**Measuring and assessing the software engineering process**

When it comes to measuring and assessing the software engineering process, the arguments for/against the idea are extensive. The old fashioned outlook of measuring performance can occasionally mistakenly be applied today when analysing a software engineering project. However, measuring software engineering is a modern concern, beginning in the 1960s, and I feel it is incorrect to apply the same principals as in other areas of work. Since 1960, there has been numerous methods applied to measure such a process and, given the right understanding of these methods, I believe they are highly useful. Some would argue that you cannot measure the process as the performance of a software engineer has no clear output. It is said that without a measure of output, it is impossible to measure productivity. I respectfully disagree and think that it has been proven to be possible.

The idea of ‘software productivity’ is a notion that has been discussed for decades, an apt definition is the ratio between the functional value of software produced to the labour and expense of producing it. There has been numerous studies that explain the importance of productivity in the software engineering process. It has been shown that its measurement is essential when it comes to the quality of a software engineering process. The idea is that if you don’t measure productivity, there is no room for improvement as “*you cannot control what you cannot measure”*.

Throughout this report I will be showing not only the different ways to measure productivity, but rather the various ways to measure/assess the **entire software engineering process.**

This report attempts to deliver a clear and concise outline of methods used to measure and assess in terms of:

* **Measurable data**
* **Computational platforms**
* **Algorithmic approaches**
* **The ethics concerns surrounding this kind of analytics**

**Life Cycle of a Project**

The life cycle of a software development project can be outlined in seven stages.

1. **Identify current problems.** Data must be gathered and questions must be asked about how a certain system can be improved.
2. **Plan.** The team must now determine the requirements of the new software and produce a breakdown of costs involved.
3. **Design.** How will the team carry out their plan? Produce software specifications or a *Design Specification.*
4. **Build.** Start to implement the design specification by generating the initial code and working from the previous steps.
5. **Test.** Must ask whether the team got what they wanted. Did they solve the initial problem?
6. **Deploy.** Start to use the project in the environment that it was intended.
7. **Maintain.** Keep working on the project until it is as close to a solution as possible.

In any given software engineering project, the calibre of data that will be analysed can vary tremendously. In other words, a substantial amount of data can be collected for any software engineering project. When collecting such data, there are certain measures used with the intention to improve the software process. Measurement is so vital in such a process as it allows room for improvement, strategy planning and insight into the current position of the project. These measures are used throughout the software development life cycle.

When a project goes through the seven stages of its life cycle, data is collected continuously. It is with this collected data, that the relevant software metrics must be used.

**Measurable Data**

Throughout this section, I will be analysing the techniques of measuring/assessing software engineering in terms of measurable data.

By definition, a software metric is “*A software metric is a standard of measure of a degree to which a software system or process possesses some property. Even if a metric is not a measurement, often the two terms are used as synonyms”.* Software metrics should be easily understandable. They should be:

* Simple and computable
* Consistent and unambiguous
* Applied with consistent units of measure
* Independent of programming languages
* Easy to calibrate and adaptable

There are different types of software metrics to use at for different priorities. Below, I will be outlining the different *productivity, process and quality software metrics*.

**Productivity Software Metrics**

Measuring the productivity of a project can spark debate, as I mentioned previously. However, as I am a believer of the measurement of productivity, below are examples of various productivity metrics. You would use these when your priority is to measure how productive your project is.

**Project or Sprint Burndown.** The idea of ‘burndown’ is work left to do versus time left to do it. Therefore, this metric is used to measure how the team/individual works.

**Ticket close rate.** This refers to the number of stories/story points you or your team solved during a certain period of time. Essentially, this metric is used to ensure that there is not a single task that is taking up an unnecessary amount of time. By analysing the data to apply this metric, it measures the productivity of the project.

**Lines of Code (LOC).** Already mentioned previously in this report, this metric is widely used to measure productivity. However, it is said to never use this metric to evaluate individual performance. This metric is used in teams, to understand where difficulties lie. For example, a huge bug fix could be in one line of code. This analysis helps improve productivity.

