MEASURING SOFTWARE ENGINEERING

Ciara O'Sullivan

17321934

Computer Science and Business

CONTENTS

INTRODUCTION

Measuring the software engineering process is necessary for the overall coordination of an organization. It keeps current processes in line with goals and ensures productivity, quality and efficiency. In the past, many people have been quick to say software engineering is too complex, unknowable or unquantifiable to be measure. Martin Fowler believes it is a fool's errand, "*I can see why measuring productivity is so seductive. If we could do it, we could assess software much more easily and objectively than we can now. But false measures only make things worse. This is somewhere I think we have to admit to our ignorance."* However, in the last number of decades, the measurement of software engineering has risen in popularity. In this report, I will look at what characterizes software measurable data and what can be used to measure it. With an overview of the various computational platforms and algorithms used, I highlight the benefits this practice brings to the workplace. The choice of metrics and how they are measured is up to the organization. Finally, I will describe some of the ethical considerations, such as panopticon and accuracy, that must be considered when making this decision.

MEASURABLE DATA

There is no universal standard for measuring software engineering. The idea rose to prominence over the past few decades and is now accepted as part of mainstream software engineering (Fenton N. E. and Martin N., 1999). Software metrics are important as they help to measure software performance, plan work items and measure productivity. Managers look at metrics relating to four functions; planning, organization, control and improvement (Stackify, 2017).

The primary goal of using software metrics is to obtain objective and quantifiable measurements. These can be used for scheduling and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments

Other goals of software metrics (Hackernoon.com, 2019):

- Increase return on investment
- Manage workloads and priorities between teams and team members
- Reduce costs
- Determine the quality and productivity of the current software delivery process, and identify areas of improvement
- Identify areas for improvement
- Reduce overtime
- Improve overall quality
- Predict quality of projects once development is complete

Software metrics have many benefits. They help to identify, prioritize, track and communicate issues. This allows for better team productivity, effective management, assessment and prioritization of problems. Detecting problems sooner makes it easier and less expensive to troubleshoot.

Unfortunately, there are also many issues with measuring software metrics. The key issue is that there is no standard definition, there are multiple ways to measure these characteristics. An example of this is the LOC method. There are two common ways to measure LOC; count each physical line ending with a return statement or count each logical statement. It is hard to know what should be counted without a standard definition, and even more difficult to compare. Another issue is that in the past, many software engineers have seen measuring metrics as trivial and less important than doing the actual work.

There are two main difficulties when it comes to collecting measurable data. Firstly, collecting metrics takes a lot of time. This is an issue as software projects are often late and there is little time to be spent on activities that do not yield immediate benefits. Secondly, manual data collection is unreliable. Possible errors or missing data affect the analysis process (A. Sillitti et al., 2003).

DECIDING ON A METRIC

There are many different types of metrics: agile process metrics, production analysis metrics, security metrics, source code metrics, productivity metrics, process metrics, product metrics, quality metrics, the list goes on and on. Before a metric is chosen, it is important to consider the different characteristics and decide on what is appropriate for the current process.

Metrics characteristics include (Stackify, 2017):

- Simple and computable
- Consistent and unambiguous/ objective
- Use consistent units of measurement
- Independent of programming languages
- Easy to calibrate and adaptable
- Easy and cost-effective to obtain
- Can be validated for accuracy and reliability
- Aid development of high-quality products

Software engineers must follow all the relevant guidelines for measurable data. Software metrics must be easily understandable so they can be compared. It is good practice to link metrics to goals. Presenting metrics as a target motivates developers to achieve these goals. It is more important to track trends, not numbers. When a target is met, it is can simply be declared as a success. Likewise, if the numbers aren't being met, then something must be done. While this is a good indicator of what needs extra attention, these targets don't give much actual information on how the metrics are trending. Analyzing trends helps managers see progress towards the target, providing the necessary insights to achieve it. Short measurement periods are more effective. While this creates a slight interruption, it gives the development team time to analyze their progress and change their process if something is not working. Teams must stop using software metrics that aren't working and don't lead to change. Some metrics have no value when it comes to indicating software quality or team workflow. These metrics must be eliminated.

WHAT CAN BE MEASURED

It is possible to measure almost anything, but it is not possible to pay attention to everything. In this section, I give a brief overview of the just a few of the many metrics available.

PRODUCTIVITY METRICS

Productivity metrics are tools used to assess the performance and efficiency of the current software development process.

Lines Of Code

LOC is a basic measurement for number estimation. It predicts the amount of effort required to develop a project and estimate the level of productivity, by counting how many lines of code were delivered. Generally, we measure productivity by size, for example, we measure our assignments by how many questions we must answer or how long the answers must be. In the following scenario, output quantity is the LOC and the period of time is the time taken to develop the project.

Productivity = output quantity/ period of time.

