Measuring Software Engineering Report

Diego Molamphy | 18326635

Throughout the years, in the world of software engineering, organizations have tried and debated what they believed were the best metrics and methods used to measure the software engineering process. Measuring software engineering has always been a controversial topic. The methodology used to develop software is very closely related to how the process is measured. In the past, various waterfall methods were used to develop software. The waterfall model followed sequential order, in which a software development team could only move to the next phase of development once the current stage was completed. During this period, companies would have focused on measuring individual productivity in the development process by using metrics such as lines of code and function points. However, these metrics were inaccurate and unrepresentative of individual productivity. Productivity is measured by analyzing the input of a process and the subsequent output. To measure individual software productivity, we must measure the individual output of any given developer. However, it is impossible to truly measure individual software output and as a result, a developer’s productivity. There are far too many factors to definitively measure a developer’s productivity. Metrics such as line of code and function points are redundant when analyzing developer productivity. We know that the most efficient and functional code is short, concise, and clean. Measuring the number of hours put into project by a software developer is flawed as a productivity metric. Developers can solve the same problem in different ways. Many would argue that the best way to get a sense of individual productivity would be to analyze the return on investment and the customer satisfaction generated by a developer’s work. In more recent years, we have seen a shift in methodologies moving from waterfall to agile methods. The agile methodology supports the continuous development and refinement of software. This model requires more communication between different people involved in the software development process such as developers, testers, designers, and project managers amongst many others. As opposed to trying to measure individual productivity within a software development team, the agile method focuses on what can be measured to a reasonable degree of accuracy, overall code quality, testing quality and team productivity. In this report, we will consider the ways in these aspects of software engineering process can be measured along with computational platforms, algorithms and ethics involved in these aspects.

Coding is the basis of software engineering processes and software development. Before looking at other aspects of software development, it is extremely important that software engineering teams ensure the quality of their code is of a high standard. High quality code is extensible, maintainable, readable, well-documented, efficient, and well-tested (we will dive further into that part soon). Code quality is analyzed by considering the characteristics of the software. Characteristics analyzed include the size, complexity, quality, and performance of the code. Metrics to measure code quality can be broken down into two categories, dynamic metrics, and static metrics. Dynamic metrics are measurements taken when a program/piece of software is in execution. Dynamic metrics are used to measure the reliability and efficiency of the software. Static metrics are used to analyze how understandable, complex, and maintainable the software is. Some popular software product metrics include Fan-in/Fan-out, lines of code, cyclomatic complexity, length of identifiers and depth of conditional nesting. Fan-in/Fan-out measure number of non-leaf functions in a piece of a software. It is used to analyze the complexity of a piece of and how integral certain functions are to the software. As you might expect, ‘lines of code’ does what it says on the label. Generally, the longer the code is, the harder it is to understand and, as a result, it is more complex and prone to bugs. In the past, the lines of code metric was used to measure software engineer’s productivity although it became clear that this made no sense as lines of code has no bearing on a developer’s productivity. Cyclomatic complexity measures the control complexity of a program. It can also be used to determine the amount of test cases required to comprehensively test the code. The general formula for cyclomatic complexity is CYC = E – N + 2P where P is the number of disconnected parts of the codes flow (for example, a calling program and a subroutine), E is the number of edges (transfers of control), N is the number of nodes (groups of statements in sequence that have only one transfer of control) and CYC is the cyclomatic complexity of the code. Length of identifiers measures the average length of distinct identifiers in the code. Identifiers are names given to variables, functions, classes, packages ETC. Generally, longer identifiers result in more readable and understandable software. This highlights the importance of naming variables appropriately. Depth of conditional nesting is related to cyclomatic complexity. Cyclomatic complexity measures the absolute number of branches whereas depth of conditional nesting measures how deeply these branches are nested. Deeply nested statements are usually difficult to interpret and can lead to errors in the software. Traditionally, code quality would have been measured manually. Today, like many aspects of software engineering, the way code quality is measured has become more automated. Tools such as SonarQube continuously inspect and review code to detect bugs and improve code quality by offering feedback on things such as code complexity and code readability. SonarQube is compatible with twenty-seven programming languages and can be integrated with popular development environments such as Eclipse and Visual Studio. In addition to this, SonarQube can analyze repositories and its branches on version control platforms such as Bitbucket and GitHub, offering feedback directly to the software engineer in the pull requests. Other popular code quality control platforms include Cruicible, Review Board and KlocWork amongst many others. Code quality has always and will always have an important in the role in the software engineering process.