**Code churn.** Typically measured as *“the percentage of a developer’s own code representing an edit to their own recent work.”* To compute the code churn, first measure the LOC that were modified over a short period of time, and then divide by total number of lines of code added. Clearly, analysing this data would help measure the productivity of a project. If there is high churn, it could suggest that it is a difficult problem. Churn levels will fluctuate throughout a project, which is normal. It is within the analytics of the churn data, through this metric, that you can assess the productivity of the project.

**Refactoring rate.** The idea of refactoring is the re-use of old code. It is straight forward how one would begin to measure refactoring, perhaps by looking through old commits. The amount of refactoring that is done in a project, can be an indicator of the productivity of the project as it goes through its software development life cycle.

**New work.** Analysing how much new code is added to a project to assess productiveness.

**Process Software Metrics**

When the priority of the assessment is the performance of projects processes/workflow, process software metrics are used. The metrics below can be chosen at the software developers discretion.

**Lead Time/Cycle time.** If the priority is to ensure continuous delivery of data these two metrics are useful.

**Deployment frequency.** Provides similar information to above.

**Commit frequency or active days.** If you enforce your workforce to commit their code often, it helps analyse the performance of the project. Otherwise, you could pay attention to how many days someone is actively working on the project, which will also help analyse the same thing.

**Pull requests-Related velocity.** This metric is highly advantageous for seeing the amount of data that is being passed through the system. It can help measure the performance of the team as it shows how fast data is being processed.

**Work in progress**. This is an indicator to the software developer’s speed. Ideally, the figure should stay stable over time. If the value increases, it shows that you are tackling a problem that isn’t being addressed.

**Commit/pull request risks.** If you monitor an average commit/pull request, you can use this data to analyse how you/your team is performing. What is meant by the risks involved is that if a commit or pull request shows a large amount of changes, it could suggest a poor performance.

**Quality Software Metrics**

In this aspect of software metrics, rather than quality being the priority, the aim is to have a project that’s quality allows changes to occur with little to no disruption.

**Number of Bugs.** By tracking how many bugs a project has, you can assess/measure the quality of the overall project. If you find that a project has a large volume of bugs, clearly the quality of the project is sub-par. This data should be constantly tracked if quality is your priority.

**Change Failure Percentage.** This analyses the number of deploys resulting in a failure on the total number of deploys. If you find from analysing the data that the failure rate increases over time, it shows evidence of a poor quality project.

**Pull request quality.** This metric is used to measure how a team works together. Measuring the data that this metric collects helps show you if your team is improving with regards to programming quality.

**Test coverage ratio.** In other words, the ratio between total lines of code in the software project, and number of lines of code that all test cases currently execute. It is said that this ratio should be about 80%, so comparing this figure to the data collected can help assess the quality of the project.

**Mean Time Between Failures & Mean Time To Recover.** A software developer can use these metrics to assess quality by seeing how both of them vary over the life cycle of the process. If the MTBF remains high, there is evidence to suggest a low quality project. Also if the MTTR is high, the same evidence exists.

**Service Level Agreement.** This is similar to MTTR, but doesn’t limit itself to software failures. It extends to any type of bug.

**Defect Removal Efficiency.** Used to quantify how many defects were found by the user after product delivery, in relation to the number of errors found before product delivery. This metric has a similar purpose to tracking the number of bugs.

**Application Crash Rate.** How many times an application is fails, divided by how many times it is used. This metric could be used if you have a specific Application Crash Rate in mind, it assesses the quality of your program by comparing your results with your aim ACR.

**Defect density.** Can be broken down into both size oriented metrics and function oriented metrics. The first is a simple metric to use once you decide on what contributes as a line of code. It helps to assess a program by analysing the amount of defects in a project. Unfortunately, you cannot use this metric to compare projects written in different languages. The function oriented metrics refer to how much functionality the software offers. Functionality is not something that is easily measured, so this metric is not commonly used.

