The Measurement of Software Engineering

Lauren Duffy – 15328891

The measurement of the software engineering process has been a prolonged topic of debate, one which has baffled experts for decades and has steered many to offer varying kinds of metrics to be used in an attempt to solve the issue. Despite extensive research in the field, software metrics continue to elude managers today. The software engineering process is comprised of a number of ordered steps from planning the software to purveying the final product.

While the classification of steps varies depending on the process model used, all models generally follow a similar structure for building software. These consist of; Requirements Specification; System and Software Design; Development and Testing; Implementation; and Operations and Maintenance. Throughout this process, the progress is to be tracked, measured and later assessed to determine the efficiency and effectiveness. This requires data to be derived and analysed in order to produce meaningful results. Many experts have offered their supposed solutions to measuring this performance. In my research, I have found that previous proposals, while many are now considered vestiges, have paved the way for a modern direction that combines many approaches.

**Difficulties in measuring performance**

The software engineering process generally varies for every project with unique tasks and setbacks, making it a difficult and tedious process to measure. We can use various data from the process to indicate the level of performance of the engineers. Moreover, there is even software available to track and measure this data. The problem lies in the question of whether we are processing relevant data and thus producing meaningful results. The difficulty in this problem is that we need to produce quantifiable data and results from a complex and qualitative process.

**History of measurement**

While the traditional approach taken has involved extracting quantifiable data, today this type of data is deemed irrelevant. These measures include number of lines of code produced within a time period, bugs eradicated and hours worked. This is gaining increased agreement that such measures do not guarantee efficient and effective work and thus is considered inadequate for indicating the level of performance. It has raised the question, is length of code an indication of good software? In most cases this is a completely irrelevant metric. It cannot be proven that this type of metric is an accurate measure of good software. In fact, there is increasing argument that this measure is completely nonsensical and can actually open a window for errors. Engineers may rush their work in order to receive higher praise. In addition to this, it can lead to inefficiencies and slower working code.

While this kind of data is undoubtedly quantifiable, it fails to provide us with any meaningful information on the level of performance of the process. Furthermore, these metrics do not guarantee a successful or quality product. It is a simplistic approach to a broad and complex field of work. We need to produce measurable data in quantifiable terms from a complex and qualitative process.

**Suggested Metrics and Limitations**

Various suggested metrics that have been put forward by many remain unsatisfactory. One suggested method proposes measuring impediments in replacement of the typical metrics used. This idea focuses on the perspective that previous measures are not only an inaccurate representation of what productivity is in software engineering but moreover, it hides the mistakes of the software that can potentially cause disastrous failures. The Ariane 5 was a heavy-lift launch vehicle which exploded shortly after launching due to a software error. This mistake cost approximately $370 million. Measuring the process of building the Ariane 5 in terms of lines of code under time restraints may have allowed a timely completion of a large project with seemingly quality results from tests. However, it is very clear to see that such a measure is completely inapt following its launch. Thus, the relevancy of measuring impediments in place of seemingly satisfactory work by programmers is unambiguous. (Barnes, xxxx).

While I agree with shifting the focus from traditional measures to more impactful measures, this idea fails to encompass the qualitative factors that contrive a successful product. Areas such as core values of the organization, a customer centric approach in building the product and employee satisfaction are some neglected factors.

A contrasting metric that focuses on areas where the previous position lacks attention is to measure employee satisfaction. This measure stems from the connection between job satisfaction and productivity, suggesting that and employee’s dissatisfaction associated with the job slows overall productivity by way of attrition. While this assertion is true, it is the case across all professions. Thus, it provides little contribution to the problem of finding an appropriate metric specifically for software engineering. The said problem is somewhat exclusive to this profession due to its complexity and the rate at which that complexity is evolving. Furthermore it is emphasized that with increased employee satisfaction comes increased productivity. Experts are placing the focal point on the more general qualitative aspect of performance rather than productivity. Productivity is quantitative whilst the process as a whole is qualitative. High productivity does not take into account the hidden errors leading to system failures that have collectively caused trillions of dollars in damage. (Fabulich, xxxx).

