MEASURING SOFTWARE ENGINEERING

Ranjana Singh - 18321141

Introduction

This report, broken up into four parts, explores the use of software metrics in Software Engineering, its benefits, uses and development. A discussion surrounding common frameworks and technology platforms that are available can be found in Part II, followed by an overview of some of the algorithms that are in use today to process this data, with illustrative studies that have been conducted in recent times. Finally, the ethical implications and questions surrounding these processes are considered.

Part I: How Can We Measure Software Engineering?

What does it mean to measure software engineering?

Prior to considering the best methods, programmes and algorithms for measuring software engineering, we must first answer the question, "What does it mean to measure software engineering?" Naturally in engineering, we are familiar with the concept of quantifying the different and relevant aspects of entities, using various tools and standard units of measurement to achieve this. In other industries, the idea of qualitative performance metrics such as personality traits, leadership qualities etc are common.[1]

A comprehensive definition for software metrics was given by Norman E. Fenton and Martin Neil, in their 1999 paper titled, "Software metrics: successes, failures and new directions":

> *"`Software metrics' is the rather misleading collective term used to describe the wide range of activities concerned with measurement in software engineering."* [2]

Software in particular has three important metric categories: product, process and project metrics. As the name suggests, product metrics deal with conditions and aspects of the offered products themselves, typically their physical properties. Product metric measurements have traditionally relied on looking at things such as the length of code, its maintainability and complexity. Process metrics on the other hand take on the challenge of trying to chronicle the steps it takes to produce said product. Specifically, the typical things measured under this metric include the time required on average to edit code, total production time, error corrections, the total changes in a class and the nature of these changes among other items.[1] Project metrics on the other hand consider aspects such as the staffing pattern during the software development cycle, the schedule and total developers employed.[3]

What are the potential benefits?

The process of measuring software engineering, while being tedious and laborious at times (a topic that will be addressed later in this report) can provide a multitude of benefits for the organisation which opts to use them. These include but are absolutely not limited to[4]:

1. Increasing Return on Investment
2. Identifying and improving on underperforming areas
3. Efficiently managing workloads
4. Reducing costs
5. Increasing productivity
6. Assessing and prioritising specific performance goals
7. Communicating the status of current work efficiently and identify issues/bugs quicker
8. Identifying top performers in the firm and vice versa
9. Producing better quality products

Source: Stackify.com [4]

Brief History of Software Metrics

Software metrics has been around and in development since the 1960s, however for many years, and indeed even today, outdated and relatively inaccurate measurement practices have been implemented from which ineffective conclusions have been drawn. The most infamous of these is the crude, Line of Code (LOC) method whereby 1). the total lines a programmer writes and 2). the number of defects per line of code that are encountered are measured. This metric was primarily utilised for assessing productivity, accuracy, quality and ultimately related to gauging the size of a program. Later programme complexity started being taken into consideration and from the 1970s onwards, due to the diversifying nature of programming languages, new metrics were needed to measure this. The Goal-Question Metric (GQM) method, which was loosely based on the concept of Total Quality Management, suggested that without being goal driven, programs dedicated to obtaining metrics were worthless to pursue. Naturally, there exist metrics which are considered to be inefficient on the whole such as LOC on its own (consider the major variances in programming languages and also that not every line written in a programme is code) and errors are common when using metrics to develop accurate predictions related to quality, performance and worker productivity. In spite of this, the field continues to develop, having embedded itself in the conventional software engineering consciousness and is evolving to include capabilities beyond mere data processing and visualisation.[2]

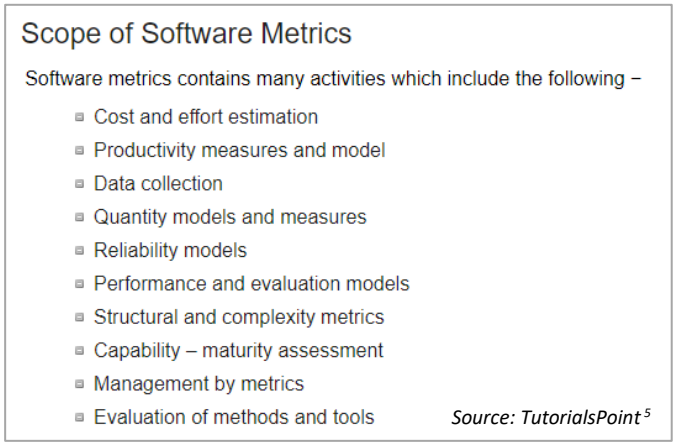
Existing Metrics

Recall the three main categories of software metrics:

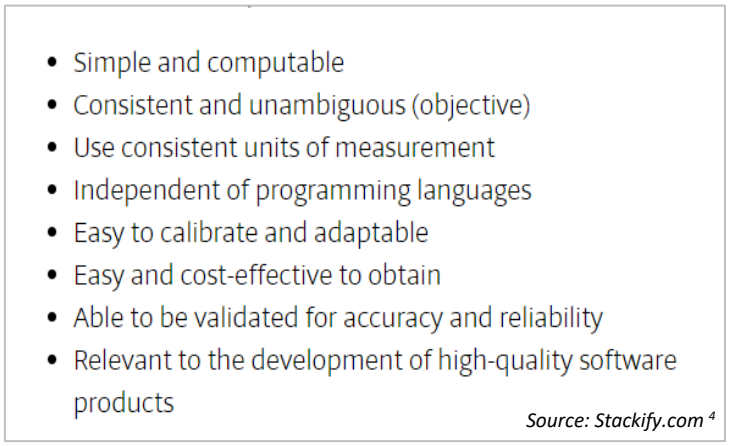
- **Product:** which focuses on the product size, complexity, design, performance and quality level
- **Process:** which focus on defect removal effectiveness, testing defect arrival, response time to fix the process
- **Project:** which focus on the project execution and include the number of software developers, productivity, cost and schedule