**Age of dependencies.** Tracking the age of dependencies can be useful to assess the quality of your code as older dependencies may require attention.

**Computational Platforms**

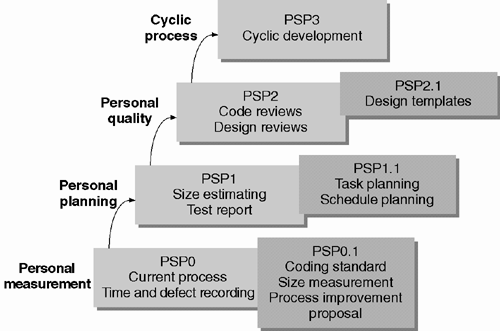
Software engineering, being a relatively new industry, has experienced a dramatic increase in the number of platforms available to write code. Over the years, companies have invented platforms, that use the metrics described previously, to allow for necessary insights into assessing the performance of a project. These platforms help to measure the project in a number of ways. If for example, a developer wants to assess a fellow team members performance, there must be a competent platform available for them to access relevant information. In the following section I will be outlining some of the most popular platforms used today, and these allow for easier measurement and assessment of the software engineering process.

I will be outlining the following five popular computational platforms:

1. **Personal Software Process (PSP)**
2. **LEAP**
3. **Hackystat**
4. **Codacy**
5. **Code Climate**

**PSP**

The Personal Software Process (PSP) is a method created with the intention to help software engineers better understand and improve their performance by tracking their predicted and actual development of their code. It is a computational tool that can be used with any programming language, on a number of different computational environments. Created by Watts Humphrey, it can be used for an individual’s assessment and has a level system that goes from PSP0 to PSP2.1, in order to allow improvement of a project.



**PSP0** introduces process discipline and measurement. It has three phases: planning, development and post mortem. A developer would establish a baseline or measurement (e.g time spent on program, size etc).

**PSP0.1.** The developer then develops a plan for improvement, introduces a coding standard and records ideas for individual growth through a PIP (process improvement proposal)

**PSP1** introduces estimating and planning. Based on the data gathered in PSP0, the developer estimates the potential size of a new program and produces a test report.

**PSP1.1** is when the developer starts to plan. They schedule planning with the estimations they have gathered through the previous levels.

**PSP2** introduces quality management and design. It also introduces two new phases: design review and code review. The aim of the PSP2 level is to remove and avoid defects. A developer will improve their process by measuring how many defects their project has. For the code reviews, engineers will use checklists to ensure their code is sufficient

**PSP2.1** involves design templates which are used to analyse the quality of the work produced.

PSP is an extremely useful tool for an individual to analyse their performance and assess their work. A developer would curate a plan for each project, record the development time, make note of the defects in said project, retain project data and then use this for future project planning. However, it has been shown that the PSP cannot be automated, so many people believe it to be insufficient as a platform for analysis. It has been shown by the University of Hawaii that its manual nature could result in human error in other words, incorrect conclusions. From this, LEAP was born.

**LEAP**

Standing for Lightweight, Empirical, Anti-measurement dysfunction, and Portable, LEAP is a project that was invented to adhere to the bugs in PSP. This toolkit allows for automation and it normalises data analysis. While it is an automated version of the PSP model, the user still inputs data at the beginning, but the rest is handled by the LEAP toolkit. It differs from PSP in a number of ways, one example being that the developer can remove the risks of measurement dysfunction by being able to control their data files. Leap creates a repository of the individuals data that an engineer can use on any number of projects. This repository allows for easy measurement and assessment of any individuals performance as they can use it between different projects. An engineer can use this data to assess themselves and improve.

**Hackystat**

While LEAP was a tremendous improvement to the PSP model, the same university that invented it admitted its flaws. It became clear that you could never fully automate the PSP model, meaning that it would always require significant manual entries from the user. Hackystat was then created.

