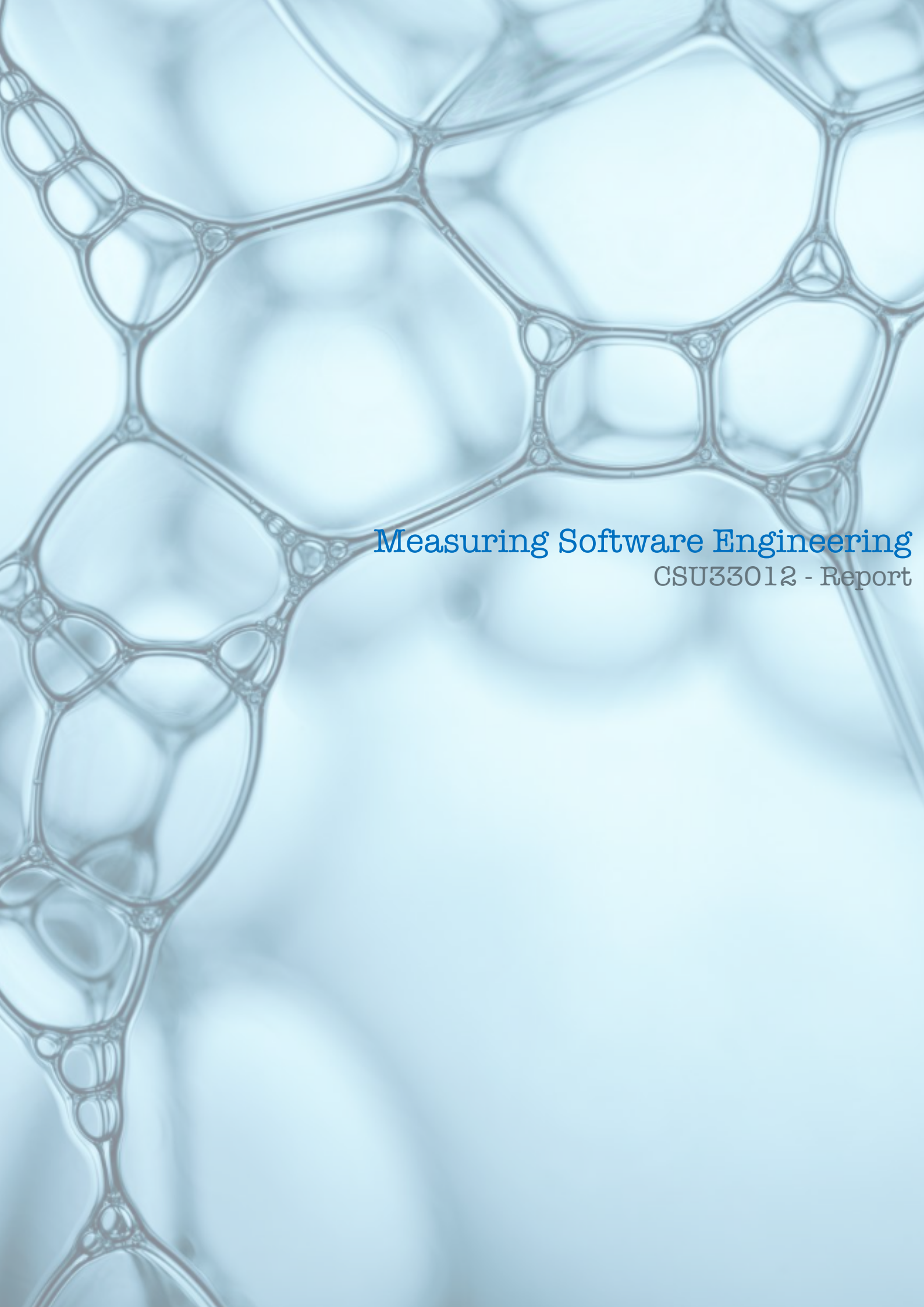


# Measuring Software Engineering

CSU33012 - Report

## Abstract

This report aims to discuss the wide-range of techniques used for assessing software development. The following topics will be discussed: Measurable Data, platforms and software used for measuring data, algorithmic approaches available and ethical concerns associated with this kind of assessment.