There are two types of LOC; physical and logical. Physical counts the total lines of code in a project. This includes comments and blank lines if the LOC consists of less than 25% blank lines. Logical LOC counts the number of executable statements. However, specific definitions are tied to specific computer languages. For example, logical LOC measure for C is the number of statement-terminating semi colons (Singh, 2019).

While this is one of the most common metrics to measure productivity, it is very difficult to generate reliable data for actual productivity and to make predictions. This is because some lines of code require more effort and logical thinking than others. For example, if statements require more thought than an output statement. An alternative and possibly more exact method is to count tokens instead of lines, or group lines of code together and only count the groups. Another big issue with LOC is the lack of a standard. There is room for debate as to whether comments should be counted. They require time and energy when documenting the code, but generally, coded lines still require more effort. Comments should only be counted if they add value and not simply stating /* this is the end of a while loop */. The ethical concern here is that programmers might add unnecessarily long comments just to look more productive. But if comments do not count, then documentation may suffer. Non-executable code, such as constant and variable declarations or function headers, is easy to write and requires little effort. But this again brings up the question as to what should be counted. Another question is should reused-code be counted in LOC? Initially, this required time and effort, but once it is coded and tested, it is a simple matter of copy and paste. Non-delivered code often requires a lot of effort through testing. This is not counted in LOC, yet arguably, requires the most effort. LOC doesn't consider the complexity, reliability, quality of the code or effort required for non-programming activities. Experienced programmers can write more efficient code which shortens their LOC, making them less productive under the LOC metric. High-level languages will have a lower LOC than low-level languages. Strongly typed languages, such as Pascal or Java, have a higher LOC.

LOC is a "success" because it is easy to do and can be visualized as we understand what a 100 LOC program looks like (Fenton N. E. and Martin N., 1999). It is both intuitive and ubiquitous. As it is widely used already, LOC allows for simple comparison between projects. Despite unreliability for individual projects, it gives reliable average results. Lengthy code can be avoided by organizing reviews. Cost estimation models, such as COCOMO, are based on LOC and show a close agreement between predicted and actual effort.

Code Churn

Code churn is used to measure the change volume between two versions of a project. It builds on LOC and is equal to the sum of added, modified and deleted lines. It enables engineers to gain an understanding into the evolution of their software systems. Measuring code churn allows managers to control the software development process, particularly quality. The code churn rate increases when there are unclear requirements, an indecisive stakeholder, a difficult problem, prototyping, under-engaged engineers, or polishing. Each of these scenarios requires additional code and commits, causing a growth in the code churn rate.

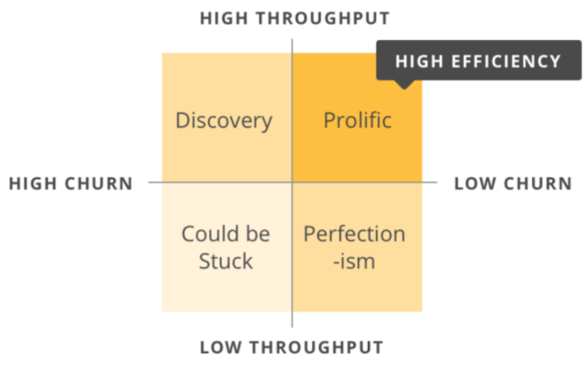
When measuring code churn, one must look at the code churn size, type of churn, breadth of churn and depth of churn. Code churn size is the number of lines changed to achieve the desired functionality. This shows new code, the existing code that was modified, and the size of the code measured by LOC. This also highlights what requires unit tests. The type of churn is used to understand the necessary tests to include. The breadth of code churn identifies the dependent functionalities and modules to be regressed. The depth of code churn helps testers to define what level tests to write. (Raj L., 2018).

Project Burndown

A burndown chart is a high-level visibility measurement tool for planning and monitoring progress (Taghi Javdani et al., 2012). The chart represents the amount of work still remining. There are two types of burndown, iteration burndown and release burndown. Iteration burndown estimates the remaining work that still needs to be completed in this iteration. Release burndown looks at the current release. Burndown is useful for comparing estimated work (ideal burndown) and remaining work. This information helps teams with making decisions to add or drop features if the project is ahead or behind schedule. An alternative to the burndown chart is a burn up chart. This focuses on incremental progress and work done instead of work remaining. This is because many projects have an unknown size.

Efficiency

Efficiency measures the proportion of an engineer's contributed code that is productive. This is calculated by comparing coding output with the code's longevity. The amount of code written is irrelevant, all that matters is if and how it works. A higher efficiency rate means that code is providing business value for longer. A high churn rate reduces efficiency. The most prolific engineers will make a lot of small commits, have a modest churn rate, and a high efficiency rate. Efficiency is important to measure because high efficiency results in the most high-quality, time and money effective projects.



