MTBF) and Mean Time to Recover / Repair (MTTR)**
   Both of these metrics measure the performance of the software in the production environment. Given the inevitability of software failures for most software, these metrics measure how well the software recovers after failure, and whether data is successfully preserved.
2. **Application Crash Rate**
   This metric is calculated by dividing the number of times the software fails by the number of times it was used. It is similar to the MTBF and MTTR, and like those metrics, gives a good indication of system performance.

Some other metrics used include the following:

- Code churn
- Endpoint incidents
- Active days
- Defect removal efficiency
- Errors per KLOC

[10] [11]

There are also a number of non-code metrics which can be useful in measuring the work of a software engineer. These metrics involve the non-technical aspect of a software engineer's work and are therefore common for most types of workers within any organisation. These metrics can include things such as the number of meetings attended or the number of answered emails. However, the

majority of my analysis throughout the rest of this report will be centred around the more technical metrics, such as product and process metrics.

The Personal Software Process

The PSP (figure 2) is an important framework for enabling software metrics to be measured. The PSP provides a personal process framework for individual software engineers, allowing them to track and measure their own performance. By adhering to the PSP process, engineers develop a plan for every project, record their development time, track project defects, retain project data in summary reports and then use this data for future project planning [12]. The PSP breaks measurements down into four types; time measures, size measures, quality measures, and schedule measures. Once recorded, these measurements are analysed, compared, and used to plan and improve future projects [13].

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

Figure 2 – The Personal Software Process [14]

Computational Platforms

Over the last decade, there has been an explosion in the number of big data analytics companies, who gather and analyse data on everything from the workplace to sports and politics. The software engineering profession is no different and has seen numerous companies set up to measure the software engineering process. These companies use metrics such as those discussed in the previous section "to obtain insightful and actionable information from software artifacts that help practitioners accomplish tasks related to software development, systems, and users" [15]. Some of these companies and their approaches to software engineering measurements are discussed below:

1. **GitPrime**
   GitPrime (figure 3) uses data from Git-based code repositories, such as Github, to help software engineering teams optimize their work patterns. GitPrime aims to make the software process more transparent, allowing managers and other key stakeholders to track the progress and performance of the software team. The

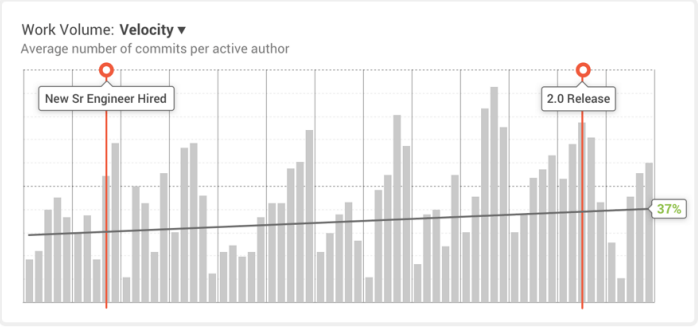

Figure 3 – GitPrime Data Visualisation

GitPrime service uses metrics such as code churn and commit velocity to enable feedback, progress-tracking, and risk identification [16].

2. Squore

Squore (figure 4) is a business intelligence tool used for software and systems project monitoring. Squore uses automated data processing to assess software based on a number of KPIs. These KPIs include code cloning, coverage compliance, complexity, violations density, and completion rate. The Squore software then produces dashboard reports and visualisations based on these KPIs, which allow Software Engineering managers and engineers themselves to analyse their work and make various comparisons and forecasts to optimise their performance and projects [17]. Renault have used Squore tools to assist in their software-related vehicle development process. Using Squore software, Renault reduced their previously 24-hour calculation and metric analysis process time by 96%, to just 15 minutes. As well as saving time, Squore's automated tools offer advantages such as quick refresh times, error-free calculations, and low maintenance [18].