All information is derived from various academic papers, and material from the CSU33012 lecture.

## Introduction

Software engineering is "the process of analysing user needs and designing, constructing, and testing end-user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development." (What is Software Engineering? - Definition from Techopedia, 2020). Although a relatively new industry, software today dominates every facet of our life; from the OS we use on our phone to set an alarm, to the app we use to check if we need to bring an umbrella. Although relatively infantile, the industry has seen an exponential growth and evolution in recent years.

As with hiring a construction company to build a house, a software development studio has much the same structure. Plans are devised by an architect to fit the customers' needs, qualified builders are required to assemble the house, raw materials are required for construction, and input from the customer is constantly taken into account to ensure it fits their exact needs. Where they differ is a problem faced by software engineers.

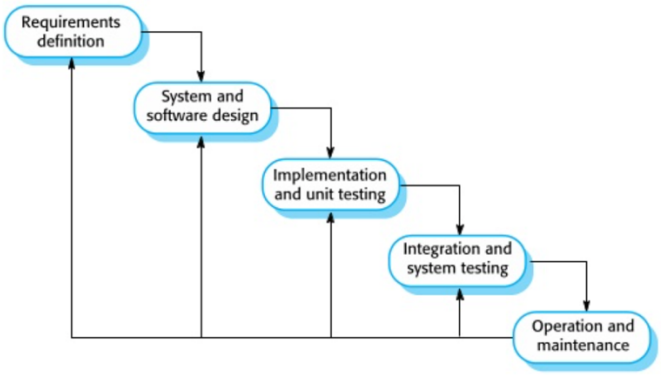
How do we assess the efficiency of the development process? How do we compare one process to another? How do we begin to put together a cost-structure for such a process? How do we rank one software engineer against another for further improvement. The goal of this report seeks to explore some methods companies use to alleviate this problem and optimise their efficiency.

Although some of the assessment criteria explored in this report may seem arbitrary, this report does not seek to find a 'definitive' process. This report merely aims to explore the concept, and help provide some clarity on processes some companies use in the real world.

## Software Development Processes

Before delving into the assessment of software development processes, I will first explore some processes used by companies. Now, one must understand there is no "one-size-fits-all" process. Particular processes suits particular projects better. This is mainly determined by 3 things:

- The size of the project
- How Plan-Driven the project is
- The flexibility required for such a development




All processes, to some degree, undergo the process shown above. The application of the software is defined and the final requirements of the software are outlined, the software is designed, tested, integrated and given to the user. The user then gives their verdict on the software, what is needed to improve and the process repeats anew. When and how often each stage is revisited is what separates development processes.

In choosing development processes, the driving-factor of your decision is based on how documented or agile you wish the process to be. Osetskyi (2017) refers to this as the "Software Development Life Cycle Model" (SDLC). He refers to the most common procedures: Waterfall, Iterative, and Spiral. For Instance, the Waterfall Method may be more suitable for large projects involving multiple organizations. In this process, a plan is pre-defined and each stage is undertaken sequentially. The software is not ready for use until the final stage here. This differs from more agile processes such as the Iterative and Spiral methods. In the Iterative model, specification, development, and validation are inter-weaved. This increased flexibility allows changes to be made more seamlessly. The product is constantly been given to the consumer and changes are made based upon their feedback. In a spiral, one or more stages of other processes are combined to create a process more suitable to a certain task. What these processes gain in agility they lack in traceability. In these processes changes are not as well documented. This would not be conducive to say, developing a massive piece of software undertaken by multiple teams worldwide that are not in contact with each other.