Throughput

Throughput refers to the total value-added work output. Measuring throughput is beneficial to detect when the team is having difficulties as the throughput rate drops. Comparing the average throughput against the current workload helps in identifying when the team is overloaded. Throughput can be increased by redefining software quality based on: code integrity, customer and operation impact of defects, date of delivery, quality of communication, system ability to meet service levels, and cancelled defects that eliminated wasted Q&E time.

PROCESS METRICS

Process metrics are used to monitor, evaluate and improve operational performance.

Lead Time and Cycle Time

These two metrics are very alike but they measure different sections of the development process. Lead time is the time between the definition of a new feature and its availability to the user (Shkabura, O., 2018). Cycle time is the total time between the moment when the work starts until the project is completed and ready for delivery LeanKit, 2019). These metrics help to estimate when a project will be completed and how quickly new features an be developed and delivered. They help engineers understand how long work takes to flow through their value streams. They are beneficial for tracking trends which provides information to evaluate changes and forecast future work.

Tracking lead time helps determine the impact of changes that have been made. Do these changes help delver value faster or slow the process down? One can understand a developer's speed per task by tracking their cycle time. This is done by breaking down the total throughput into median time by status or issue type. This helps to pinpoint bottlenecks and set accurate expectations.

Commit Frequency or Active Days

A commit is an individual change to a file. Monitoring a developer's commit frequency helps in gathering data for their active days. An active day is any day that an engineer has contributed code to a project. Non-engineering tasks, such as meetings or planning, can impact the number of active days. They add up and often result in the loss of at least one active day. Calculating this metric uncovers the hidden costs of interruptions. Pushing code is one of the most central ways engineers add value to an organization. "Commit often, perfect later, publish once" shows the importance of committing. If an engineer fails to commit and then somehow how loses or breaks their code, they will have created a lot of trouble for themselves and their team (Anaxi, 2019).

WIP

Work in Progress involves tracking items that have been started but have not been not finished yet, therefor, not providing value to the customer. This helps to improve overall flow of value through the system. Work adds no value to the customer, team or organization unless it is finished.

A Kanban system with too much WIP could cause a bottleneck, ultimately delaying the delivery of value. - (LeanKit. 2019).

WIP can be represented in a cumulative flow diagram. WIP is often used for estimating the delivery date and lead time. This means it can be used to correct problems before they become too big and too expensive (Taghi Javdani et al., 2012).

QUALITY METRICS

Quality metrics are used to ensure quality in the overall software development process.

MTBF and MTTR

Both Mean Time Between Failures and Mean Time To Recover/ Repair measure how a project performs in the production environment. They aim to compute how well the software recovers and preserves data (Anaxi, 2019).

MTBF measures the of reliability of a system. When MTBF increases, there is an improvement in the quality of processes and expectedly, the final product. MTBF computes the average time between a failure arising and the next time it occurs. It can be calculated as follows:

MTBF = total time of correct operation in a period/ number of failures.

MTTR is the time taken to run a repair after a failure occurs. It measures how fast repairs can be deployed to consumers (Shkabura, O., 2018). It indicates the efficiency of a corrective action on a project. As developers gain a better understanding of issues and how to fix them, they become more efficient. This results in a smaller MTTR (Anaxi, 2019). MTTR can be calculated as follows:

MTTR = total hours of downtime caused by system failures/number of failures.

Code Coverage

Code/ Test coverage is a percentage of the lines of code that are executed while running the test suite. Generally, an engineer would aim for between 80-90% code coverage. 100% is almost unattainable. Code coverage is not necessarily a measure of quality. More so, it is an indicator of the process a team is using to achieve quality. Code coverage is an important metric to identify areas that are untested. A falling rate in code coverage means the team should devote more time to test driven development. Code coverage can be calculated as follows (Medium, 2017):

Code coverage = number of lines executed/ total number of lines of code.

PSP

The Personal Software Process is a methodology to monitor software development from the beginning of a project to the finished product. PSP can be used with any programming language or design methodology. The aim of this process is to produce zero-defect products on time within budget. It is particularly effective in helping engineers to achieve these objectives when used in conjunction with team software process (TSP). PSP helps engineers improve their planning and estimating skills, make commitments and schedules they can meet, reduce defects in projects, manage quality, run tests, write requirements, and define processes (w. B.,2018). The continuous monitoring encouraged by PSP helps developers track their actual performance and compare it to their planned performance. With PSP, developers can analyse specific elements, such as behaviours or interactions, that affect them.

As every software engineer is different, the PSP process is flexible, and plans for work can be based on their personal preferences. Engineers should use well-defined processes to improve their performance. They are personally responsible for the quality of the product they make. It is easier to prevent errors than finding and fixing them. It is also much cheaper to trace and fix defects early, before they become a major problem. The key purposes of PSP are to estimate variables such as time schedule or quality, of projects, to look for improvements in the development process and to highlight problems (A. Sillitti, et al., 2003).