Arguably, one of the most important aspects of the software engineering is testing. Testing allows developers to find defects in their code and fix these issues, ideally but not always before reaching the user. It is important that testing is as comprehensive as possible. Comprehensive testing is achieved by fully testing the code with quality test cases. Test quality, also commonly known as product quality, is determined by analyzing how efficient and effective the test suite implemented is in detecting defects in software. In addition to this, test metrics analyze the type of defects occurring throughout the software’s development. By gathering valuable information on the defects and their impact, software engineering teams can improve their software development process to prevent defects, particular the severe ones, from happening in the future. There are hundreds upon hundreds of metrics available to software engineering teams to measure test quality. These metrics can be broken down into categories including test coverage, test tracking and efficiency, test effort, defect distribution, test execution, and regression. Test coverage metrics measure how much of the software has been appropriately tested. These metrics highlight areas of the software that may require more comprehensive testing. Test tracking and efficiency metrics measure how effective test are at discovering defects in the software, giving a sense of how the test suite is performing. Test effort measures basic characteristics of the testing framework such as the number of tests and defects per test hour. Defect Distribution metrics quantify which areas of the software are the most vulnerable to bugs and draws the developer’s attention to this area. Test execution metrics analyze surface level information a testing framework relays to the developers such as number of tests passed, and number of tests failed. Regression metrics measures the impact of defect removal and bug fixes on improving the clients experience with the software. Some popular test metrics include cost of poor quality, requirements and requirements coverage, defect open and close rate, mean time to detect (MTTD), mean time to resolve (MTTR), defect removal efficiency, testing and defect trends and defect severity. Cost of poor-quality measures the cost of having to fix and redevelop aspects of the software. The general formula for cost of poor-quality follows the format: Cost of poor quality = rework effort/ total effort x 100. Cost of poor quality in some cases is even broken down into two different metrics, cost of poor-quality pre-release and cost of poor-quality post-release. Cost of quality post release is almost always more expensive than cost of poor-quality pre-release as the software must essentially go into production again to fix a defect post release of the software. Requirements/Requirements coverage measures how many of the required tests have been implemented and what type of tests were written for each requirement needed in the testing. Defect open and close rate measures how long it takes to discover and fix a defect in the software. MTTD measures the average time taken for bugs to be detected in the software. Likewise, MTTR measures the average time taken for defects to be removed from the software. Defect removal efficiency measures the number of defects found during the software’s development and the amount of these defects that were eradicated from the software. Testing and defect trends are used to analyze the testing and defect removal process, looking for patterns that occur in this process to further improve the process for future projects. Defect severity analyzes the impact a defect has on the end user and on the company itself. The data provided by these metrics is extremely valuable to organizations for number of reasons. It allows for better planning and scheduling of projects, highlights defects that have a large impact on customer satisfaction and aids the improvement of testing in the future. Like we saw with code quality, the way in which we test software has become almost fully automated. With the exception of tests to measure the severity of defects and impact of defects (difficult to fully automate these tests), most testing is automated in the world of software engineering today. Tools such as Selenium, LambdaTest and Katalon Studio allow software developers to implement fully automated testing and get instant feedback on tests passed/failed, defects and inefficiencies in their code.