As you can see, processes are flexible and myriad. For all, the basic fundamental building blocks of development must be undertaken. But costs vary dependent on the process. An added function in the Waterfall method must go through 3 more stages whereas for an Iterative model it doesn't. Can a model be made for calibrating this?

## Measurable Data

Data Analysis is something which is ubiquitous throughout all industries. One means of assessment utilising is from using software metrics. A software metric is the "term used to describe the wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterise properties of software code through to models that help predict software resource requirements and software quality" (Fenton and Neil, 1999).

Software engineering differs from most fields in the fact that it is not easily definable. Productivity cannot merely be determined by output or hours worked. Software metrics were introduced to provide a means of improving performance while minimising waste.
All metrics used must fulfil a certain set of criteria
They must be:

- Simple and quantifiable
- Unambiguous
- Consistent across all different processes and programming languages

# Ways of measuring software development process

### 1. Lines of code/Source Lines of Code (LOC/SLOC)

Measuring quality by the amount of lines of code is a primitive type of assessment that has long been a standard for measuring the software development process. Software development promotes efficiency and transparency across the industry. Bhatt, Tarey and Patel (2012), express multiple concerns in their paper 'Analysis Of Source Lines Of Code(SLOC)', namely, how a metric that promotes inefficiency in programming is inherently flawed. Through this means of assessment, a programmer is incentivised to write a greater number of less efficient lines, or write redundant lines that grant it a superficial higher quality using this metric

### 2. Number of Commits

Although measuring code by the frequency and number of commits can provide some reassurance to the client, as it lets them know the project is still getting attention, it is simply a flawed approach. The size and frequency of commits do not give any indication of the completion of the project. It also gives no indication of the quality of the work. Programmers have their own personal tendencies, and what should and shouldn't be commit is subjective. It also does not take into account that code can be removed during a commit, and quite often is. Overall, number of commits should solely be used as an indication of activity on the project

### 3. Code Coverage

This code aims to measure the total coverage of test cases made by the developer. Code coverage measures the percentage of lines of code executed by the program from the tests as a percentage of the total program. Generally, code with a high code coverage have less likelihood of having an undiscovered software bug. This gives some indication to the developer and client of the overall thoroughness of the code.

### 4. Code Churn

Code churn measures the evolution of code. It equates to the moving average of number of lines added against number of lines removed. In lamen's terms, a project with high code churn signifies a difficult project. It also signifies that the developer spends a lot of time fixing their own mistakes. It is a good comparison metric, as it can find inefficiency among developers and rewards developers if they find a more efficient way of performing a function.

### 5. Lead Time/Cycle Time

Lead time can be defined as the time elapsed between the identification of a requirement and its fulfilment. Similarly, cycle time can be defined as the time elapsed between the commencement and fulfilment of a project. This metric is easily quantifiable, simply just from seeing date of commencement of project and conclusion date. Lead time is also very consistent across projects. Simply giving lead times of previous projects of similar complexities to new clients presents a level of transparency which is attractive to new clients. It also allows you to see the efficiency of teams, or even individuals. By keeping track of these variables, it allows inefficiencies to be identified and eliminated. Some concerns arise with the definition of 'fulfilment' of a project. Fulfilment is a very subjective term, and comparison across different companies is impractical, but when used in isolation by one team, is a very effective means of improvement.

### 6. Refactoring Rate

Refactoring is the re-use of old code. Reusing old code is more efficient as the implementation is already tested and successful, therefore less bugs arise. Is it very easily assessable through looking at old commits from previous projects. Problems arise when the implementation does not work well initially, and the developer spends too much time trying to essentially 'shoving a square peg in a round hole'.

### 7. Bug Fixing

Bugs are natural in a software development cycle. Developers should not be alarmed by the number of bugs that are found in their program. However, the severity of these bugs should be taken into account. Less time spent bug fixing would generally be a sign of a more experienced developer. When bugs are logged, similar bugs arising in the future are solved quicker as their solution is well documented. This enhances the productivity of the team as a whole.