“*A framework for collection, analysis, visualisation, interpretation, annotation, and dissemination of software development process and product data”.*

The Hackystat framework supports three communities: researchers, practitioners and educators. Users would attach ‘sensors’ to their development tools, which collect and send data to a repository designed by the Hackystat creators. This repository can then be accessed by other services to form assessments and analytics about the data involved. The goal of this method is that the work of a team/individual, can go through quality assurance and professional analytics within the ‘cloud’ of the internet. Essentially, Hackystat monitors programmers work and collects data which is uploaded to the Hackystat server. The platform goes as far as being able to track when a developer edits their code.

An issue that Hackystat experienced was that engineers were not happy with the way the platform recorded continuously as they worked. On top of this, Hackystat allowed for their engineers to have total access to any users information/data. Quickly, Hackystat became an ethics issue and was inevitably seen as a poor way of assessing/measuring a software project.

**Codacy**

In 2012 one of the more modern computational platforms available to measure software engineering, Codacy was founded. Supporting 28 different programming languages, Codacy is a platform to “*automatically identify issues through static code review analysis”.* In a study done by Codacy, they claimed that their product could improve code perception by up to 20% and optimise the code review process by up to 30%. Codacy provides: quality evolution metrics, per-commit analysis, Codacy dashboard, file metrics, file code quality summary, code quality goals, triggers and configuration files. All of these tools provided by the platform allow for assessment and measurement of a software project. It is known for its highly user friendly interface and is among the best platforms to measure a project.

**Code Climate**

Another modern platform, founded in 2011, Code Climate is used to determine the quality of code. After each git push, it analyses your code for complexity, duplication, and common smells to determine changes in quality and technical difficulties. It sends automated code review comments on the users pull requests, which allows for assessment and potential improvement. The automation of this platform allows the user to avoid self-assessment as it is provided through this useful tool.

**Algorithmic Approaches**

While I have discussed that the software engineering process can be measured by both measurable data methods, and automated computerised methods, there are other ways to approach it. Algorithms are used in the software engineering life cycle throughout, they are the nature in which a software runs and they can also be used to measure or assess. Manually identifying problems in a project can be difficult, as shown with the PSP model, so analysing the data through algorithms is a superior approach. By definition, machine learning is “*an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves”.* The main aim of machine learning is to allow the computer to ‘learn’ without having to be programmed to do so. It can be broken down into **supervised**, **unsupervised** and **reinforcement** learning. These are what I will be discussing below.

**Supervised machine learning algorithms** must involve information that has been learned in the past, and you then use this data to make predictions on future events. You have an input variable and an output variable, and it is with the algorithm that maps the input to the output. The goal is to approximate the mapping function so well that for when you have new input data, you can correctly predict the output data. It is known as supervised learning because you initially already know the correct answer, and the algorithm iteratively makes predictions that is corrected. Popular examples of supervised machine learning include linear regression, logistical regression, neural networks and support vector machines. Supervised algorithms are both classified and labelled, and can be grouped into **classification** and **regression.** Classification refers to when the output is a category, and regression is when the output is a real value.

**Unsupervised machine learning algorithms** are neither classified or labelled, this means that unlike supervised algorithms, the data is fed into a learning algorithm that then determines what to label them as. From this determination, a decision is made on what unsupervised algorithm to use. In other words, you have an input variable but no output variable, and the aim is to model the underlying structure in the data in order to learn from it. Unlike supervised algorithms, there is no correct answer, the algorithm presents its own analysis of the data. The idea of unsupervised learning is that the computer can identify problems and processes without any guidance. It can be broken down into **clustering algorithms** and **association** algorithms. Clustering problems are ones where you want to discover the groupings found in data. Association problems is when you want to associate the input variable to the output data, through an unwritten rule. For example, an association problem would be finding a relationship between two variables.