As we discussed earlier, Productivity is gauged by analyzing team productivity. Popular metrics today include cycle time, release cycle team, team velocity, throughput, lead time and sprint burndown. Cycle time measures the time taken to complete a certain task in the software engineering process, starting from when work was started on the task. Cycle time allows companies to estimate the time needed to deliver a new feature to the user. In addition to this, it is a good indicator of speed the software development team is working at for different types of tasks. Tasks that have had a high cycle time can be analyzed to find ways to improve its cycle time and planning can be improved to account for high cycle times of tasks of this nature. Similarly, lead time is the time taken to complete a task, starting from when the task arises or is created. It is important to note the subtle yet significant difference between lead time and cycle time. Lead time is another appropriate indicator of team performance and speed. Release cycle time analyzes the time taken from a when a client requests a new feature to the delivery of this feature. Like cycle time and lead time it is an important indicator of a team’s performance. Sprint burndown details the work completed throughout the sprint (short period of time when a team works to complete a set number of tasks) based on story points. Story points are a way of rating the relative effort of work in a format like that of Fibonacci. Story points are preferred over standard time estimations for a few reasons. Story points bring no emotional attachment in the way that dates do. They account for work that arises outside of the project such as emails and meetings. Story points allow for quick allocation work once each story point is given a value. Story points reward solving the task at hand rather than hours spent on solving the task. By using sprint burndown as a productivity metric, each sprint can be scheduled more appropriately and accurately so that sprint deadlines can be met right on time. Team velocity measures the amount of software development completed during a sprint (in story points). Team velocity gives organizations valuable information that allow them to scale future sprints and software development projects more accurately by analyzing challenges and issues that hampered the team’s velocity. Throughput measures a team’s output by the amount of work completed that adds value to the software’s development and ultimately, to the end user. Throughput is usually measured by the amount of tickets or tasks completed during a defined period and gives insights into how the team is dealing with the tickets. A drop in throughput may indicate that the team has come across a roadblock or are unable to cope with the current workload. Version control tools such as GitHub, Bitbucket, SVN and GitLab are particularly important in allowing companies to monitor the software engineering process and get the information needed to compute productivity metrics. By using these platforms, organizations can apply use the metrics to estimate the productivity of the team. Scrum tools such as monday.com, Scrumwise and Project Manager play an important role in analyzing productivity by tracking progress on tasks and sprints

In recent years, we have seen a further increase in the focus on monitoring productivity. Somewhat against agile methodology principles, companies have started to closely monitor the productivity of employees to analyze things such as how they are spending their time and their impact on team chemistry. Computer Tracking software’s such as Workpuls have become popular and common in the workplace, not only in the mega organizations, but also in companies of a smaller scale. In addition to this, some companies have given their employees tracking devices such as FitBits in the hope of increasing employee’s self-awareness of their productivity through monitoring things such as activity and sleep habits using these devices. Some of these tracking devices implemented can even monitor the tone of voice and can be used to analyze team dynamics and team chemistry. Naturally, this intense level of employee monitoring has risen some ethical and moral concerns within the working world, the argument being that some of these methods of tracking employee productivity are a blatant invasion of employee privacy. I believe it is too general to call employee productivity tracking ethical or unethical. However, I do think there are few key factors that determine how ethical or unethical this type of tracking is. Firstly, I think it is essential that companies are transparent with employees about the data they are collecting and how this data is being used. In my opinion, Tracking employee’s data without their consent and without them being well-informed on the matter is a complete invasion of privacy. To aid employee’s knowledge on how they are being tracked and how their data is being used, it is important that policies are set out. It is also vital that GDPR laws are always followed. Another important aspect to consider when tracking the productivity of employees to this degree is where and when is this information is gathered. Personally, I believe employees should only be tracked during working hours. Tracking employees outside of working hours is not only unnecessary most of the time but unethical. Employees should be allowed their right to a private life outside of work. Sure, employers can promote productivity through self-monitoring outside of the workplace by giving devices like FitBits, but the line is drawn here. In relation to monitoring conversations, I am somewhat divided whether it is ethical or not. Although it may be beneficial for organizations to monitor their employees’ tone of voice to analyze team chemistry and dynamics, in many ways I think it would be counter intuitive. Would the team dynamics, chemistry and the conversations taking place during working hours be the same if employees knew they were being monitored? Personally, I do not think so. I think monitoring employees to this degree would put them on edge and hurt the atmosphere in the work environment. Ultimately, in each case where employee productivity tracking is implemented, there are going to be cases in which it is implemented ethically and unethically to varying degrees and people will have their own perception on this ethical dilemma. Regardless, I think we will see more and more companies, from top to bottom, implement employee productivity tracking in the foreseeable future. In my opinion, for employee productivity tracking to be ethical, it is vital that we maintain the distinction between employees and other business assets. Although employees are assets, they are also human beings with a complex range of emotions and feelings.

In conclusion, measuring software engineering is not as clear cut as we would like it to be. In a perfect world, there would be a single metric to quantify the output of developers. However, this just simply is not the case. Why? It ultimately boils down to fact that software engineering is crafted, not manufactured. There is an element of creativity and variability in software engineering that cannot be quantified. However, as we have seen, we can measure important aspects of the software engineering process such as code quality, test quality, employee productivity (on a general scale) and team productivity. Who knows, in twenty years’ time there may be another Trinity student writing this exact report and the way we measure software engineering may well be completely different by then. Software engineering is always evolving and that is part of its beauty.