## Platforms Available

Once data has been collected, it must be analysed to be utilized efficiently. Rapid analysis facilitates a faster response, granting competitive edge to companies that grow and adapt quickly. Software metrics, as some discussed above, are fairly easily measured, but can be difficult to interpret and apply to help improve efficiency. There are myriad platforms used for analysing, of which are ubiquitous throughout the software development industry

### *1.) Personal Software Process (PSP)*

PSP was first introduced by Watts Humphrey (2000). Watts was a big activist for structured development processes and punctilious tracking of actions. He believed this would help engineers streamline their process immensely. Ultimately, improvement is based upon the actions of the software engineer, but PSP provides a basic framework for measure and monitor their performance, and adjust their process as issues arise. The PSP uses forms as a means collecting information from the developer. A developer logs his project plan summary, records any defects found, and fills out a checklist of required functions. This method helps institute proper coding standards across the team, errors are more efficiently logged, and helps with self-evaluation and improvement.

Problems arise in the manual input of these forms. Proper software metrics are intended to be consistent and unbiased. As these forms are filled out individually, data can sometimes be manipulated to draw incorrect conclusions, as discussed by Johnson (2013). This partly is why the LEAP toolkit was made.

## 2.) Leap Toolkit

LEAP toolkit "attempts to address the data quality problems we encountered with the PSP by automating and normalizing data analysis" (Johnson, 2013). The LEAP toolkit still implements manual input akin to PHP, it subsequently automates PSP and provides additional tools for analysis, such as regression analysis. It creates a repository of personal process data that developers keep with them as they shift from and to projects.

After many software metric analysis metrics tried and failed, it was discovered that manual input would be required to some degree. Human error could not be completely eradicated, merely minimised. By taking in user input, and standardising it somewhat using data analysis, a platform was created that was consistent while also remaining flexible. In the quest for truly unbiased analysis, a tool was constructed called Hackystat.

## 3.) Hackystat

Hackystat is an open source framework for collection, analysis, visualisation and interpretation of the software engineering process. (Hackystat, n.d.). It is easily integratable with code repositories such as GitHub and Bitbucket, making it popular among users for its portability. Hackystat collects raw data from the developer and sends this to a server to be analysed. Hackystat collects both client and server side data. Hackystat collects data unobtrusively, meaning the user may not even be aware that data is being collected from them. It gathers the same data as PSP, such as the time spent on the project, the size of the project, the number of commits, but also some more advanced data such as code coverage and complexity.

The in-depth data being collected gives managers a succinct insight into the workings of the developer, but developers have also raised concerns over the discreet collection of their data. With data being perpetually collected, managers gain a commanding position over their employees which brings discomfort to developers. This has prevented Hackystat from fully establishing itself as *the* industry standard for software development process analysis

| Project (Members) | Coverage | Complexity | Coupling | Churn | Size(LOC) | DevTime | Commit | Build | Test |
|---|---|---|---|---|---|---|---|---|---|
| DueDates-Polu (5) | 22.0 | 1.5 | 5.6 | 73.0 | 490.0 | 4.2 | 5.0 | 18.0 | 224.0 |
| duedates-ahinahina (5) | 50.0 | 1.8 | 4.6 | 283.0 | 2033.0 | 5.2 | 3.0 | 6.0 | 12.0 |
| duedates-akala (5) | 38.0 | 1.4 | 7.2 | 329.0 | 3400.0 | 11.4 | 14.0 | 29.0 | 71.0 |
| duedates-omaomao (5) | 71.0 | 1.6 | 4.6 | 3488.0 | 5264.0 | 13.8 | 70.0 | 36.0 | 38.0 |
| duedates-ulaula (4) | 64.0 | 1.5 | 4.8 | 2443.0 | 4039.0 | 8.1 | 36.0 | 37.0 | 25.0 |