Source: TutorialsPoint[5]

Which are comprised of a variety of activities:



And which should have the following qualities:



Let us now consider some of the metrics that can be used and what they measure:

- **Cycle Time** – tracks how long it takes for bugs, features or tasks to go from one state to another
- **Sprint burndown** – rate it takes to finish work and how much is left to do (presented as a graph)
- **Velocity** – how many complete features or ready-to-test code, developers complete within a period
- **Open pull requests** – how many pull requests have yet to be responded to
- **Thoughtput** – tracked over time, throughput assesses total work output (everything that's ready to test)

Source: indexcode.io [6]

- **Lead Time** – the time it takes to take ideas and develop them into software
- **Active Days** – tries to assess the costs of interruptions and measures how much time developers contribute to coding
- **Efficiency** – how much of the code presented by a developer is actually productive
- **Churn** – the lines of code modified, added or removed during a certain timeframe
- **MTTR** – mean time to recover/repair the data or software
- **ACR** – application crash rate based on how many times it doesn't work to the total times it's used
- **Lines of Code/ Kilo Lines of Code** – measures of program size related to total code lines written

Source: Stackify.com [4]

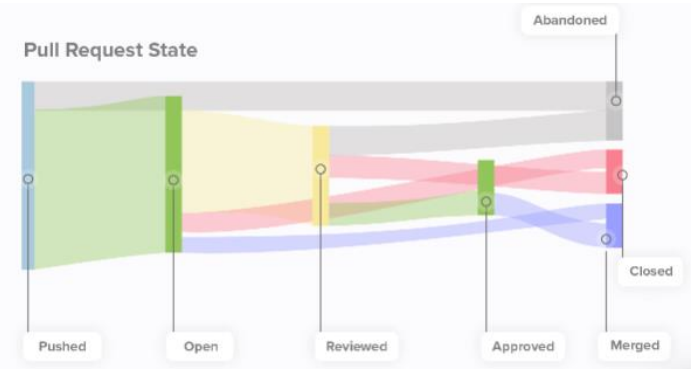
Part II: Existing Platforms and Frameworks

Having established some of that which falls under the broad definition of software metrics, it's also worth considering what modern tools and frameworks are used to try and create as seamless and effective a software engineering metrics programme as possible – especially in large firms.

Examples of Platforms:

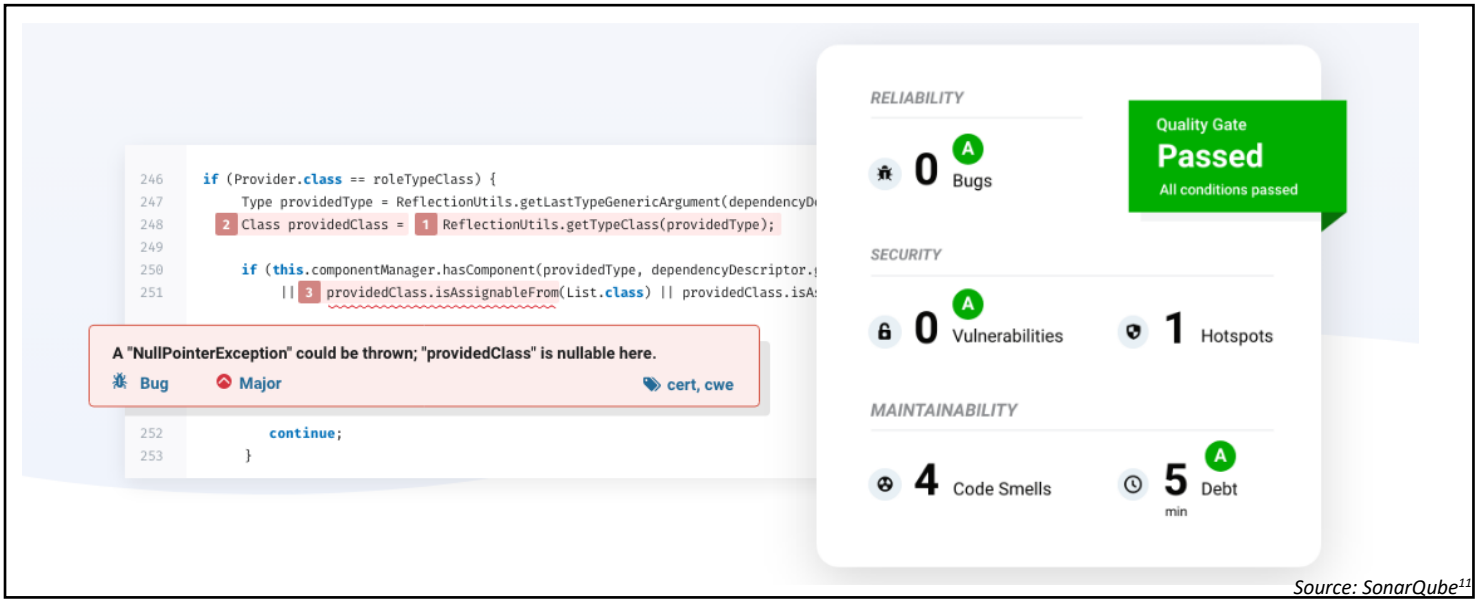
1). Pluralsight Flow: In January 2020, GitPrime became known as Pluralsight Flow a year after having joined Pluralsight with the aim of wanting "to help technology organisations reach their full potential." Originally having a sole focus on providing learning material to enterprises in order to equip their teams with the necessary technical skills, following the launch of Flow, Pluralsight had entered the Git Analytics market.[7] Flow mines Git data, reducing the need for human logging (which can be incredibly time intensive), while focusing on a wide array of metrics including, churn, commits per coding day, efficiency and impact. The site offers benchmark figures for various metrics while also integrating seamless visualisation features which helps managers and engineering team leads gain a better overall picture of both their production and team's status.[8] In October 2020, a new Delivery Module for Flow was announced by the company. This tool is the premier instrument using data from Jira to "measure human interactions that occur during the software development process". With other new features such as the Retrospective Report (a dashboard gathering ticket movement, flow efficiency and activity levels metrics) and Ticket Log (a report that, along with cycle time, queue time and blackflow rate presents a team's tickets), Pluralsight's offering of both software and code review analytics has been enhanced.[9]