COMPUTATIONAL PLATFORMS

In order to properly assess the software engineering process, it is not enough to merely collect data. The data must be collected, organized, analyzed and compared. There are many computational platforms to choose from, all with different benefits and areas of expertise. The following platforms are some of the most popular for measuring the metrics outlined above.

CODESCENE

CodeScene is a visualization tool that uses predictive analytics to uncover hidden risks and social patterns in code. It is easy to set up by simply logging in with a GitHub account. CodeScene analyses trends to uncover possible productivity bottlenecks, parts of the code that may be difficult to maintain, technical risks, areas for improvement regarding productivity and quality gain, and information about the knowledge distribution between team members (Codescene.Io, 2015). This platform enables developers to visualize their development process. By measuring code churn, it helps to predict post-release defects. When code churn increases, the code gets more unstable and more susceptible to defects. Code churn rates indicate to developers if they must stabilize the code closer to the delivery date.

CodeScene makes it easy to track trends by tasks. This allows developers to inspect the size and impact of individual tasks, and if they are on an appropriate level. The platform generates a commit activity chart which shows the number of commits and authors. This can be used to highlight productivity issues, such as an increase in authors but not an increase in commits, i.e. there are authors who are not committing enough to the project. CodeScene calculates an active contributors' trend to show the number of authors in a codebase. With this information, teams can evaluate the effect when a project is scaled up or down. CodeScene measures two churn metrics, the number of added lines of code, and the number of deleted lines. The reason for this is to uncover long-term trends in code churn rolling average.

HACKYSTAT

Hackystat is a framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data (Hackystat, 2019). This framework allows individuals to collect and analyze PSP data automatically. It is the third generation of PSP data collection tools developed at University of Hawaii. A major benefit of Hackystat is its support for the exploratory nature of telemetry-based decision making.

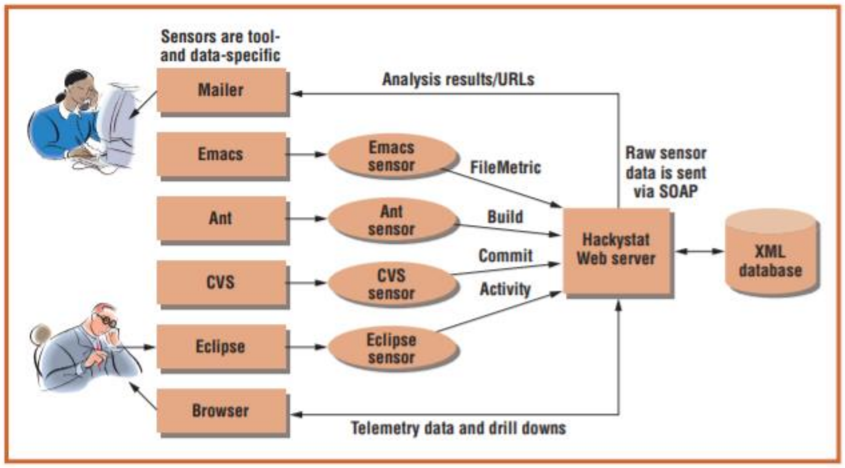
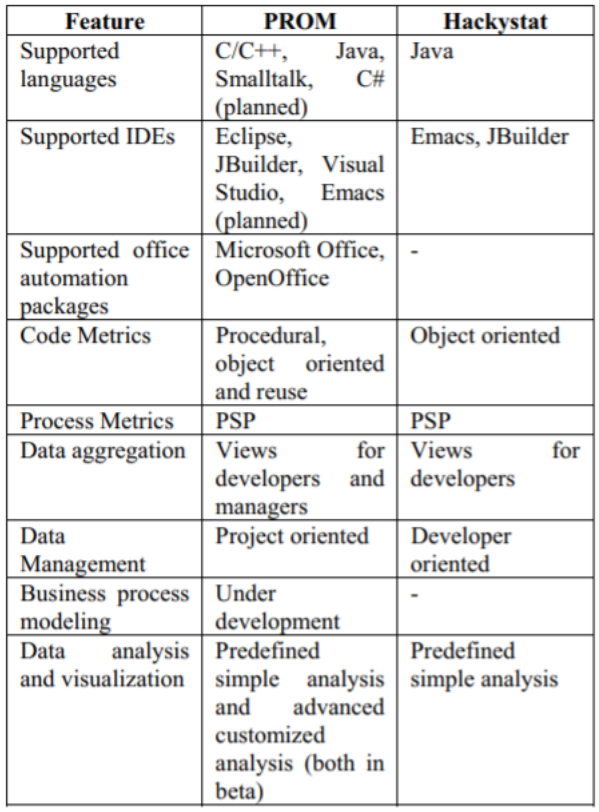


Figure 1. Hackystat's basic architecture. Sensors are attached to tools that developers directly invoke (such as Eclipse or Emacs) and tools that developers implicitly manipulate (such as Concurrent Versions System or an automated build process using Ant).