**Reinforcement learning algorithms** refer to taking suitable action to maximise reward. It is based on decision making and trains to make the best decision possible. Being a trial and error algorithm, reinforcement learning strives to learn from experience. In supervised learning, the algorithm is trained with the correct answer to learn from, while in reinforcement learning there is no correct answer for it to work from. It differs from unsupervised learning because there is a reward to be maximised. The output of a reinforcement algorithm depends on the state of the current input (aiming to learn from experience). Falling in between both supervised and unsupervised algorithms, reinforcement learning algorithms allow for a problem to be solved without the aid of any background knowledge. Reinforcement learning can be used in robotics and automation.

In summary, all of these algorithmic approaches are used to measure the software engineering process by analysing data and making predictions on future values.

**Supervised:** All data is labelled and algorithms learn to predict output data from input data

**Unsupervised:** Data is not labelled, algorithms learn to inherent structure from input data

**Reinforcement:** Trial and error algorithm used to make most efficient decision to maximise some reward.

**Ethics Concerns**

In just about every industry you can think of, there will be ethical issues, software engineering is no different. Becoming increasingly prominent in the world today, privacy is an important topic of discussion. Recently in the USA, it was found that the NSA were monitoring peoples devices. More and more people seem to be covering their laptop cameras over from fear of intrusion. As mentioned briefly when discussing Hackystat, there can be issues regarding invasion of privacy when it comes to platforms that analyse software projects. The public weren’t comfortable with their work being recorded non-stop, so the platform experienced problems due to these ethical dilemmas of privacy that they faced.

Unfortunately, many approaches to measuring and assessing the software process require the users data to be inputted into some technique, whether it be computational or not. As there has been a rising popularity of storing data in a ‘cloud’ environment online, the ethical concerns of data privacy are becoming more relevant today.

**Data sovereignty** is the concept that information which has been converted and stored in binary digital form is subject to the laws of the country in which it is located. This concept is linked with the security of users data and was created to tackle the ethics issues of privacy. Many countries have adopted these laws for decades, however with the rise in data analytics in recent times, the concept is becoming increasingly prominent. Following from this, countries like Russia, China, Germany, France, Indonesia, and Vietnam, all have strict data security laws. They require that their citizen’s data must be stored on physical servers within the country’s borders, in order to protect the misuse of such information. Data sovereignty was created to tackle the ethical concerns of data security - who should be able to see someone’s information? The ‘cloud’ environment that data is stored on sparks controversy which relates to data sovereignty. When someone’s data is uploaded to a cloud server, data sovereignty protects that information from being publicly accessible.

The introduction of the **GDPR (**General Data Protection Regulation) in May of 2018 introduced a new solution to the ethical issues surrounding data privacy. It allows for citizens of the EU to have more control over their data and aims to simplify the regulatory environment for business. This is so both citizens and business’ can benefit from the digital economy. It was created to adhere to the modern age of society, and regards how personal data is collected and stored. Under the terms of the GDPR, organisations must ensure that personal data is stored legally and securely, and they also must protect it from misuse and exploitation. These laws are enforced with penalties that companies will face if the GDPR is breached.

Facebook experienced such penalties when it was discovered that they breached the laws of the GDPR and abused users data. They did so by deceiving users by undermining how their personal information is shared and had to pay a fine of $5bn.

Another example of data privacy being breached, would be with the PRISM program of the NSA. It was designed in order to monitor devices over the world with the intention for security protection. It worked by collecting information and analysing the data for patterns of terrorist or other criminal activity. The problems arose when it became present that they were monitoring peoples emails, phone calls, video clips and photos of people all around the world. A public outrage occurred when this information came out, and it summarised the most topical ethical issue surrounding software engineering, by far.

To avoid ethical issues, a code of ethics has been created that has 8 principles.

