Software Engineering Report



# Introduction

In this report I am going to consider the ways in which the software engineering process can be assessed. This will primarily focus on doing so in terms of measurable data. In order to provide an overview on this area, I will focus on four main topics; the software engineering process and what it entails, how to assess the process using measurable data, available computational software that can be used, available algorithmic approaches used, as well as the ethical concerns surrounding the analysis of the software engineering process. I will be basing this report on all material from module CSU33012, including lecture content and reading materials.

# WHAT IS THE SOFTWARE ENGINEERING PROCESS?

‘Software engineering’ is the term given to using engineering in the process of developing software using systematic methods. The term was first coined in the 1960s at a NATO conference where it was decided that a systematic process along the lines of the processes employed in physical engineering should be utilised in software development.

Despite this being a relatively young discipline, software engineering has evolved extremely quickly. As a discipline, it encompasses all aspects of software production. Software engineers to utilise engineering systems to design, produce, release, manage and maintain any software they create. It differs from traditional computer science in that it focuses on developing high quality usable software with industry applications, rather than focusing on the theories and algorithms behind software development which underpin most computer science fields.

Similar to most companies and industries nowadays, software engineers and software engineering firms want to be able to measure their performance in order to find weaknesses in their operations and ways that they can improve their service for client benefits, as well as possibly improving their own internal performance, e.g. by realising cost saving associated with more efficient processes. in this report I am going to discuss how they can do this using quantifiable, measurable data.

# ASSEssing the software engineering process with measurable data

*“If we have data, let’s look at data. If we all have opinions, let’s go with mine.”*

-Jim Barksdale

One of the main challenges associated with measuring the performance of a software engineer is how do we measure a person? There is no simple metric that can achieve such a conclusion. After all, it is a huge combination of factors that combine to make someone a good software engineer, and not all of them can be measured. Some aspects are so subjective that what could be construed as a positive for one assessor could be a negative in another’s eyes.

With that said it is possible to objectively measure aspects of a software engineers’ performance and derive a numeric value or result for them. By comparing these results across a company or industry it is possible to draw conclusions about the quality of work done and assess the effectiveness of the tools, processes and methods employed. The basis upon which this analysis is built is understanding the Software Development Life Cycle which is used to divide projects into distinct phases. These following phases are generally used across the software engineering industry.

* Planning: The project requirements are gathered, and the scope that the project will operate under is determined. As part of this, a plan of action is also created.
* Development: this is the phase of the process where the actual code is written.
* Testing: Extensive testing is done to identify and correct any bugs or defects in the software.
* Deployment: Following the testing stage, once the software has been deemed to be bug free and fully functional, it is deployed for general use.
* Maintenance: It can often take longer to improve and upgrade software if given new requirements that must be fulfilled. In order to adapt to these changes in demands, software engineers must constantly update, upgrade and maintain the initial software.

Why is it important to gather data on software engineers? Because they are the most essential part of a software engineering company, and without them the company cannot function as it intends. Despite, as previously mentioned, the fact that there is no one agreed metric for judging performance, companies still engage in analysis and performance.

The primary goal of such activities is to increase performance, both in the individual and in the company as a whole. This is done by identifying any noticeable flaw and exchanging them for the adopting of the most efficient processes. A secondary goal of data gathering should be to establish questions to be answered by the data gathered. This is essential to avoid the over gathering of data which can mean a cumbersome overload of information.

The main measurable data which can be gathered from a software engineers work is as follows:

* Readability of their code: How easy the code is to understand from the outside looking in.
* Source Lines of Code: The number of actual lines of code written. It can potentially be a very misleading measure because it can vary highly depending on the engineer’s approach and/or skill level.
* Number of Commits: The number of times the software engineer contributes to or updates their software. This number does not necessarily give the data to determine whether the software engineer is good or bad, but it gives a useful insight into their working practices and the methodology they employ when programming.
* Lead Time: This can be an incredibly useful piece of data if analysed correctly. It's defined as "the time elapsed between the identification of a requirement and its fulfilment. It tracks the total time taken for a software engineering project, from the time the project was commissioned to the time that it was delivered to the client. This data can help the management data predict how much work a given software engineer can complete in a given period f time, as well as comparing engineer’s performance in similar projects.
* Cycle Time: Quite similar to Lead Time, this data tracks the time taken for the project to be completed from the time that the software engineer begins working on the project, not when the request was initially made. This is useful when analysing the performance of a specific engineer rather than that of the company as a whole.