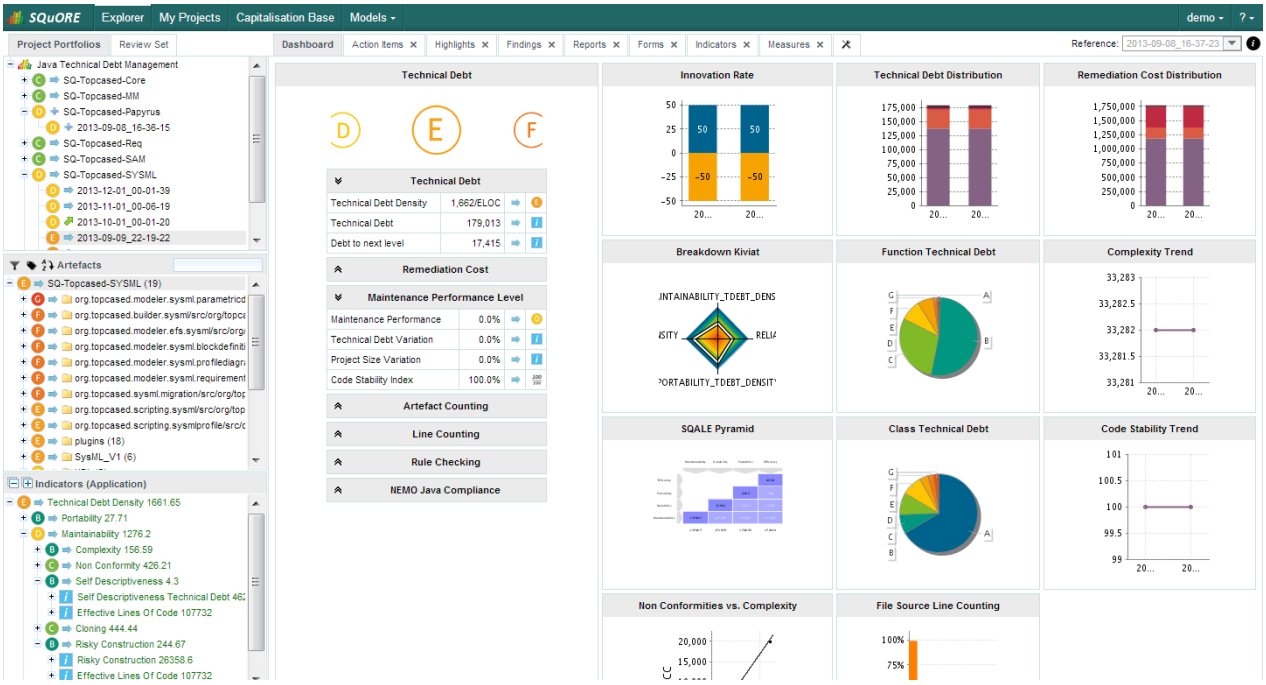

Figure 4 – Squore Dashboard

3. Codacy

Codacy is a software company that provides tools for static program analysis. Static program analysis refers to the automated analysis of source code, which identifies and flags errors, as well as collecting and presenting quantitative and qualitative metrics relating to the source code. Codacy supports the analysis of 28 different languages and offers numerous key functions to software engineering teams. Codacy identifies security vulnerabilities in code, standardizes code across teams and projects, identifies occurrences of standards being violated in a project, and also displays key metrics regarding projects [19]. One key metric that Codacy tracks is technical

debt. Technical debt measures the cost of "the extra development work that arises when code that is easy to implement in the short-run is used instead of applying the best overall solution" [20]. Technical debt is a major problem for many development teams, but with the help of tools such as Codacy, it can be reduced or eliminated. As technical debt is often associated with the need to 'complete' a project, agile methods such as Extreme Programming can often mitigate the causes of technical debt. The continuous iterations carried out in agile development allow development teams to stay on top of technical debt [21]. Codacy is used by a number of large companies including PayPal, Adobe, and Deliveroo [22].

Other similar companies that provide tools for static program analysis include Code Climate, Sonatype, Assembla, and Sider.