Hackystat sensors can be installed into various tools such as editors, build systems, and configuration management systems. These sensors collect data which is then sent to the Hackystat SensorBase web service where it is stored. This ensures a high level of privacy, which is important in business, but limits sharing data within group or an organization. The SensorBase repository can be queried by other web services to utilize the data, integrate data into a project, or generate various visualizations of the data. Engineers can also set alerts to be notified when specific conditions occur (A. Sillitti, et al., 2003).

Hackystat supports four types of telemetry: development, build, execution, and usage. Development telemetry gathers data by observing the project developers' and managers' behavior based on their tool usage. This data is information about the files they edit, time spent using tools, changes made to projects, and the sequences of tool or command invocations. It can be implemented in editors such as Eclipse, or management tools such as CVS or Jira. Development telemetry is useful for metrics such as code churn, lead time, and cycle time. Build telemetry gathers data by observing the results of tools invoked to compile, link, and test the system. It can be implemented in build tools such as Ant and testing tools such as Junit. Build telemetry is useful for code coverage. Execution telemetry gathers data by observing the system's behavior as it executes. It can be implemented in tools that test for load or stress such as JMeter. Usage telemetry gathers by observing the user's interaction with the system (Johnson, Philip M., et al., 2005).

Table 1: PROM – Hackystat comparison



Feature	PROM	Hackystat
Supported languages	C/C++, Java, Smalltalk, C# (planned)	Java
Supported IDEs	Eclipse, JBuilder, Visual Studio, Emacs (planned)	Emacs, JBuilder
Supported office automation packages	Microsoft Office, OpenOffice	-
Code Metrics	Procedural, object oriented and reuse	Object oriented
Process Metrics	PSP	PSP
Data aggregation	Views for developers and managers	Views for developers
Data Management	Project oriented	Developer oriented
Business process modeling	Under development	-
Data analysis and visualization	Predefined simple analysis and advanced customized analysis (both in beta)	Predefined simple analysis

PROM

PROM or PRO Metrics is an automated tool for collecting and analyzing software metrics and PSP data (A. Sillitti, et al., 2003). It acts as an automated data collection and analysis tool that collects both code and process measures. It covers a wide range of metrics including all PSP metrics, procedural and object-oriented metrics. It also looks at ad-hoc developement metrics to track non-coding activities such as writing requirements with a word processor. Data is collected at different levels of granularity; personal, workgroup, and enterprise. The architecture of PROM is extensible to support new IDEs, kind of data, and analysis tools. IDE dependent plug-ins need to be as simple as possible. Developers can work off-line. PROM has for main components; PROM database, PROM server, plug-ins server, plug-in.

GIT

Git is an open source version control system. It can handle all types of projects with speed and efficiency.

GITHUB

GitHub is a development platform that allows engineers to host and review code, manage projects, and build software (GitHub.com, 2019). It provides the functionality of Git. A code security feature allows developers to create protected branches that only they can access. Access control limits access to only those who need it with granular permissions and authentication. While the main function of GitHub is not to measure software engineering, it does provide many tools to aid this. GitHub records all commits by a developer. These can be visualized in a graph. It tallies the total LOC committed by a developer on each project. It also has a public API that allows for extracting data from public accounts. This data can be commits, contributors, and push and pull requests. This information can be used to analyze productivity metrics, such as commit frequency and active days.

GITPRIME

GitPrime provides the functionality of Git and more. This platform collects git data and generates it into easy to understand insights and reports, to help make. GitPrime creates and environment to debug development processes with objective data, identify bottlenecks and compare trends (GitPrime, 2019).

Some benefits of GitPrime are:

- To track velocity over time. This helps continuous improvement and ability to deliver by understanding software engineering trends.
- To manage pull requests. A visualization of commit and pull requests is generated to see work clearly and help reduce cycle times.
- To share knowledge in code reviews. Visualizations of work provides an insight into code review and teamwork dynamics

VELOCITY – CODECLIMATE

Velocity turns data from commits and pull requests into understandable data to help make improvements in productivity. The aim of Code Climate is to "shine a light on your software development process" (CodeClimate, 2019). A key feature of Velocity is the commit-to-deploy visibility which highlights bottlenecks hidden throughout the process. This platform benefits managers, engineers, executives and product leaders. Code Climate provides resources to analyse test coverage, code churn, cognitive complexity, cyclomatic complexity, duplication, and maintainability. Velocity integrates easily with popular tools such as GitHub, Git and Jira. Due to this integration it is easy to import repositories and manage pull requests. This in conjunction with a public API, enable managers to extract data similarly to that of GitHub. This information can be used to measure various metrics, such as active days or throughput.

ALGORITHMIC APPROACHES

As well as the various computational platforms, algorithms can be used to analyse data. The following algorithmic approaches perform calculations on measurable data, which managers can use for comparison.