* Technical Debt: Specific to software development, this is debt that arises when extra development work is required because code that was easy to implement in the short term was used instead of the best overall solution. Technical debt should be paid back promptly with a rewrite.
* Code Churning: This is usually measured as lines of code that were modified, added and deleted over a short space of time. It is also counted as the percentage of a developer's own code representing an edit to their recent work. If the churn rate suddenly spikes, it can indicate that the project has encountered a problem. This can vary from a software engineer putting final touches on their code prior to release to a serious unexpected issue arising that the engineer doesn’t know how to solve.

* Test Coverage: This checks how much of the source code is covered by the developers tests and shows how much of the source code is executed when the tests are run. Management will often decide what test coverage they deem adequate and appropriate and see if these standards are met by the software engineers.
* Bug Fixing: This data tracks how long, per project, a software engineer spends fixing bugs in the source code. Management should be more concerned with the severity of the bugs that arise, and how long the engineer takes to solve them, rather than the amount of bug arising in an engineer’s source code.
* Task Complexity: It can be very useful to analyse the complexity of given projects and whether the project was performed by an individual or by a team.

The major issue with this data is that it only tracks the product metrics, i.e. the final product produced. It does not track fewer tangible factors, such as an engineer’s contribution to meeting, how responsive they are to client and supervisor emails, and their ability to work in a team of engineers. These factors can be more difficult to measure and are somewhat subjective, but they are also very important and cannot be ignored just because the difficulty in quantifiably measuring them.as well as this, the issue of measurement dysfunction cannot be ignored.

The pressure that may be put on employees to produce good quantifiable results, may lead their performance in non-measurable areas to be reduced so much that the company performs worse as a result.

# available computational platforms

This part of the report covers the platforms and programs that are available to help you analyse and interpret data once you have gone through the process of collecting it.

The **Personal Software Process (PSP)** was developed by the Software Engineering Institute, with the goal of identifying potential improvements for individual software engineers. In 1995, Watts Humphrey wrote a report on the PSP. It was his opinion that structuring the software development process with the inclusion of constant progress tracking would make the process more efficient.

PCP is based on collecting data from the developer, such as a project plan summary, a log of defects, design checklist and a code checklist. PSP relies on developers using their judgment to log the data accurately and independently. This is the primary flaw with the PSP as it can be prone to human error.

The **Leap Toolkit** aimed to build on the PSP and solve some of the data quality issues that the PSP had encountered. It still required developers to manually enter the details, but automated the following analyses, and provided some alternative analysis methods that PSP didn’t have such as regression analysis. The Leap approach is considered “Lightweight” because it doesn’t determine the sequence of development activities. It also tries to avoid dysfunction in measurement by allowing the software engineers to control their data files.

The increase in automation made some analytics easier to collect and led to improvements in certain areas, it had a converse effect elsewhere in that it made others increasingly difficult due to the lack of customisation. Overall it was concluded that it would never be possible to ever fully automate the PSP, as some form of manual input would always be required. It became apparent that in order for unbiased analysis, automated input would be required. This led to the development of **Hackystat.**

Hackystat works off of a service-oriented architecture which implements sensors on development tools that gather data relating to the process and product and send it to be stored on a server. From there, other services can query the data and build high level analyses. This requires minimal overhead for developers, a key benefit of this platform. Hackystat made use of four core design elements.

1. Dual client and server-side data collection.
2. Unobtrusive data collection.
3. Fine-grained data collection.
4. Dual personal and group-based development.

The main issue faced by Hackystat was to do with its reception from developers. Many of them did not like the way that it collected data without making them aware. Especially due to its final two core features, which allowed data to be collected minute by minute, and also tracked how developers worked together when working on a group project, many developers were vocally against it and had a deep mistrust of the software. Overall it solved the issues encountered by PSP and Leap but fell down majorly when developers grew to dislike it.

**Code Climate** is a specialized software which allows software engineering projects to be analysed. It allows code to be checked and reviewed, to make sure that it is clear and understandable, that it doesn’t repeat itself and that it is easily modifiable and reusable. The benefit that it gives organisations is that it allows them to thoroughly analyse their code through full testing coverage and also by allowing them to reduce their code complexity.

Another major benefit is that it tests code for maintainability which is an important measure of product as I mentioned earlier. It gives them a full and comprehensive analysis of both their product and process metrics and makes sure that any updates added give increased value and are suitably tested. Code Climate can also be used in conjunction with Github, on which it can show things like code coverage, technical debt and a progress report.

# available algorithmic approaches

Machine learning algorithms can be classified into two main areas; supervised and unsupervised.