*A Software ICU (intensive care unit) display based on Hackystat. The Software ICU assesses a project's health both alone and in relation to other projects.*

## 4.) Code Climate

Code Climate is a software program that that offers advanced data analytic tools to measure the software development process. Code Climate "incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous, and ubiquitous." (Code Climate, 2018). Code Climate performs automated analysis of code, identifying logical flaws in the code, inefficiencies, and any glaring security flaws. Code Climate seamlessly integrates with Github, pulling some data from their servers such as technical debt, and can also we used to set acceptable levels of code quality.


Code Climate allows code to be reviewed to ensure there is no repetition and is easily understood. Various qualitative and quantitative metrics can be used to summarise various aspects of the code. Overall, Code Climate helps ensure full test coverage, decrease code complexity, simplifying reusing of code and ensures any additions to the source code have value.

## 5.) Github

Github is a version control software and collaboration platform used worldwide by over 40 million software developers. Github is open-source and may be used to track the software engineering process. Using the REST API, certain metrics can be found such as number of commits, frequency of contribution, and number of developers working on the project. Github is well integrated with the majority of industry standard assessment tools, and tools such as GitPrime allow visual dashboards to be constructed detailing certain metrics.

## 6.) Other Platforms

Myriad other analysis platforms exists, indicative of the popularity of this kind of assessment in industry. These include : Codacy, Waydev, Velocity, Sonar etc.


# Algorithmic Approaches

Naturally, manually entered data has long since been replaced as the industry standard by more automated processes. Automated processes are more efficient, standardised, unbiased and informative. All software mentioned above would use various algorithms to perform their assessment. Algorithms also help identify various structures in the code and process.

The industry has seen a distinct rise in efficiency since the advent of Artificial Intelligence and Machine Learning. Artificial Intelligence is intelligence displayed by machines. The goal of AI is to attempt to mimic human thinking functions such as learning and problem solving. Through AI we can teach a computer to perform a simple function without hard-coding the steps the program should take. Most popular algorithmic approaches would employ some element of machine learning. In this section I will discuss some types of machine learning used by developers to automate their software development process analysis.

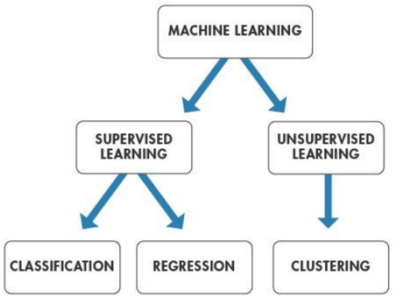
## Types of Machine Learning Algorithms

Macgine Learning can be divided into the following 3 main methods (Brownlee, 2019):

1.) Supervised Learning

2.) Unsupervised Learning

3.) Reinforcement Learning




## Supervised Learning

Supervised learning is where the algorithm generate s afunction that maps input to defined output i.e $y = f(x)$ where y is the output and x is the input. This is known as supervised as the data analyst defines to the algorithm what conclusions it should be drawing from the algorithm. Training the data is done by giving mock data with relevant inputs and outputs. The machine learning process begins here as the algorithm makes predictions based upon the data and corrects outliers. This process is repeated until a desired margin of error is met. Some supervised learning methods are as follows:

### 1.) K-Nearest Neighbours

K-Nearest neighbours is a non-parametric methods of classification. Initially, "training data" is fed into the system of known classification. This algorithm then uses the existing data to make a prediction for and classify new data points. It does this by analysing the "K-Nearest neighbours" to this i.e if $K = 3$ then the machine will choose the variable which dominates the 3 nearest observations to this point (White, 2020).