FUNCTION POINT ANALYSIS

Function Point Analysis was developed by Allan Albrecht in the late 1970's at IBM and then was further developed by the International Function Point Users Group (Totalmetrics.Com, 2019). It is a method of Functional Size Measurement. FPA uses a standard metric for calculating the overall complexity and size of each application within a system.

FPA gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer. - ("Software Engineering | Functional Point (FP) Analysis - GeeksforGeeks")

FPA expresses the functionality an information system provides to a user. The algorithm design supports streamlining the counting process. Instead of reading each individual line of code, function points are used to measure business functions as individual units. FPA calculations are generally used to work out the cost (money or time) per unit. Five parameters, divided into two types, are used in these calculations. Data functions focus on two parameters; Logical Internal Files or External Interface Files. Transaction functions focus on the remaining three parameters; External Input, External Output, External Inquiry (Tutorialspoint.Com, 2019,).

The calculations and the categorization of data creates a standardized approach to measuring team productivity, risk, and system complexity. Analysing applications with FPA enables organizations to assess and compare each developer, team, and vendor throughout the development process. While this FPA generates valuable data, it is hard to do properly and is unnecessarily complex (Fenton, N. E., Martin, N., 1999).

CYCLOMATIC COMPLEXITY NUMBER METRIC

McCabe's cyclomatic number (McCabe, 1976) is very popular as it is easily computed by static analysis. McCabe's is often used for quality control (Fenton, N. E., Martin, N., 1999). It measures the complexity of a program by counting the number of linearly independent paths through the program. A higher cyclomatic number indicates a more complex code. This means the code will be more difficult to understand and have a higher probability of containing defects. The number also indicates the amount of test cases that must be written for all paths to be executed. Cyclomatic complexity can be calculated as follows (Chambers.Com, 2019):

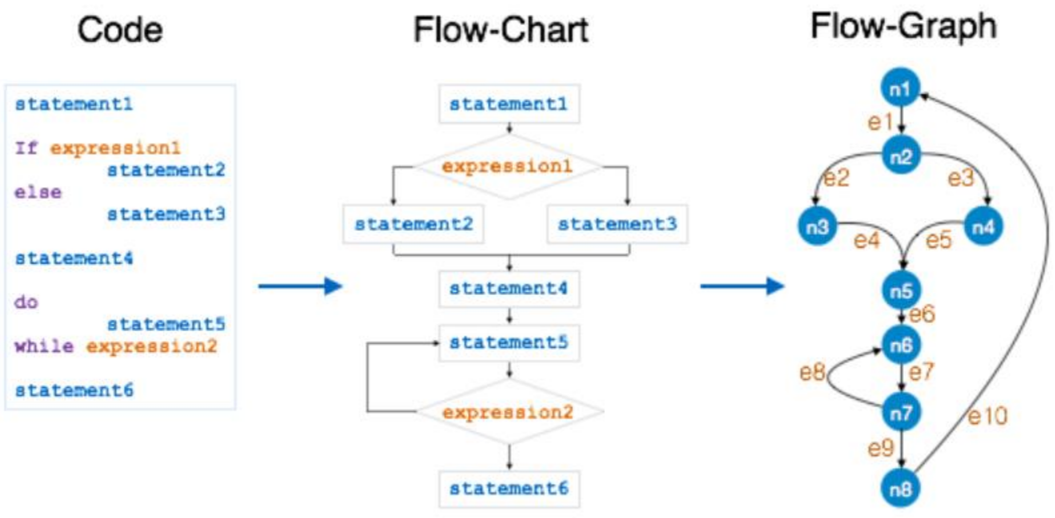
$$\text{Cyclomatic complexity (CC)} = E - N + 2P$$

Where:

P = number of disconnected parts of the flow graph
E = number of edges
N = number of nodes

This algorithm is beneficial as it highlights areas for improvement and possible risks for bugs. If a high cyclomatic complexity number is returned, developers should aim to test and simplify the code.

Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.
- Alan Perlis, American Scientist



HALSTEAD'S SOFTWARE METRICS

In 1977, Maurice Howard Halstead used metrics to measure software complexity directly from source code (Ibm.Com, 2018). The aim of Halstead's metrics is to count tokens and determine if they are operators or operands. This algorithm can be used for C/C++/Java source code. According to Halstead *"A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands".*

The basic measures are:

n1 = number of unique operators
n2 = number of unique operands
N1 = number of total occurrence of operators
N2 = number of total occurrence of operands

These are used to calculate the various metrics as follows:

Vocabulary (n): n1 + n2
Size (N): N1 + N2
Volume (V): N * log2 n
Difficulty (D): (n1 / 2) * (N2 / n2)
Effort (E): V * D
Errors (B): V / 3000
Testing Time (T): E/ k

k acts as the stroud number, given a value of 18. This number can be adjusted depending on test conditions, such as team background.