1. **Adhere to the public.** A software engineer must act always in the interest of the public, and not have selfish tendencies. A software project must only be accessible to the public when it is considered safe and ready.
2. **Adhere to the client and the employer.** While a software engineer must act with the public interest in mind, they must balance the interest of the public, client and employer. This principle also regards keeping being fully confidential in their work. Software engineers must ensure they are using correct software that has been approved by their employer.
3. **Adhere to the product.** Software engineers must ensure that their product is of the highest possible standard before releasing it to the public. They must ensure they are qualified for any project that they work on. They must be able to provide adequate documentation for any project they work on.
4. **Adhere to their judgement.** Software engineers should maintain integrity and independence in their professional judgement and remain unbiased. They should never work on a project they feel incompetent in.
5. **Adhere to management.** Software engineers must promote an ethical approachto the management and the maintenance/development of their product. They must be aware of the companies policies and standards upon starting a project.
6. **Adhere to their profession.** Software engineers must advance the integrity and reputation of their profession consistent with the public interest. They must promote public knowledge of their profession and not promote their own interests. They must act ethically in their promotion and not violate any of these eight principles.
7. **Adhere to their colleagues.** Software engineer must be fair and supportive of their colleague and assist colleagues in personal development. They must show respect and give a fair hearing of their colleagues opinions.
8. **Adhere to themselves.** Software engineers must participate in life-long learning regarding to their profession and promote an ethical approach to their practices. They should improve their ability to create safe, reliable and useful software within reasonable time. They should always strive to improve.

When one of these eight software ethics principles are not adhered to, ethics concerns can arise. This code of ethics can be used to ensure that practices are fair and ethical. However, with this industry, there is room for debate and discussion. This is shown if we go back to the example of the NSA.

In terms of the PRISM program, while it can justifiably be said that this disobeys many of the ethical codes outlined above as it is a total intrusion of privacy, there is also another argument. It could be said that the NSA did have the publics best interest in mind as it was intended to monitor and stop any criminal activity. This is why it can be tough to narrow down software engineering with ethical issues, as there can occasionally be arguments for both sides.

**Conclusion**

In conclusion, over the course of this report, I believe I have sufficiently shown how it is possible to measure and assess the software engineering process. There are countless methods available to get some insight into your performance and your project, and it is through these insights that you can further improve and develop your skills as a software engineer. Whether it be through software metrics, online platforms, or algorithms, there is no shortage of ways you can measure and assess a project. Software engineering is a new industry, but it has taken the world by storm with its rapid development. As it is still growing as an industry, there will certainly be more and more ways to measure and assess the process in the future.

**Bibliography**

<https://anaxi.com/software-engineering-metrics-an-advanced-guide/>

<https://stackify.com/what-is-sdlc/>

<https://pdfs.semanticscholar.org/b73a/d86acf479a0d590d5331ea7977f37b5d34e6.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf>

<https://en.wikipedia.org/wiki/Personal_software_process#PSP_structure>

<https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf>

<https://explainagile.com/agile/personal-software-process/>

<http://www-public.imtbs-tsp.eu/~gibson/Teaching/CSC5524/CSC5524-PSP.pdf>

<http://grantome.com/grant/NSF/CCF-9804010>

<https://www.codacy.com/?utm_source=GoogleAds&utm_medium=ppc&utm_campaign=GoogleAdsBrand-Exact&utm_term=codacy&utm_device=c&gclid=EAIaIQobChMI1IzX6L605QIVTFXTCh1SUgiOEAAYASAAEgITG_D_BwE>

<https://dzone.com/articles/codacy-a-new-easy-to-use-code-quality-and-review-a>

<https://codeclimate.com/quality/>

<https://expertsystem.com/machine-learning-definition/#targetText=Machine%20learning%20is%20an%20application,use%20it%20learn%20for%20themselves.>

<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>

<https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>

<https://towardsdatascience.com/reinforcement-learning-an-introduction-to-the-concepts-applications-and-code-ced6fbfd882d>

<https://en.wikipedia.org/wiki/Data_sovereignty>

<https://www.computer.org/education/code-of-ethics>

<https://www.archive360.com/blog/data-sovereignty-and-the-gdpr-do-you-know-where-your-data-is>

<https://www.siliconrepublic.com/enterprise/facebook-breach-gdpr>