2). Code Climate: Founded in 2011, after the founders made the decision that managers had struggles beyond the actual quality of the code, Code Climate, like Pluralsight, conducts analysis on data from GitHub repositories. Used by VMWare, Condé Nast and Instacart, Code Climate's principal product, known as Velocity, focuses on commit and pull request data to generate useful visualisations:[10]



Source: Code Climate [10]

3). SonarQube: Taking code quality measurements into account, SonarQube calls itself the "leading tool for continuously inspecting the Code Quality and Security of your codebases and guiding development teams during Code Reviews". Founded in 2007, and originally known as Sonar, SonarQube now covers 27 programming languages and is used by huge global corporations (and smaller companies alike) and has been distributed over 170,000 times. It offers many features including having the ability to align with the software pipeline currently being used by a team; to be easily integrated with Jenkins, Azure DevOps and TeamCity which are examples of continuous integration engines and to provide support for tools like Git, Subversion and CVS. The open source platform uses static code analysis (essentially analysing code against certain fixed rules) to catch bugs, security issues, duplicate code and code smells. The code is inspected as it is written and feedback can also be received as SonarQube analyses branches of GitHub, Bitbucket etc repositories and presents notifications within the pull requests themselves.[11]



Source: SonarQube[11]

4). Allstacks: Backed by Techstars, S3 Ventures, Polaris Partners and Hyperplane, Allstacks is a platform that uses AI and machine learning models to offer predictive forecasting and risk management capabilities for managers overseeing the software development process.[12] In June 2020, Allstacks became the first provider of these services to have cleared their AOC 2 Type II Audit, which certifies their compliance and commitment to user privacy, security and integrity throughout these processes.[13] Allstacks can gather data from Jira, GitHub, CircleCI and other stacks which would allow them to generate reports and forecast the progress, health and productivity of a team working on a specific project.[14]

Examples of Frameworks:

1). Personal Software Process (PSP): This is a software metrics framework which has a specific emphasis on the individual and the responsibility they have in tracking and improving their own work. There are different components to this process one being time management, where the software engineer has to keep track of the length of time they spent on various aspects of a particular project. Another component is planning which ensures that the developers are able prioritise, structure and create strategies to complete tasks more efficiently. Finally, four different levels exist in this process including:[15]

- **PSP 0:** Aspects of this initial level centre around personal development, basic size measures and coding standards

- **PSP 1:** This is the second level which has a major focus on planning

- **PSP 2:** At this stage, the quality and design of the code at a personal level with code reviews is considered

- **PSP 3:** The final level of this PSP process is concerned with refining how you work, improving your methods, logging these changes as you implement them

There are many benefits to PSP such as allowing software engineers to have better organisation, time management and planning skills. The quality of their work also improves, as well as the standards of their projects. However, the major drawback is that this is a very time-consuming endeavour.[15]

2). Team Software Process (TSP): Where PSP focuses on the individual, TSP focuses on teams within an organisation consisting of 2-150 members, with additional versions available for even larger projects. TSP requires Team Building and Team Working with capable developers who've learned self-discipline having engaged in PSP. In Team Building, Launch is when members determine their project strategy, goals, roles, quality plans and risk analysis. These are all documented. Team Working involves activities like maintaining process discipline through effective leadership, communicating regularly about the status of the project and engaging in frequent reviews of quality (through unit testing and some of the aforementioned software metrics). TSP has been proven to be effective, with Boeing as an example, who saw a "94% reduction in system test time…and allowed Boeing to deliver a high-quality product ahead of schedule."(Watts H., 2000) As can be seen from the above diagram, PSP and TSP help organisations to more efficiently implement the Capability Maturity Model which provides benefits for the organisation as a whole.[16]

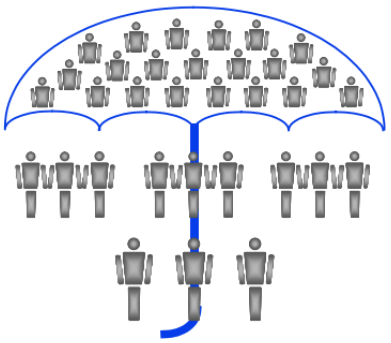


Figure 1: Process Improvement Methods

Source: Watts H., 2000 [16]

3). Hackystat and ISEMA: Hackystat is defined as "an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data."[17] Hackystat works by allowing developers the ability to add "sensors" for software to their work. These sensors collect unprocessed data, sending it to Hackystat SensorBase, a web service where it can be queried by other services to process and visualise the data in a meaningful way which would provide insights about the software development process.[17] Currently Hackystat is no longer in development, but it laid the groundworks for the In Process Software Engineering Measurement and Analysis (ISEMA) frameworks, a space where new systems are being developed and utilised by corporations, academics, researchers etc.[18]

4). ProM: ProM is shorthand for Process Mining. We know that practically all organisations follow various systems for implementing their respective and unique processes. There is much data, however, from these activities which can be analysed from a business' process execution logs to determine information relating to the business such as their performance, control flow and organisational perspective. ProM is free to download and is utilised through Java using plugins.[19]

A Note Covid-19's Impact on the Software Development Landscape – Analysing Data Example:

Following the extensive lockdowns and social distancing guidelines imposed by the world's nations to limit the devastating effects of the current pandemic, a majority of software engineers are working from home or taking a more flexible approach to in-person and at-home work. As managers have even less face-to-face contact with their teams, being able to effectively measure their team's performance despite the distance is critical. Before the lockdowns, projects that were handled in person accounted for 60% of all projects.[20]