Supervised learning is when there is both an input and an output variable, with specific mapping included which matches the input variable to the output variable. The main belief underpinning supervised learning is that you will be able to approximate the output value so well that you will be able to compute the output for new input variables as well as existing ones. The reason that supervised learning is named as such is that we know the answer that we are expecting to get from the input data and, due to this, we are able to monitor the algorithm and make amendments if they are required.

Linear Discriminant Analysis (LDA) is a supervised learning algorithm which allows you to classify multivariate data by groups. LDA is defined as the process of defining a classification decision bound and classifying multiple instances of this data by the boundary.

K-Nearest Neighbours is a non-parametric method of classification which is notable because it makes no assumptions about the spread of data within each class. It is implemented by getting the new point for which a prediction is required and then classifying it based on available data. The number K of nearest neighbours is chosen to classify the point and the machine will then choose the variable which dominates the choice of the K nearest observations to the new point.

Unsupervised learning differs from supervised learning in that the input variables have no set output variables. This means that unsupervised learning requires you to model the set of data in order to learn more about it. Obviously, due to the fact that the process can’t be monitored, there are no correct or incorrect results. There is no guarantee what the data will look like and it must be analysed by an appropriate algorithm.

K-Means Clustering is a clustering algorithm which is very useful as it allows data to be divided into cluster based on its characteristics. First you specify K, the number of clusters you want to analyse and assign a data point randomly to each cluster. When they are created, you compute the central data point in each cluster and then reassign the data points to each cluster centroid that it is closest to. This process is repeated until it can’t be improved any more.

These algorithms do not require a computer however automating them allow analysis to be done faster. These algorithms are not suited to soft data, which is why they haven’t taken over analysis as a whole. There is still a human aspect, but the door to more new opportunities in the field of machine learning has been opened.

# ethical concerns

Most companies engage in measuring performance in order to get a greater understanding into the productivity and performance of their company so that they can identify inefficiencies and make improvements if needs be. This raises a number of ethical concerns to be aware of. These concerns vary from invasion of developer’s privacy to worries regarding the amount of information stored.

These concerns should be raised and addressed before beginning any monitoring type work. This is essential in order to protect employee rights and if it is not addressed correctly, it can negatively affect employees. This, in turn, can affect the company by opening them up to the possibility of a court case, or costing them staff.

With the introduction of the new GDPR legislation companies need to be very careful to follow the rules in order to avoid legal action. The legislation states that firms can be fined the higher of €20 million or 4% of annual turnover for a breach.

In my opinion, when companies are looking to expand their analysis processes, they take the advice and opinions of their staff on board and make it a collaborative effort. If this isn’t done, employees tend to revolt against the new processes and there is significant backlash. Clear and open communication can help to ease these sorts of transitions.

# conclusion

In conclusion, this report assesses the process of measuring software engineering and ethical implications of it. The development of AI and the advancement of computational software have ed to incredible improvements in the field of software engineering. Due to the benefit and improvements that monitoring can lead to, it is unrealistic to think that companies aren’t going to monitor and assess the work of their employees. However, as further developments in the field are made, further legislation which builds on GDPR will be required.

Organisations want to increase their employee’s productivity and make the working processes more efficient in order to increase profit. On the other hand, employees want to learn, improve and get job satisfaction. While these two goals overlap to a reasonably large extent, measuring employees can cause friction between the two parties, leading to the goals of the two parties going out of alignment.