A more in depth approach to productivity metrics, is to measure the success and quality of the product. It begins with the view that managers should revert back to the goals and core values of the organization. The universal goal of all businesses is to successfully sell a product that is valued by the customer. Performance metrics are to stem from this idea and should focus on overall business outcomes. In a general sense, managers are to be persistent in instilling the goals of the entire business into the minds of the engineers, promoting a shared goal for the success of the product. These goals ensure that employees do not get disconnected with the objective. The focus is on delivering the best value to the customer by providing a quality product or service. This should be specified during the requirements stage and will establish a clear purpose of the product in the minds of the developers. Compliance with this will ensure that engineers do not give devoted attention to irrelevant new parts of a project. For example time and money is often wasted on developing new features to a product that is subsequently deemed undesirable by the customer. With this idea in mind, software engineers will contribute more effectively to the performance of the business.

In conjunction with this, more defined goals are to be put in place throughout the process. A number of outcome-based metrics that are aligned with the business’ goals can be used. Establishing to the employee, before the product is developed, the metrics that will be used in evaluating the success of the product as well as the engineering performance upon completion provides clarity to the employee. Other such metrics to be used include recording cycle times for projects and product scorecards. (Ziegler, xxxx) **more detail for product scorecards etc?**

**A New Modern Direction**

The fundamental question at hand is can we deliver definitive results from various metrics that will highlight the value of a software engineer’s work. While it is undoubtedly a complex field to research the validity of data, my view is that any process using inputs to produce outputs can be measured and assessed. The process of measurement could perhaps be as complex as the process of software engineering itself. However if we want to produce rational and reliable results of the measurement process we need to treat it as a complex task. In doing this, the software engineering process may even slow down in efficiency and speed, but the process of measurement will give the most accurate indication of performance quality.

As the process continues to gear towards quality and customer satisfaction a more detailed and restructured process model that includes the areas of quality planning and control and customer satisfaction analysis needs to be put in place. It is clear that while there are many arguments in favour of various metrics, every approach to date has faults. Perhaps this is due to this continuously growing complex realm that we are attempting to measure using individual simplistic methods. The reality is that, if we want to have the most effective software that is built in an efficient manner and is valued by the client, we need to measure every aspect of the process conjointly. As well as this we need to continuously restructure the software engineering process model to include new relevant features. (Kan, 2002).

As it currently stands, all software engineering models used include the most general steps which are software specification, software design and implementation, software verification and validation and software evolution. The importance of quality control and assurance should be emphasized during the process and perhaps should even be included in the model. As argued, one way to prove the success of the process is to show the quality of the product. Quality control can be implemented in every step of the software engineering process, closing the gaps of many software lifecycles. (El Gabry, 2017)

In theory, the project manager will gain a new role, becoming the analyst of quality and performance throughout the process. Ensuring quality can be most effectively covered in the software testing stage. This should include checking the requirements covered by test cases, noting the percentage of tests that have passes and the fluctuation rate over time. Defects can provide some useful observations for performance. The cycle time for fixing defects should be recorded from when the defect is found up to when it has been erased. Where defects are found, noting the most common areas and their resulting impact should also be studied. Recording this data and using it to produce means, ratios and correlations

will firstly, allow us to identify weak spots in the software and secondly, map the process to easily recognise the level of performance over time. Additionally, the time taken to complete small scale tasks is also a relevant measure can provide valuable data. The project manager should prioritse retrieving this kind of data in order to draw statistical measures. Results can be displayed on a dashboard supporting further study and analysis. Finally, customer experience is the last yet most relevant stage to record. Success of the product is determined greatly by the customer’s experience and this should be included in the measuring of performance.

**Computational platforms**

There is certainly no shortage of platforms or tools to track this data. The range of platforms available to analyse and map progress computationally is vast. Several code analysis tools provide automated testing services and are easily available. Examples of popular platforms are Coverity, AppScan, Vercode and HPE Fortify. Codacy is a software that automatically reviews code and identifies issues such as security vulnerabilities, code coverage, code duplication and code complexity. **say how code coverage reduces risk which is v important for avoiding failure/disasters**

Similarly, Sonarqube provides software to detect bugs, ensure code quality, as well as providing an in depth analysis of code and visualization of progress over time, enabling the user to see their performance graphed contrastingly using different metrics.

Furthermore, application performance monitoring software that offers a review of operations is also available. Differentiating from code analysis tools that focus on bugs, security issues and code quality, APM generates data from the software’s operations.

New Relic is just one application performance monitoring software offering digital intelligence tools analyzing data from personalized metrics, visibility dashboards showing analytic results as well as alerts set up for specified metrics. AppDynamics, Dynatrace, Compuware APM, BMC APM, CA APM and IBM APM all offer platforms that measure performance of the software.