4. **PSP Measurement Tools**

   There are also a variety of platforms which specifically aid the Personal Software Process and allow the important metrics from the PSP to be tracked and analysed. One such platform is an open source project called "The Software Process Dashboard", which allows metrics deemed useful in the PSP to be visualised and analysed on a series of dashboards. Another framework for automatically collecting and analysing PSP data is Hackystat. Hackystat is a data collection tool developed at the University of Hawaii which allows developers to collect data through sensors attached to IDEs such as Eclipse. This data can then be retrieved and analysed in XML form. An alternative to Hackystat is PROM, which is a tool used for automated data acquisition and analysis. PROM collects and analyses both product and process metrics [23].

Algorithmic Approaches

There are numerous algorithms available which make analytical tools such as those discussed above possible. The majority of these algorithms fall under the category of predictive analytics and therefore I will be basing my exploration of algorithmic approaches used in measuring the software process on predictive analytics. One major area of predictive analytics is machine learning. Machine learning refers to the ability of a computer to 'learn' from data without being explicitly programmed. Machine learning uses historical data to form models which can make predictions and inferences about the nature of data. There are two main types of machine learning algorithms:

1. **Supervised Machine Learning**
   Supervised learning is a machine learning technique that maps an input to an output using data for which the input and output are already defined. This dataset, for which inputs and outputs are already known, is known as the training set. The supervised learning process involves the algorithm making predictions on the training dataset, which are then corrected. This trial and error process repeats until the algorithm's performance is deemed acceptable. There are two types of supervised learning algorithms:

- **Classification**

  Classification algorithms are used to classify observations into specific categorical groups. An example of a classification algorithm is k-nearest neighbours (k-NN). The k-nearest neighbours algorithm is a non-parametric method, meaning it makes no assumption about the underlying distribution of the data. The k-NN algorithm classifies an observation as being a part of the group for which the majority of the observation's k nearest observations are a part of. Other examples of classification algorithms include logistic regression, random forest, and support vector machines.

- **Regression**

  Regression algorithms are used to make predictions when the output is continuous. For example, in software engineering analytics, one might be interested in predicting lead times for a particular project based on numerous inputs such as the number and complexity of initial specifications. Here the training set might be similar past projects, where the initial specifications and the eventual actual lead times are known. An example of a regression algorithm is simple linear regression (figure 5). Here the model is of the form Y = B0 + B1X, where Y is the output variable, B0 is the Y intercept, B1 is the slope of the regression line, and X is the input variable. The parameters B0 and B1 are estimated from the training data. Another example of a regression algorithm is multivariate regression.

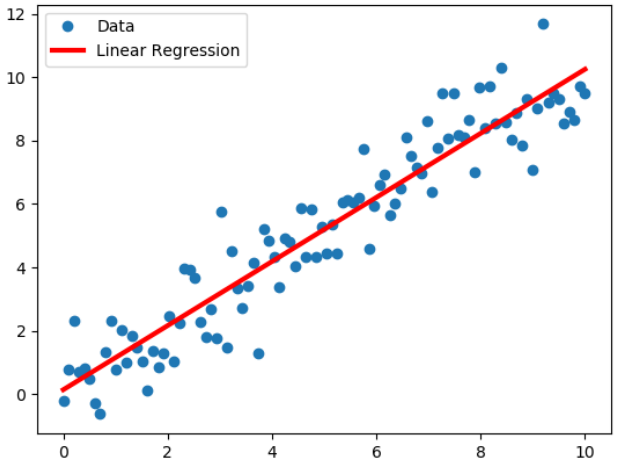

Figure 5 – Simple Linear Regression

2. Unsupervised Machine Learning

Unsupervised learning algorithms are algorithms for which there is only input data and no output data. The goal of unsupervised learning is therefore to model the structure of the data in order to make inferences about that data. Here there is no correct answers for the algorithms to learn from. Instead the algorithms attempt to find structure in the data by themselves. There are two types of unsupervised algorithms:

- **Clustering Algorithms**

  Clustering algorithms are used to find groupings within data. Groupings are decided based on similarities between the input data of the observations. The goal of clustering algorithms is to make observations within groups similar, and observations between groups dissimilar. An example of a clustering

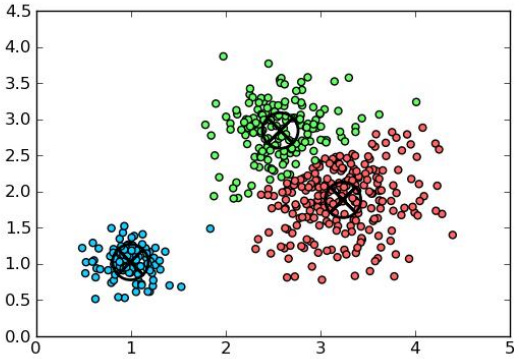

Figure 6 - K-Means Clustering

algorithm is k-means clustering, which partitions n observations into k clusters of observations. Each cluster has a mean value known as a centroid. Observations are classified into the group of the closest centroid. The number of clusters used in this algorithm must be decided before the algorithm is run and is often difficult to choose. Some conditions for choosing k include background knowledge of the data and finding a suitable trade-off point between minimising the sum of squared errors and the complexity of the model. An example of clustering in the measurement of software engineering could be to see if there are groupings of software engineers based on various metrics such as code churns and other metrics discussed earlier in this report. Other examples of clustering algorithms include hierarchical clustering and mean-shift clustering.

- **Association Rule Learning Algorithms**
  Association rule learning algorithms are used to discover interesting relationships between observations in a dataset. The goal of this type of machine learning is to "help a machine mimic the human brain's feature extraction and abstract association capabilities from new uncategorized data" [24]. An example of an association rule learning algorithm is the Apriori Algorithm. In a software engineering measurement context, association rule learning algorithms could be used to see if certain behaviours of an engineer are more likely to result in a certain project outcome.
  [25] [26]

The Limits of Algorithmic Approaches

While it is clear that predictive analytic techniques can be extremely useful in the field of software engineering analytics, there are some limitations. In many cases, predictive analytic models can be highly complex and often the humans behind these models do not understand why a machine is coming to a certain conclusion. Many people argue that a model should not be used if humans cannot understand how the results are being derived. Another key concern around machine learning is cognitive bias. As many datasets fed into machine learning algorithms have been influenced by humans, any human bias will inevitably affect the outcome of the predictive model. One example of human bias in machine learning has been seen in data-driven job hiring algorithms. Numerous job hiring algorithms have been found to reinforce human bias in the hiring process by excluding candidates on the basis of gender, age, race, and disability. These results were proven by sampling identical resumes, where factors such as gender and name were changed. The resume-scanning algorithms were found to dismiss candidates where the gender was female or where there was a foreign-sounding name. These biases occurred because of real human-bias in the data being used to train the algorithms. It is clear that great caution must be taken when feeding data into data-driven algorithms [27].

The Ethics of Analytics

Having explored many of the metrics and methods used for measuring the work of a software engineer, it has become clear to me that there are a number of key ethical issues that must be

confronted. These issues are often highly subjective and therefore, I will attempt to examine each concern from a number of different angles, as well as offering my own personal opinion.

Legal Systems and Data Sovereignty

The first issue that must be discussed when exploring the ethics of data collection is the legal system and data sovereignty. Data sovereignty refers to the idea of data being subject to the laws and regulations of the country from which it is collected. The issue of data sovereignty has become more prevalent and relevant in recent times due to the increase in cloud computing capabilities. Many businesses around the world have cited data sovereignty laws as a barrier to cloud adoption and feel that failure to move to the cloud could cause long-term damage to their business [28]. In Europe, the recent introduction of the General Data Protection Regulations has placed restrictions on how companies can use the data of EU citizens. The GDPR regulations have seen companies, who fail to protect their customer's data, being subject to large fines. In contrast, the US Patriot Act, signed into law in 2001 following the 9/11 attacks, states that the US government can access any data within the US, regardless of the data's origin. The contradicting laws regarding data in the US and Europe lead us to question whether being legal makes something ethical and vice-versa.