Allstacks, for three months of the shift to Work from Home (WFH) analysed the data that they received from their users and uncovered the following insights:[20]

1. From March 2 to April 6, Allstacks noticed a drop in the number of bouncebacks as teams focused solely on current and new work. However, from May onwards, as these teams began to address the backlog, they had to revisit the older requirements and so the bouncebacks figures began to rise again.[20]
2. Unplanned work (which, for example, can mean one-off features) and planned work were variable for teams before the pandemic, however, since then, 40-50% of the current focus is on planned work. This comes as developers face less interruptions while working.[20]
3. Initially there were increases in product delivery slippage which is typically between 5-10 days. However, from the end of April, these values returned to their previous ranges, suggesting that companies are adapting to the changes and challenges that this new era of work from home brings.[20]

Part III: Algorithmic Approaches

Those of you familiar with Machine Learning (ML) may be aware of its relationship with artificial intelligence, the official definition being: "*Machine learning is a branch of artificial intelligence (AI) focused on building applications that learn from data and improve their accuracy over time without being programmed to do so.*"[21]

There are different categories of Machine Learning from supervised and unsupervised methods to deep learning some of which I've explained below:

Supervised Machine Learning: Typically used for regression and classification problems, supervised machine learning attempts to learn the best way to map an input variable to an output variable using an algorithm. Once the algorithm is able to predict new output variables with an acceptable level of accuracy, the learning is considered complete.[22]

Unsupervised Machine Learning: Some algorithms used by this type of ML include k-means for clustering problems or the likes of the Apriori algorithm for association rule learning tasks. Association rule learning is when researchers try to uncover some rules which would accurately describe large components of the data that they have, i.e. if people with X conditions also have a higher risk for Y condition. Clustering tries to uncover inherent groups or clusters in the data. What's distinct to unsupervised ML is that there are no pre-existing output variables for the input variables, the algorithm has to try and discover the underlying structure of the data by itself.[22]

Semi-supervised Machine Learning: This type of ML occurs when there are lots of input data, but not all of the output data is available or labelled. The majority of ML problems in the real world come under this category. An example would be if there were pictures labelled as either cars, pedestrians or trucks but some other images didn't have these labels meaning the algorithm has to be able to guess accurately what the unlabelled items were based on the inputs it's already come across. This technique can be used for predictions and discovering the structure of input variables.[22]

Reinforcement Machine Learning: The algorithm used in reinforcement machine learning doesn't make use of sample data to train itself, rather it works by trial and error. An example of this method in action is the 2011 IBM Watson System that was able to decide for itself whether it wanted to answer a question, the square it wanted to select and the amount it wanted to wager in a game of *Jeopardy!* [21]

Deep Learning: These types of algorithms try to comprehend the way that the human brain learns by using artificial neural networks. Neural networks consist of many algorithms. Examples of these models include Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Uses for these include

computer vision, speech recognition systems and self-driving cars. Deep learning requires the gathering and processing of large amounts of data, passing through multiple levels of calculations which are weighted and biased in order to try and continually improve the algorithm's results.[21]

Computational Intelligence:
We've seen above how ML is a subset of AI and also how Deep Learning is one of the uses for ML. Another notable subset of AI is Computational Intelligence (CI) which consists of three main pillars so to speak. These are Neural Networks, Fuzzy Systems and Evolutionary Computation. CI is a field that is continuously evolving and, relatively speaking, still in its early stages. It's a term that encompasses the "theory, design, application and development of linguistically motivated computational paradigms" (*IEEE*). CI is being used to create and develop new intelligent systems from games to cognitive developmental systems. Deep Learning has major uses in creating virtual assistants, personalisation of services (medical, entertainment etc) and also for developing self-driving vehicles. It takes no leaps of the imagination to picture what computational intelligence and advanced, complex systems may be capable of in the future especially when it comes to automating human tasks that have typically required more thought and creativity.[23]

Neural Networks: As stated under Deep Learning, neural networks are used to mimic how the human brain works by developing groups of algorithms that are able to learn by repeating tasks and correcting errors as they go.[23]

Fuzzy Systems: These have a more linguistic focus, based on how humans use language (especially the imprecisions in linguistics) to solve problems where traditional logic is generalised so that we can perform approximate reasoning.[23]

Evolutionary Computation: As the name suggests, this approach derives inspiration from biology and the evolutionary process that takes place. This technique is used to solve optimisation problems where a population of possible solutions are developed, evaluated and then modified as needed.[23]

Examples of Machine Learning Algorithms to Measure Software Engineering

Study 1: Measuring software development productivity: a machine learning approach
This paper presented in 2018 at the International Conference on Computer Aided Verification by J. Helie, I. Wright & A. Ziegler, seeks to automate the process of software development productivity measurement through the use of machine learning. The study takes a particular focus on measuring the outputs of software engineers by analysing their commits (specifically the time between commits) and source their data from the lgtm.com database, a very large dataset, which at the time, had over 10 million commits by 300 thousand developers. They considered two aspects of output: quantity of code changes and quality of the source code, using a Neural Hidden Markov Model and deep mixed density network (MDN) for measuring coding time and determining the best regression model to use. The neural Hidden Markov model was trained by the researchers to predict the probability of a developer coding at present for 1 minute (initial state) of their recent history and then used this to approximate the expected coding time between two commits. After aggregating all of this data, the researchers used the MDN to find a relationship between the code change and the actual coding time in order to attempt and quantify the total amount of code. For measuring the code quality, the researchers considered run-time bugs and technical debt and thus used alerts in source code to flag these before using a group of non-linear regression models to develop rankings for code quality.[24]