This algorithmic approach is helpful as an indicator if the development process is on track with forecasted goals. It builds on LOC but instead counts tokens, making it a more accurate representation of productivity. It also predicts the maintenance effort, rate of effort and quality of the project, unlike LOC. This algorithm doesn't require an in-depth analysis of the programming structure to achieve these results. However, this method does have its drawbacks. Differentiating between operands and operators can be a challenge for more complex codes. This method is also incapable of measure the structure of a code, inheritance, and interactions between modules (Ayman Madi et al., 2013).

ETHICS

Ethics are the moral principles that govern a person's behavior or the conducting of an activity. It is important for organizations to take the following ethical concerns into consideration before choosing a software metric.

PANOPTICON

The panopticon is a disciplinary concept introduced by Jeremy Bentham (ethics.org, 2017). It is described as the central observation tower within a circle of jail cells. The guard in the tower can see every cell but the inmates are unable to see into the tower. That way the inmates know that at any point in time the guard could be watching them, but they are unable to see him. This seemingly constant surveillance is enough to encourage good behavior within the prison for fear that the guard is watching.

Philosopher and psychologist, Shoshanna Zuboff, compared the role of PCs to an "information panopticon." Computers can constantly monitor the amount of work completed by developers. Employers can implement programs to track work done by employees and, therefore, measure their software engineering practices. They may also implement keystroke tracker. While this could be considered an effective metric as there would be endless amounts of measurable data, it is intrusive and lacks transparency. In this set-up, the computer represents the panopticon, while the developer is the prisoner. A developer's actions can always be monitored and analyzed, while they never see the analyzer. Keystroke trackers completely invade an employee's privacy. They may be storing personal information on their computers. With GDPR, this is not only unethical, it is in violation of the law.

DEMOTIVATION AND ACCURACY OF DATA

The constant measuring of software engineering could have detrimental results. With more pressure to meet shorter lead time or cycle time, developers may take short cuts. These short cuts risk delivering a product that was not fully tested and may possibly contain defects. This could have a terrible affect on the company's public image. Failure to test code extensively only to uncover a bug later, ends up costing more in both time and money.

If being measured on LOC, developers may be tempted to write lengthy, inefficient code to look more productive. They may also write irrelevant comments which could overshadow the necessary information. LOC doesn't consider the complexity, reliability, quality of the code or effort required for non-programming activities. As Bill Gates said, "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs." How is LOC to yield accurate results for productivity and effort without all these variables? If employees are not being rewarded for their extra effort, productivity in these areas may decline.

CONCLUSION

As can be seen in this report, almost anything can be measured. It is up to the organization to choose their metrics based on the measurable data that is most relevant to them, within their time and money constraints. It is good practice to invest in measuring a range of metrics, as thee are positives and negatives to all. While LOC is one of the most popular measurements of productivity, it ignores many of relevant variables that should be tested. LOC is more efficient is it is measured in addition to other metrics such as code churn. With the widespread computational platforms and algorithms available, it is easier than ever before to track the software development process. Software engineers and managers have chosen to follow the words of two scholar's, Galileo's "What is not measurable, make measurable," and Tom DeMarco's "You can neither predict nor control what you cannot measure" (Johnson, Philip M., et al., 2005). However, with the increasing popularity of these practices, there are increasing ethical concerns.

BIBLIOGRAPHY

1. 5 Developer Metrics Every Software Manager Should Care About. (2019). [Blog] *GitPrime Blog*. Available at: https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/ [Accessed 11 Nov. 2019].
2. Akhnoukh, N. (2015). *You CAN (And Should) Measure Software Engineering Performance*. [online] Engineering.kapost.com. Available at: https://engineering.kapost.com/2015/08/you-can-and-should-measure-software-engineering-performance/ [Accessed 9 Nov. 2019].
3. Anaxi. (2019). *Software Engineering Metrics: An Advanced Guide*. [online] Available at: https://anaxi.com/software-engineering-metrics-an-advanced-guide/ [Accessed 9 Nov. 2019].
4. Files.ifi.uzh.ch. (2019). [online] Available at: https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0203/Seminar_3.pdf [Accessed 9 Nov. 2019].
5. Humphrey, Watts. 2000. *The Personal Software Process (PSP)* (Technical Report CMU/SEI-2000-TR-022). Pittsburgh: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5283
6. Hackernoon.com. (2019). *7 Rules to Track Software Engineering Metrics Correctly*. [online] Available at: https://hackernoon.com/how-to-use-and-not-abuse-software-engineering-metrics-3i11530tr [Accessed 8 Nov. 2019].
7. Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157
8. Johnson, Philip M., et al. "Improving software development management through software project telemetry." IEEE software 22.4 (2005): 76-85.
9. Taghi Javdani , Hazura Zulzalil, Abd. Azim Abd. Ghani, Abubakar Md. Sultan, On the current measurement practices in agile software development, International Journal of Computer Science Issues, 2012, Vol. 9, Issue 4, No. 3, pp. 127-133
10. 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.