Abuse of Data

From researching various controversies surrounding data collection, it seems clear to me that the line between ethical and unethical data collection practices is often blurred. One high profile example of seemingly unethical data practices is the NSA's PRISM program in the US. This program, the details of which were leaked by the now famed whistle blower Edward Snowden, allowed the US government to monitor the emails, video clips, photos, and phone calls of people all around the world. Many people believe strongly that this was an abuse of power by the NSA, and that the program breached the basic human right to privacy. However, some people argue that this program was in the interests of national security. After all, do the government really care about the conversations you have with your wife, so long as you're not planning the next major terrorist attack?

Another more recent issue has arisen with Amazon's Alexa device. In the US, a judge has ordered Amazon to release recordings made by one of their Alexa devices in the home where a man has been charged with murdering two women. The fact that the Alexa device is recording, even when not being used, has been a major cause for concern amongst owners of the device. Amazon has so far refused to release these recordings on legal grounds. However, given the serious nature of the incident in question, there seems to be ethical grounds for releasing these recordings. But we must ask ourselves whether this data should exist in the first place [29].

Conclusion

From studying various cases of controversial data practices, I believe that, in general, there should be a limit on what data is recorded and on who can see this data. In my opinion, governments should reserve the right to access information about the citizens of their country, but not the information of citizens from other countries. I also believe, however, that the extent of this data

should be limited, but I am unsure as to where to draw the line. In the case of the NSA's PRISM program and Amazon's Alexa device, I feel that the collection of live voice and video recordings should not be permitted, as it is a step too far. Regarding the recording of metrics related to the work of software engineers, I believe that the nature of this data does not pose a threat to one's privacy. However, such data should perhaps be kept within the confidential confines of the employers of these engineers.

Bibliography

[1] https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc

[2] https://www.agilealliance.org

[3] https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/an-operating-model-for-company-wide-agile-development

[4] https://resources.collab.net/agile-101/what-is-scrum

[5] https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

[6] https://semmle.com/assets/papers/measuring-software-development.pdf

[7] https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451

[8] http://theworkspacetoday.com/2017/01/13/for-engineering-performance-stop-measuring-productivity/

[9] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

[10] https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams

[11] https://stackify.com/track-software-metrics/

[12] ftp://ftp.sei.cmu.edu/pub/documents/articles/pdf/psp.over.prac.res.pdf

[13] https://resources.sei.cmu.edu/asset_files/SpecialReport/2009_003_001_15029.pdf

[14] https://www.researchgate.net/figure/The-Personal-Software-Process_fig4_220542225

[15] D. Zhang, S. han, Y. Dan, J.-G. Lou, H Zhang: "Software Analytics in Practice". IEEE Software, Sept./Oct. 2013, pp. 30-35.

[16] https://www.gitprime.com/product/

[17] https://www.squoring.com/en/produits/squore-software-analytics/

[18] https://hal.archives-ouvertes.fr/hal-01708535v1/document

[19] https://www.codacy.com/

[20] https://www.techopedia.com/definition/27913/technical-debt

[21] https://www.bmc.com/blogs/technical-debt-explained-the-complete-guide-to-understanding-and-dealing-with-technical-debt/

[22] https://techcrunch.com/2017/08/17/codacy/

[23]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf

[24] https://searchbusinessanalytics.techtarget.com/definition/association-rules-in-data-mining

[25] Brett Houdling, Trinity College Dublin – ST3011 Lecture Notes

[26] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[27] https://hbr.org/2016/12/hiring-algorithms-are-not-neutral

[28] https://www.forbes.com/sites/forbestechcouncil/2017/04/11/is-data-sovereignty-a-barrier-to-cloud-adoption/#7d2a90411c82

[29] https://www.theweek.co.uk/97835/amazon-alexa-to-give-evidence-in-double-murder-case