Neural Hidden Markov Model:
A hidden Markov Model is a probabilistic way to model a system which is assumed to be a Markov process with "hidden states" and which aren't possible to observe. Based on what is observable, information about this internal state can be determined. The models include the probability distribution of the initial state, the transition probability from one state to another and the likelihoods of an observation given an individual initial state.[25]

In the above study, a commit was assumed to only happen when coding (with a trained probability C), the two states are coding and non-coding with developers who are currently coding, ending their run in the next minute (this process has probability E(t)) and non-coding developers starting in the same period have a probability S(t). The probability values were obtained from a simple neural network which had five inputs.[24]

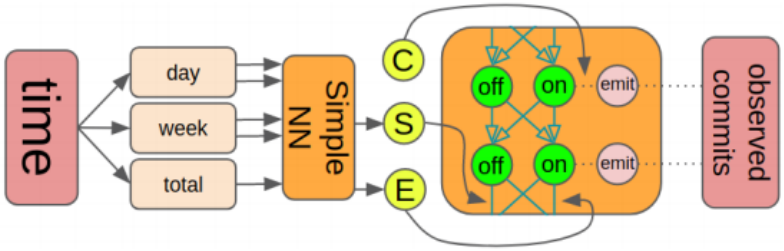
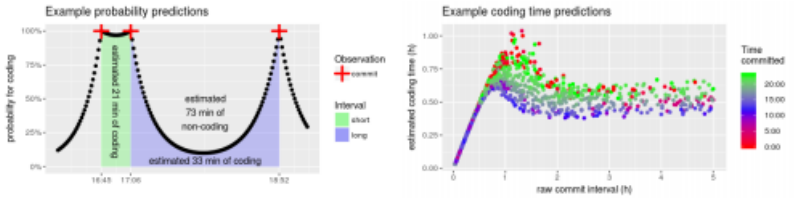


Figure 1: The NN computes time dependent transition probabilities S and E for the HMM. The HMM's state probabilities are fitted to the observed sequence of commits with the forward-backward algorithm. The NN and emission probability C are trained to maximize the likelihood of the sequence.

Figures' Source: J. Helie, I. Wright & A. Ziegler 2018 [24]



(a) The HMM assigns a coding probability to each minute. Short commit intervals often indicate uninterrupted coding, whereas longer intervals suggest coding interruptions. We estimate time spent coding as the expectation value, which can be computed by taking the area under the probability curve.

(b) Short intervals (e.g., < 1 hour) are often spent coding nonstop. So, counter-intuitively, they provide evidence for more coding time than medium intervals (e.g., 2 - 5 hours), which probably include coding breaks. This model picks up that the developer has fewer interruptions in the early morning.

Figure 2: Using the neural HMM to estimate coding times from commit intervals.

The above study used machine learning algorithms found in deep learning and semi-supervised methods.

Study 2: Software Development Effort Estimation Using Ensemble Machine Learning

This 2017 study dealt with a classification problem, using the K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) ML techniques, both individually and combined on the Desharnais and Maxwell public datasets to determine which algorithm(s) performed best. The focus of the study was to predict the effort required for software development projects having studied previous samples. The dependent attribute was the actual effort and the values related to this effort were the independent attributes. The Desharnais dataset had 81 projects to analyse while Maxwell had 62 available projects all ranging from the late 1980s to the early 90s. Both datasets were then analysed with each of the algorithms which are briefly explained:[26]

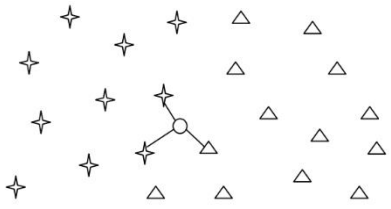


Fig. 1 Calculating the distance between new sample and the others

1). K-NN: is a non-parametric machine learning algorithm used for classification where the distance for a particular point from other points is found before determining the k-nearest neighbour for the specific point. Following this, the effort was estimated.[26]

2). SVM: is a parametric ML method used in both classification and regression problems. To make the switch to a classification problem from regression, the authors used Discretisation. How SVM works is by using a hyper-plane to try and maximise the margin between the instances from two separate classes. The term support vector refers to the points close to the border.[26]

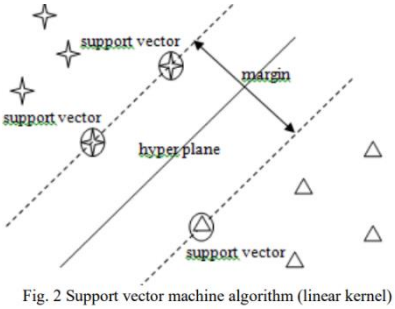
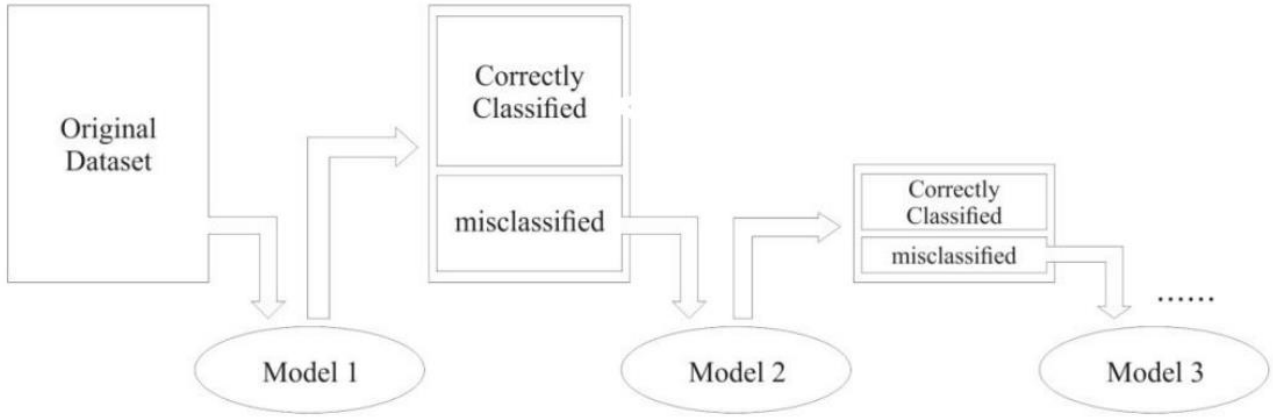