# bibliography

* Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.
* Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. – no loac
* Grambow, G., Oberhauser, R. and Reichert, M (2013) Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. Int'l Journal on Advances in Software, 6 (1 & 2). pp. 213-224. -no
* Dittrich, Andrew, Mehmet Hadi Gunes, and Sergiu Dascalu. (2013) "Network Analysis of Software Repositories: Identifying Subject Matter Experts." Complex Networks. Springer Berlin Heidelberg. pp. 187-198.
* P.M. Johnson et al., “Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined,” Proc. 25th Int’l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.
* Johnson, Philip M., and Hongbing Kou. "Automated recognition of test-driven development with Zorro." Agile Conference (AGILE), 2007. IEEE, 2007.
* W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.
* 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.
* Hassan, A.E. and T. Xie, Software intelligence: the future of mining software engineering data, in Proceedings of the FSE/SDP workshop on Future of software engineering research2010, ACM: Santa Fe, New Mexico, USA. p. 161-166.
* Di Penta, M. (2012) Mining developers' communication to assess software quality: Promises, challenges, perils. In Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on. IEEE.
* E. B. Passos, D. B. Medeiros, P. A. S. Neto and E. W. G. Clua, (2011) "Turning Real-World Software Development into a Game," Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on, Salvador, 2011, pp. 260-269.,
* L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," Games and Software Engineering (GAS), 2012 2nd International Workshop on, Zurich, 2012, pp. 5-8.
* Silverman, Rachel (Nov 2, 2011). "Latest Game Theory: Mixing Work and Play — Companies Adopt gaming Techniques to Motivate Employees". Wallstreet Journal.
* http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf
* R. Want, A. Hopper, a. Veronica Falc and J. Gibbons. The active badge location system. ACM Trans. Inf. Syst., 10(1):91–102, 1992.
* Brooks, F. and Kugler, H.J., 1987. No silver bullet (pp. 1069-1076). April.
* Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. 2004. How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Trans. Softw. Eng. 30, 12 (December 2004), 889-903
* Johnson, Philip M., et al. "Improving software development management through software project telemetry." IEEE software 22.4 (2005): 76-85
* Cockburn, A et al. Agile Manifesto, http://agilemanifesto.org
* Pfleeger, S.L., Jeffery, R., Curtis, B. and Kitchenham, B., 1997. Status report on software measurement. IEEE software, 14(2), pp.33-43.
* Beck, K. (2000). Extreme Programming Explained. Addison-Wesley.
* Brownlee, J. (2016, March 16). Supervised and Unsupervised Machine Learning Algorithms. Retrieved from Machine Learning Mystery: https://machinelearningmastery.com/supervised-andunsupervised-machine-learning-algorithms/.
* Citizens Information. (2018, September 5). Overview of the General Data Protection Regulation (GDPR). Retrieved from Citizens Information: http://www.citizensinformation.ie/en/government\_in\_ireland/data\_protection/overview\_of\_general\_data\_protection\_regulation.html
* Code Climate. (2018). About Us: Code Climate. Retrieved from Code Climate: https://codeclimate.com/about/
* Gabry, O. E. (2017, March 17). Software Engineering — Software Process and Software Process Models. Retrieved from OmarElGabry's Blog: https://medium.com/omarelgabrysblog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc.
* Johnson, P. M. (2013). Searching under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.
* Littlefield, A. (2016, Septemeber 12). The Beginner’s Guide To Scrum And Agile Project Management. Retrieved from Trello: https://blog.trello.com/beginners-guide-scrum-and-agileproject-management.
* Rouse, M. (2016, November). Software Engineering Definition. Retrieved from WhatIs: https://what.is.techtarget.com/definition/software-engineering
* VersionOne. (2018). 12th Annual State of Agile Report.
* Xenos, M., & Stavrinoudis, D. (2008). Comparing internal and external software quality measurements. Greece: ResearchGate.
* Carpenter, W., 2018. Code Climate: How it Works and Makes Money. [Online] Available at: <https://www.investopedia.com/articles/investing/022716/code-climate-howit-works-and-makes-money.asp#ixzz5XCtdZq26>
* Glass, R., 1994. A tabulation of topics where software practice leads software theory. Journal of Systems Software, Volume 25.
* Hackermoon, 2018. 10 Soft skills every developer needs. [Online] Available at: <https://hackernoon.com/10-soft-skills-every-developer-needs-66f0cdcfd3f7>
* Houlding, B., 2018. [Online] Available at: www.scss.tcd.ie/Brett.Houlding/ST3011
* Humphrey, W. S., 1995. A Discipline for Software Engineering. Addison-Wesley.
* IEEE, 2018. What is Computational Intelligence? [Online] Available at: https://cis.ieee.org/about/what-is-ci
* Kuchler, H., 2018. Apple boss says tech industry regulation ‘inevitable’. [Online] Available at: https://www.ft.com/content/b8bc0108-eb51-11e8-89c8-d36339d835c0
* Lowe, S. A., 2018. 9 metrics that can make a difference to today’s software development teams. [Online] Available at: https://techbeacon.com/9-metrics-can-make-difference-todays-softwaredevelopment-teams
* Lowe, S. A., 2018. Why metrics don't matter in software development (unless you pair them with business goals). [Online] Available at: https://techbeacon.com/why-metrics-dont-matter-software-developmentunless-you-pair-them-business-goals
* Pomeroy-Huff, M., 2009. The Personal Software Process, Body of Knowledge, Version 2.0.
* Sommerville, I., 2011. Software Engineering. 9th Edition ed. s.l.:Pearson Education.
* Torres, O., 2018. 7 Short-Term AI ethics questions. [Online] Available at: https://towardsdatascience.com/7-short-term-ai-ethics-questions32791956a6ad