### 2.) Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a classification technique that allows you to place your multivariate data into classification groups. LDA is the process of defining a classification decision bound, say 0.2, and classifying multiple instances of this data by the boundary. This method is widely used with machine learning and artificial intelligence. (White, 2020)

## Unsupervised Learning

Although useful in some regards, Supervised Learning methods are somewhat restrictive in what they can and can't do. Unsupervised Learning methods are comparatively far more flexible. The aim of machine learning is to have a computer learn how to accomplish a function without a developer instructing it how to do so. This allows the algorithm to identify complex processes and patterns the developer may not be aware of. Types of unsupervised algorithms are as follows

### 1.) Principal Component Analysis

Principal Component Analysis (PCA) is a means of compressing data with many variables/dimensions into a few comprehensible variables with high correlation that accurately summarise the data. PCA is a method for re-expressing data so as to reveal its internal structure and explain its variation through the use of a few linear combinations of the original variables. The goal is to identify relationships between variables that explain most of the data e.g developers with higher code churn may have a longer lead-time (White, 2020)

### 2.) K-Means Clustering

Characterising elements is a crucial component of data analysis. Being able to assign a label to something based on their characteristics is an efficient process which when dealing with large amounts of data decreases the margin of error. K-Means clustering is an iterative clustering algorithm. Initially, the number of clusters is specified and a data point is randomly assigned to each cluster. Once the clusters are created, you must compute the central data point in each cluster and then re-assign the data points to each cluster centroid based on their proximity to a cluster. The central data point is then recalculated and each data point is reassigned based on their proximity again. This process is repeated until there are no improvements possible. (White, 2020)

## Reinforcement Learning

Reinforcement learning algorithms are centred around decision-making. Certain weightings are assigned to each decision based upon their outcome, assigning a reward or penalty to each action. Ultimately, the goal of the algorithm is to maximise reward and minimise loss.

Reinforcement Learning algorithms strive for perfection, and this is why their application in the software development process is not as commonplace as you may think. These algorithms don't take into account natural human error or judgement. These algorithms are not suited however to softer data, such as social interaction, and it is for this reason that machines are not taking over analysis as a whole. While these algorithms are very sophisticated, they are not sophisticated enough to mimic human thought. Human input is vital to the software development process, and disregarding this in favour of the 'infallible' nature of a machine learning algorithms may result in limitations in the future.

## *Ethics*

As you can see, the software development process is greatly enhanced through the application of data collection and analysis. This kind of assessment has only recently seen application in the software development industry, and because of this, very little legislation has been brought in regarding the collection and application of developers data. The real matter at hand is – Is this all ethical?

As I have outlined above, data analysis is so prevalent and unobtrusive that most people are not aware it is happening. An ethical collection is raised as the data is only being collected for the betterment of society. While the company ultimately aims to increase their profits by improving their software through this data collection, conversely it can be argued our data is being collected to provide us an enhanced service. But where is the line drawn at what can and can't be collected?

### *Data Collection*

The method of data collection can raise a number of ethical concerns. As previous discussed with Hackystat, the unobtrusive, constant and discreet data collection can make employees feel uncomfortable. For an employee, the added pressure of knowing your manager is watching your every action may hinder productivity, and industrial relations disputes may arise. Intrapreneurship may be harmed as an employee may be scared of the scrutiny they would get by doing something outside of the accepted standard, and as their actions are consistently tracked, there is no escaping this consequence.

As Sommerville (2011) claims, confidentiality, honesty, and integrity are key to the performance of a software engineer. While an employer does have access to personal information of a software developer, steps must be taken to ensure the only data collected is that which helps improve decision-making, identifying problems and find solutions in the business. A balance must be struck between privacy for the employee and gathering impactful data for the betterment of the firm.

Transparency is an important aspect of data collection. People should be informed of the usage of their data, and whether it will be passed on or held internally within the firm. It gives people some comfort to know the firm is not profiting as a result of passing on their data, but on applying it internally to enhance productivity within the firm. When people know they do have a choice in how their data is collected, it provides some reassurance which counteract the concerns I have discussed above.