**Sources**

* Altvater, Alexandra, “What Are Software Metrics and How Can You Track Them?”, Stackify, 16/9/2017, <https://stackify.com/track-software-metrics/>.
* “Software Measurement Metrics”, tutorialspoint, <https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm>.
* Upadhyay, Rajkumar, “Product Metrics in Software Engineering”, GeeksforGeeks, <https://www.geeksforgeeks.org/product-metrics-in-software-engineering/>.
* Britton, Jill, “What is Cyclomatic Complexity?”, PERFORCE, 22/10/16, <https://www.perforce.com/blog/qac/what-cyclomatic-complexity>.
* Machado, Ignacio “Code quality, measuring and improving it”, devgurus, 27/9/18, <https://medium.com/devgurus/code-quality-measuring-and-improving-it-2137416d899a>.
* SonarQube, Wikipedia, 13/11/20, <https://en.wikipedia.org/wiki/SonarQube>.
* Ivanovs, Alex, “Top 11 Free and Paid Code Quality Tools for Web Developers 2020”, colorlib., 25/8/20, <https://colorlib.com/wp/code-quality-tools/>.
* “What Are Test Metrics?”, Sealights, <https://www.sealights.io/agile-testing/test-metrics/>.
* “Code Quality Metrics: Is Your Code Any Good?”, Sealights, <https://www.sealights.io/code-quality/code-quality-metrics-is-your-code-any-good/>.
* Zurawiecki, Marcin, “Why requirements test coverage is key to success”, Deviniti, 15/10/18, <https://deviniti.com/testflo/why-requirements-test-coverage-is-key-to-success/#:~:text=Requirements%20coverage%20is%20a%20useful,tests%20were%20written%20for%20them.&text=Requirements%20and%20tests%20are%20equipped,status%20to%20enable%20easier%20verification>.
* “MTTD vs MTTF vs MTBF vs MTTR”, Alertops, 7/5/18, <https://alertops.com/mttd-vs-mttf-vs-mtbf-vs-mttr/>.
* “TOP 10 SOFTWARE DEVELOPMENT METRICS TO MEASURE PRODUCTIVITY”, infopulse, 19/10/18, <https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/#:~:text=Software%20productivity%20can%20be%20defined,of%20outcomes%20from%20an%20activity>.
* Wagner, Stefan, Deissenboeck, Florian, “Defining Productivity in Software Engineering”, Springer Link, 8/5/19, <https://link.springer.com/chapter/10.1007/978-1-4842-4221-6_4>.
* Radigan, Dan, “Story points and estimation”, ATLASSIAN Agile Coach, <https://www.atlassian.com/agile/project-management/estimation>.
* “Product Release Cycle Time”, KPILibrary, <http://kpilibrary.com/kpis/product-release-cycle-time#:~:text=KPI%20%3A%20Product%20Release%20Cycle%20Time%20%7C%20KPI%20Library&text=The%20time%20between%20request%20and,service%2C%20required%20by%20a%20process>.
* Hu, Y., Zhang, J., Bai, X. et al. “Influence analysis of GitHub repositories.”, SpringerPlus, 2016, <https://doi.org/10.1186/s40064-016-2897-7>.
* Valijarevi, Kristina, “Ruthless Big Brother or Caring Supervisor: How Ethical Is Employee Monitoring Software?”, workpuls, 19/7/19, <https://www.workpuls.com/blog/employee-monitoring-software-ethics>.
* Fowler, Martin, “CannotMeasureProductivity”, martinFowler.com, 29/8/03, <https://martinfowler.com/bliki/CannotMeasureProductivity.html>.
* “Agile Vs Waterfall: Know the Difference Between Methodologies”, Guru99, <https://www.guru99.com/waterfall-vs-agile.html#:~:text=Waterfall%20is%20a%20Liner%20Sequential,in%20the%20software%20development%20process.&text=Agile%20allows%20changes%20in%20project,once%20the%20project%20development%20starts>.
* Aston, Ben “10 Of The Best Scrum Tools To Increase Your Team’s Productivity”, dpm, 1/10/20, <https://thedigitalprojectmanager.com/best-scrum-tools/>.