Fig. 2 Support vector machine algorithm (linear kernel)

3). Adaptive Boosting (AdaBoost): Boosting is a method, which is used in this study, to increase the accuracy of learning algorithms. This is done by creating an ensemble of models that have been fitted to the learning algorithm(s) which have a low error rate, thus improving their performance. AdaBoost is one such algorithm which works by sequentially building a strong model by amalgamating weaker classifiers which don't perform very well into one strong classifier. As can be seen in the diagram below, preceding models' mistakes are improved on by the succeeding ones.[26]



Figs 1-3 Source: Hidmi, Omar & Sakar, Betul 2017[26]

Fig. 3 How Adaptive Boosting Works

Naturally there are far more algorithms than the above and a multitude of studies that have been completed to try and solve different aspects of the software engineering measurement problems. As stated in the metrics section, many types of these metrics were manual and labour intensive, such as PSP. Since then, ML and CI advancements are helping to automate this process which will make these measurement systems and metrics far more efficient and useful to both managers and employees alike. Below is a diagram from a study found on IntechOpen.com which contains a list of different experiments within the previous decades using Data Mining and Machine Learning for Software Engineering Measurement.[27]

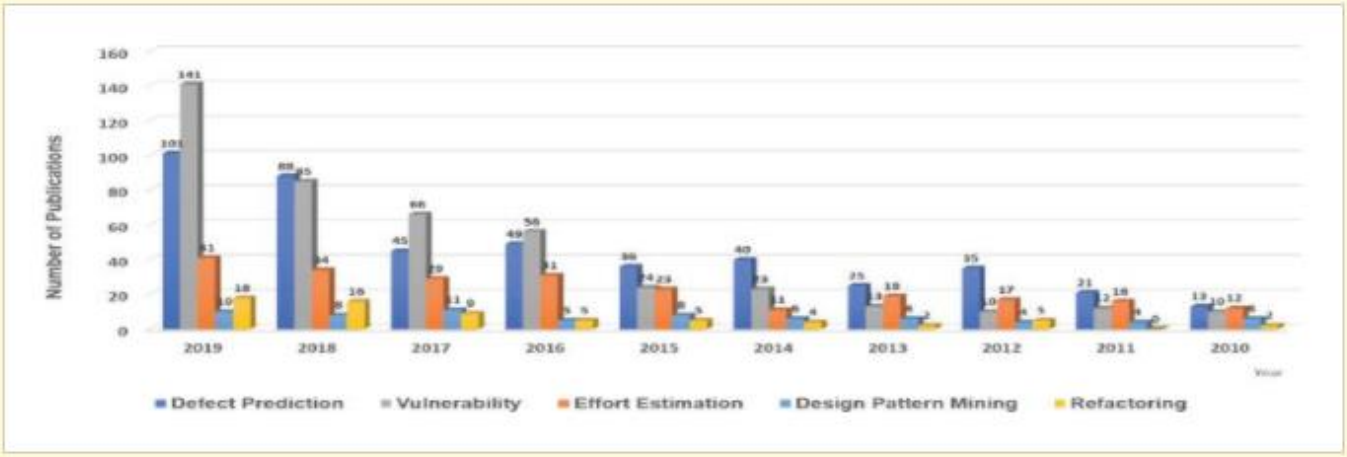


Figure 4.

Number of publications of data mining studies for SE tasks from Scopus search by their years.

Source Elife Ozturk Kiyak 2020[27]

Impact on the Future of Work?

We may look at the developments happening in the AI, ML and CI space with the idea that perhaps AI will replace the work of programmers as we know it. Could it be that one day, a machine will be able to code solutions to complex problems without any human intervention? The short answer is no, this is very unlikely to happen any time soon. Yes, we can automate certain tasks like gathering and visualising software metrics data, categorising code and measuring their quality as developers write them, translating different coding languages (Java to Python) or as many IDEs do, autocompleting code. Thus far this technology has been used to streamline the coding process, allowing developers to focus their attention on the more difficult and relevant aspects of coding solutions. We may think that machines can be capable of working like a human, but a machine is not a human. They learn from the data presented to them and it's up to clever and talented developers to decide what data to feed these algorithms, how to prepare it and how to measure whether or not the model created is accurate. What does seem likely to happen though, is the growth in AI-driven programming supporting human developers in their work.[28]

Part IV: Ethical Considerations

In the Age of Information, data is a precious resource and the implications of gathering and storing large amounts of personal data (especially its misuse and abuse) continues to be a major talking point and regularly makes headlines. In this section, I consider whether the act of convening these measurements and analyses can ever be fully ethical, and also some ways that it can be more regulated.

Measuring Software Engineering – Is it Ethical?