### *Data Regulation*

In May 2018, the EU released the General Data Protection Regulation (GDPR) to replace the previous Data Protection Act (DPA). As a result of this, organisations must place an emphasis on "transparency, security and accountability by data controllers and processors, while at the same time standardising and strengthening the right of European citizens to data privacy" (Dataprotection, 2018). Breaching this regulation makes a company liable for a fine of either €20 million or 4% of annual turnover, whichever is higher. This standard of data privacy seen now is unprecedented, as companies do not want to leave themselves open to the financial turmoil as a result of a violation. This legislation is particularly pertinent to software development firms, as little standards have been implemented in this industry regarding data collection.

### *Data Usage*

Management must ensure data analysis is fair and impartial and does not promote improper coding practices. For example, being assessed for the number of lines of code you write, as discussed above, is inherently flawed as the industry should promote efficiency and transparency. Recording someone's total number of hours worked on a project may promote excessive overtime, as seen with 'Crunch Time' and burn-out currently plaguing the game industry. The benefit of data is maximised when it is used fair and impartially.

Data should not be used to discrimate in any way, shape or from. For instance, the ethnicity of a developer should not influence his decision for promotion. If data being collected signals an employee may have a disability they did not declare in their application, this information should not be used in any decision-making process

## Conclusion

In conclusion, this report has explored the software development process and the means of assessing it and improving it. Software Development is still a relatively new industry, but one that has grown exponentially since its birth. While it drew inspiration from other production focused industries, having rigorous structured development processed such as the Waterfall method, the industry in recent years has rewarded agile and flexible development processes. There are myriad assessment methods and platforms available, each featuring benefits and drawbacks, and finding one that maximises the benefit to your firm is of the utmost importance. While these software metrics may be useful, I have discussed the ethical concerns relating to their collection and usage above. Although gargantuan, the software development industry is still in an infantile state in relation to the processes applied for development. There is still room to grow and I have no doubt in a few years the process will become even more streamlined and efficient

# *Bibliography*

Bhatt, K., Tarey, V. and Patel, P., 2012. Analysis Of Source Lines Of Code(SLOC) Metric. International Journal of Emerging Technology and Advanced Engineering, 2(5).

Brownlee, J., 2019. Master Machine Learning Algorithms. s.l.:s.n.

Code Climate. (2018). About Us: Code Climate. Retrieved from Code Climate: https://codeclimate.com/about/

Dataprotection.ie, 2018) - GDPR - Data Protection Commission - Ireland. [online] Available at: https://www.dataprotection.ie/docs/GDPR/1623.htm [Accessed 25 Nov. 2020].

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

Hackystat, n.d. A framework for analysis of software development process and product data. [Online] Available at: https://hackystat.github.io/ [Accessed November 2019].

Humphrey, W. S., 2000. The Personal Software Process (PSP). [Online] Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

Johnson, P. M. (2013). Searching under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.

Osetskyi, V., 2017. SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral. [online] Medium. Available at: <https://medium.com/existek/sdlc-models-explained-agile-waterfall-v-shaped-iterative-spiral-e3f012f390c5#:~:text=Taylor%2DJames%20Mendiola-,SDLC%20Models%20Explained%3A%20Agile%2C%20Waterfall%2C,V%2DShaped%2C%20Iterative%2C%20Spiral&text=SDLC%20%E2%80%94%20is%20a%20continuous%20process,no%20one%20single%20SDLC%20model.> [Accessed 19 November 2020].

Sommerville, I., 2011. Software Engineering. 9th Edition ed. s.l.:Pearson Education

Techopedia.com. 2020. What Is Software Engineering? - Definition From Techopedia. [online] Available at: <https://www.techopedia.com/definition/13296/software-engineering> [Accessed 19 November 2020].

White, A. (2020, November). STU33011: Notes . Retrieved from Blackboard TCD. Accessed 26/11/2020.