These make it easier for managers to inspect and examine the progress of the team and easily retrieve specific data. On occasion, the project manager may need very specific data such as duplicated lines. Continuously racking of this kind of data means that it is easily available to managers at any time. The manager’s time is not wasted gathering, analyzing and searching for this data. It is efficiently generating new data daily. The manager’s job is therefore simplified greatly as they are only required to read the results and interpret their meaning. The availability of such platforms reduces time inefficiently spent by managers of gathering and mapping the data. This has allowed faster and more efficient development while simultaneously increasing the software’s quality. Building quality software at a minimum cost of design while upholding its stability is in essence the primary goal in software development.

**Algorithmic approaches**

These platforms use many different algorithms to construct and display the results. Test automation algorithms are used in all software development. It entails designing and implementing tests that assess every line of code to ensure risks or errors are found and eliminated prior to release of the product. Testing follows a structural format to be followed and on completion, the algorithmic approach taken is comparing expected outcomes with actual outcomes. From here, failures are detected when there is a mismatch.

Application performance monitoring analyses the operations of software using various algorithms. These are made up of four main algorithms: Anomaly detection, clustering, correlation and forecasting algorithms. Anomaly detection checks for unusual events within the software. This is extremely useful in detecting many types of faults. It operates using statistical calculations which produce various distributions. The resulting data is used to characterize the expected behavior of the program. Using this algorithm, code analysis tools generate precise outliers and deviations.

Clustering uses various pattern recognition algorithms. Its use in performance monitoring involves the conversion of repeated data to patterns of numbers. This approach enables an easy way to finding small faults among huge amounts of complicated source code.

Correlation algorithms use statistical methods to produce correlation values. In performance monitoring, a problem is detected when the correlation doesn’t match expected patterns.

Forecasting algorithms involves the use of machine learning to analyse trends and generate predictions. This enables the system from then preventing the problem from developing saving reaction time.

Codacy uses an algorithm called Code Pattern to analysis software code and identify impidements. The software allows the user to change the pattern of the algorithm to suite the needs of the software.

**Ethics**

There are ethical concerns associated with measuring the software process. These concerns are largely related to the individual engineer and focus on the issue of fairness. A common argument is the inability to measure the process definitively as a whole. The measurement is a complex matrix of metrics and often will not provide a clear conclusion. This highlights the point that the result can be inconclusive and subjective leading to harsh or unfair criticism. Furthermore, the measurement can lead to added pressures for the engineers themselves in addition to the stress already incurred from the work itself.

Often, in order to implement a strong process of developing software in an efficient and timely manner, software engineers are likely to work much longer hours leading up to the deadline in fair of being labeled as slowing down the process, particularly when said workers will be compared to others around them. It promotes unrealistic expectations and causes unethical treatment of the workers.

**References:**

1. Unknown Author. (2018). Tasktop Website [online] <https://www.tasktop.com/integrations>

2. Fenton, E. & Neil, M. (1999). Software metrics: successes, failures and new directions

3. Granbow, G. , Oberhauser, R. , Reichert, M. (Unknown Year). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology

# 4. Kan, S. (2002). Metrics and Models in Software Quality Engineering

# 5. El Gabry, O. (2017). Software Engineering — Software Process and Software Process Models (Part 2) [online] <https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc>

6. Unknown Author. (2007-2018).Tasktop: When Agile, DevOps and Lean Aren’t Enough: The Missing Imperative in Software Delivery <https://go.tasktop.com/rs/246-WDG-185/images/Tasktop-Whitepaper-Missing-Imperative.pdf>

7. Unknown Author. (2007-2018). TaskTop: Better Decisions, no more bottlenecks: 5 metrics for cross-lifecycle software Testing [online] <https://go.tasktop.com/rs/246-WDG-185/images/Tasktop-5%20Metrics%20for%20Cross-Lifecycle%20Software%20Testing.pdf>

8. Unknown Author. (2008-2018). Sonarqube [online] <https://www.sonarqube.org/features/clean-code/>

# 9. Padhy, N., Singh, R.P., Satapathy, S. C. (2017). Software reusability metrics estimation: Algorithms, models and optimization techniques

# 10. Rauf, A., Reddy, M. (2015). Software Test Automation: An Algorithm for Solving System Management Automation Problems