In a business context, there are many groups involved in the measurement and also the reporting process of measuring employee performance by gathering data. The ethical implications of this for both the evaluators and evaluatees and also for third parties who assess these measuring systems, are explored in the 2003 article: Ethical Dilemmas in Performance Measurement by Inge C. Kerssens-van Drongelen and Olaf A. M. Fisscher. Some of these actors are mentioned in the diagram below:[29]

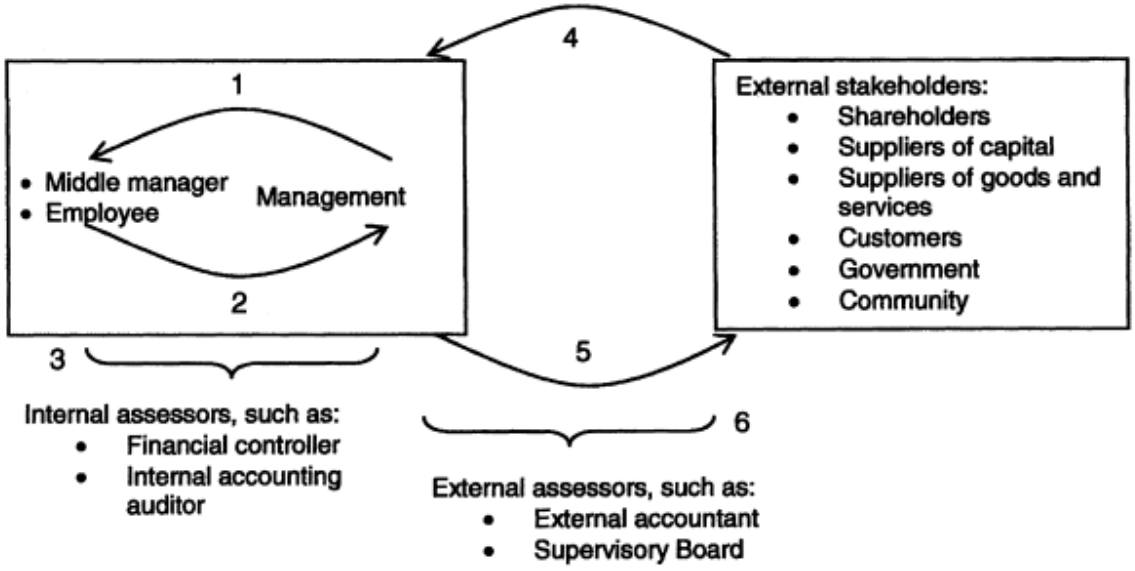


Figure 1. The different actors involved in the measurement and reporting process (Based on: Kerssens-van Drongelen and Fisscher, 2002).

Source: Drongelen, Inge C. Kerssens-van, & Olaf A. M.2003[29]

Evaluators:
Humans are not always as impartial as they would hope to be. By having a structure where managers are at the top of the hierarchy with both the power and responsibility to implement software metrics programmes as well as the consequences for different results (whether it's promotions, warnings, bonuses etc.), this power can be either properly used or abused, especially if certain managers have subtle biases. Some evaluators may even use these metrics to achieve short term targets and gains which would boost their own accomplishments. However, this could be at the expense of the long-term wellbeing and performance of employees and other stakeholders, especially for the company's overall stability. Evaluators may not store the data they gather on employees in a safe and secure way. It's not difficult for data breaches or carelessness to cause the exposure of people's sensitive information. As mentioned above, this type of data is incredibly valuable and may be treated nefariously by those with ill intentions.[29]

Evaluatees:
It is the evaluatee who provides the evaluators with the information that will be measured. Some employees driven by personal gain may seek ways to game the system either by tricking the tools that gather data or by sabotaging other people. This can be particularly pertinent if good performance under these metrics yield major rewards. Another consideration to make is that using these metrics for handing out rewards and punishments may be more of a hindrance than a motivating factor. If an individual is already motivated and passionate enough about their career, they will not need to be encouraged by fear or financial rewards. I can't imagine many people would like to be quantified or even reduced to certain metrics, possibly finding this restrictive to their work and progression. Humans' movements and behaviours are certainly predictable, but we are far more complex than we give ourselves credit for. It's near impossible to capture the full picture of an individual's capabilities through empirical assessments.[29]

Assessors and Stakeholders:
Third parties, whether they're assessors of metrics or also an evaluator, may not always act in the best interest of the business. If a fault or an inconsistency with the measurement process is discovered, it isn't unlikely that these assessors may struggle with the moral dilemma of deciding whether to report the incident or not. If a system is not up to standard, not fully developed and is also prone to much error, these should be flagged immediately. When devising metrics or considering what is efficient measurement, assessors and stakeholders must consider what is relevant to a specific goal. For example, if you want to determine whether a software engineer produces code of very high quality, measuring this by LOC would be futile. In addition to this, LOC may not be the best measure of productivity, especially if it encourages developers to write longer and less efficient code.[29]

As with anything that has to be measured while also involving systems and hierarchies, there are plenty of ways that it can go wrong, whether by appointing poor leadership, people with poor morals or implementing poorly structured systems. This can be a particular challenge when trying to make software engineering metrics as ethical as possible. In spite of this, I think there are benefits to software measurement, that it can be done in a way to minimise the impact of unethical behaviour and to make it a more morally acceptable process in general. It won't ever be perfect, in my opinion but the worst can be mitigated as outlined below:

Ways to Encourage Ethical Software Measurement:

- Follow a code of conduct that has been drawn up following relevant consultation with experts who would be able to devise a moral set of guidelines that would map the firm's actions and responses to various situations.
- Involve employees and those who are going to be measured in the process of drawing up different metrics. It's important to get the insights from evaluatees who know best how they do their job and what works or doesn't work for them in terms of measurement.

Source: Drongelen, Inge C. Kerssens-van, & Olaf A. M.2003[29]

- Any targets that are set by the stakeholders and managers for the business must be realistic. Once again, these must be based on the results of previous metrics and past performance, the opinions of those being evaluated and an obtainable reality.
- Establish and implement systems to protect sensitive data by restricting its access to authorised personnel and storing it securely.
- Ensure all data collected is GDPR compliant and held for as long as needed, but immediately destroyed when no longer relevant.
- See if there are ways to anonymise the data collected in such a way that, bar a select few people, no one would be able to trace source code, performance, commit/pull history to any one individual (perhaps by creating a complex system with private and specific numerical ID's for evaluatees).
- Be transparent about the measurement process at all times and offer timely and open feedback to all evaluatees.

Source: Drongelen, Inge C. Kerssens-van, & Olaf A. M.2003[29]

Part V: Conclusion

The measurement of software engineering is not a wholly new concept and has been in development for many years. As it is built and improved upon, it undergoes a continuous evolution that carries on to this day. A variety of metrics have come and gone, with new ones being devised, tested and studied for accuracy, efficiency and usability. It's important to note that no one metric or set of metrics will ever fully capture the whole picture and great thought, along with adequate research, is required to choose the most optimal set. As gathering and processing this data can be laborious and time consuming, there has been immense interest in the use of Machine Learning Algorithms and Computational Intelligence to automate these processes and use it to predict different performance metrics. A wealth of studies exist and are being devised to experiment with a multitude of data mining and machine learning algorithms with continuous improvements being seen regularly. Considering that these techniques have acquired advanced capabilities, it may be a worry for some who fear the takeover of developer jobs. However, this is unlikely to happen and rather algorithm driven learning will allow developers to be far more productive and efficient as they are released from the shackles of labouring over mundane, repetitive coding tasks. Finally, the ethics of these metrics need to be considered. While I personally wouldn't be comfortable with giving up so much of this confidential information which will later be used to make judgements about my abilities, I recognise its benefits and even the necessity of some of these metrics. I believe there are many things that can be done to make this process as ethical and secure as possible, which will naturally be developed over time so that even those of us who are sceptical about them will feel comfortable with these measurements.

References in Order of Appearance

[1] 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

[2] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

[3] Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[4] A. Altvater, “What Are Software Metrics and How Can You Track Them?”, Stackify.com, 16th September 2017, https://stackify.com/track-software-metrics/

[5] “Software Measurement Metrics”, tutorialspoint.com, https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm#:~:text=Software%20metrics%20is%20a%20standard,process%20metrics%2C%20and%20project%20metrics

[6] E. Heaslip, “Best KPIs to Measure Performance Success of Software Developers”, indexcode.io, 13th January 2020, https://www.indexcode.io/post/best-kpis-to-measure-performance-success-of-software-developers

[7] A. Circei, “Goodbye GitPrime, welcome Pluralsight Flow to the Git Analytics market”, medium.com, 15th January 2020 https://medium.com/billme/goodbye-gitprime-welcome-pluralsight-flow-to-the-git-analytics-market-ecee1c07d6d0

[8] Flow Metrics, pluralsight.com: https://help.pluralsight.com/help/metrics

[9] “Pluralsight Flow Expands Visibility Into Engineering Workflows With New Delivery Module Tool”, GlobeNewswire, 14th October 2020: https://www.globenewswire.com/news-release/2020/10/14/2108571/0/en/Pluralsight-Flow-Expands-Visibility-Into-Engineering-Workflows-With-New-Delivery-Module-Tool.html

[10] CodeClimate.com https://codeclimate.com/

[11] SonarQube: https://www.sonarqube.org/

[12] Allstacks: https://www.allstacks.com/

[13] “Allstacks' Predictive Forecasting for Software Development Now SOC 2 Type II Certified”, PR Newswire, 3rd June 2020: https://www.prnewswire.com/news-releases/allstacks-predictive-forecasting-for-software-development-now-soc-2-type-ii-certified-301069784.html

[14] Angel.co: https://angel.co/company/allstacks

[15] R. Sharma, “Personal Software Process”, GeeksforGeeks, 28th May 2020: https://www.geeksforgeeks.org/personal-software-process-psp/

[16] Humphrey. Watts, "The Team Software Process (TSP)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2000-TR-023, 2000.
http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5287

[17] Hackystat:
https://hackystat.github.io/

[18] P. M. Johnson, "Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System," First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, 2007, pp. 81-90.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4343735&isnumber=4343709

[19] ProM Tools:
https://www.promtools.org/doku.php

[20] H. Tapadia, "People, Process, and Outcomes: COVID-19 and the State of Software Development", Allstacks, 2020:
https://www.allstacks.com/covid-report-state-of-software-development#outcomes

[21] "Machine Learning", IBM Cloud Education, 15th July 2020:
https://www.ibm.com/cloud/learn/machine-learning

[22] J. Brownlee, "Supervised and Unsupervised Machine Learning Algorithms", 16th March 2016- Updated 20th August 2020:
https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[23] "What is Computational Intelligence?", IEEE Computational Intelligence Society, nd:
https://cis.ieee.org/about/what-is-ci

[24] Hélie, Jean, I. Wright and A. Ziegler. "Measuring software development productivity : a machine learning approach." (2018).
https://semmle.com/assets/papers/measuring-software-development.pdf

[25]J. Hui, "Machine Learning – Hidden Markov Model (HMM)", Medium, 9th August 2019
https://jonathan-hui.medium.com/machine-learning-hidden-markov-model-hmm-31660d217a61

[26] Hidmi, Omar & Sakar, Betul. (2017). Software Development Effort Estimation Using Ensemble Machine Learning. International Journal of Computing, Communication and Instrumentation Engineering (IJCCIE) ISSN 2349-1477. 4. 143.
https://www.researchgate.net/publication/318126628_Software_Development_Effort_Estimation_Using_Ensemble_Machine_Learning

[27]Elife Ozturk Kiyak (March 5th, 2020). Data Mining and Machine Learning for Software Engineering [Online First], IntechOpen, DOI: 10.5772/intechopen.91448.
https://www.intechopen.com/online-first/data-mining-and-machine-learning-for-software-engineering

[28] S. Cocking, "Artificial Intelligence: Will Robots Replace Coding Jobs?", Irish Tech News, 23rd September 2020:
https://irishtechnews.ie/artificial-intelligence-will-robots-replace-jobs/

[29] Drongelen, Inge C. Kerssens-van, and Olaf A. M. Fisscher. "Ethical Dilemmas in Performance Measurement." Journal of Business Ethics, vol. 45, no. 1/2, 2003, pp. 51–63. JSTOR,
http://elib.tcd.ie/login?url=https://www.jstor.org/stable/25075055