11. Ayman Madi, Oussama Kassem Zein and Seifedine Kadry, "On the Improvement of Cyclomatic Complexity Metric," International Journal of Software Engineering and Its Applications Vol. 7, No. 2, March, 2013
12. E. B. Passos, D. B. Medeiros, P. A. S. Neto and E. W. G. Clua, (2011) "Turning Real-World Software Development into a Game," Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on, Salvador, 2011, pp. 260-269.,
13. LeanKit. (2019). *7 Lean Metrics to Improve Flow | LeanKit*. [online] Available at: https://leankit.com/learn/kanban/lean-flow-metrics/ [Accessed 12 Nov. 2019].
14. Lowe, S. (2019). *9 metrics that can make a difference to today's software development teams*. [online] TechBeacon. Available at: https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams [Accessed 8 Nov. 2019].
15. Medium. (2017). *13 Essential Software Development Metrics to Ensure Quality*. [online] Available at: https://blog.usenotion.com/13-essential-software-development-metrics-to-ensure-quality-219cfc264ed1 [Accessed 15 Nov. 2019].
16. OpServices | Gerenciamento de TI & Dashboards em tempo real. (2015). *MTBF & MTTR | What are they and what are their differences?*. [online] Available at: https://www.opservices.com/mttr-and-mtbf/ [Accessed 12 Nov. 2019].
17. Raj, L. (2018). *Code Churn — A Magical Metric for Software Quality - DZone Performance*. [online] dzone.com. Available at: https://dzone.com/articles/code-churn-a-magical-metric-for-software-quality [Accessed 12 Nov. 2019].
18. Shkabura, O. (2018). *Top 10 Software Development Metrics to Measure Productivity*. [online] Infopulse.com. Available at: https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/ [Accessed 12 Nov. 2019].
19. Singh (2019). *Line of Code (LOC) Matric and Function Point Matric*. [online] Slideshare.net. Available at: https://www.slideshare.net/diehardankush/loc-metric-and-function-point-metric [Accessed 15 Nov. 2019].
20. Stackify. (2017). *What are Software Metrics? Examples & Best Practices*. [online] Available at: https://stackify.com/track-software-metrics/ [Accessed 8 Nov. 2019].
21. w., B. (2018). *What is Personal Software Process? - PSP*. [online] Explainagile.com. Available at: https://explainagile.com/agile/personal-software-process/ [Accessed 10 Nov. 2019].
22. "11 Test Automation Metrics and Their Pros & Cons | SeaLights." Sealights, Sealights, 2018, www.sealights.io/regression-testing/11-test-automation-metrics-and-their-pros-cons/. Accessed 10 Nov. 2019.
23. Atlassian. "Five Agile Metrics You Won't Hate | Atlassian." Atlassian, 2019, www.atlassian.com/agile/project-management/metrics. Accessed 10 Nov. 2019.
24. ---. "Jira Cloud." Atlassian, 2019, www.atlassian.com/software/jira. Accessed 12 Nov. 2019.
25. "Build Software Better, Together." GitHub, 2018, github.com/. Accessed 15 Nov. 2019.
26. "Code Churn — CodeScene 1 Documentation." Codescene.Io, 2015, codescene.io/docs/guides/technical/code-churn.html. Accessed 14 Nov. 2019.
27. "Engineering Metrics to Improve Continuous Delivery Practices | Velocity." Codeclimate.Com, 2019, codeclimate.com/. Accessed 15 Nov. 2019.

28. “Ethics Explainer: The Panopticon - What Is the Panopticon Effect?” THE ETHICS CENTRE, 16 May 2017, ethics.org.au/ethics-explainer-panopticon-what-is-the-panopticon-effect/. Accessed 9 Nov. 2019.
29. “GitHub Glossary - GitHub Help.” Github.Com, 2019, help.github.com/en/github/getting-started-with-github/github-glossary. Accessed 15 Nov. 2019.
30. “Hackystat.” Hackystat, 2019, hackystat.github.io/. Accessed 15 Nov. 2019.
31. “IBM Knowledge Center.” Ibm.Com, 2018, www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.2/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm. Accessed 13 Nov. 2019.
32. Kraaijeveld, Jos. Exploring Characteristics of Code Churn.
33. “McCabe’s Cyclomatic Complexity | Software Quality Metric | Quality Assurance | Complex System | Complex | Software Engineering.” Chambers.Com.Au, 2019, www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php. Accessed 13 Nov. 2019.
34. “Software Design Complexity - Tutorialspoint.” Tutorialspoint.Com, 2019, www.tutorialspoint.com/software_engineering/software_design_complexity.htm. Accessed 14 Nov. 2019.
35. “Software Engineering Metrics for Engineering Leaders & Management | GitPrime.” GitPrime, 2019, www.gitprime.com/. Accessed 15 Nov. 2019.
36. “What Is Function Point Analysis (FPA)?” Totalmetrics.Com, 2019, www.totalmetrics.com/webhelp2-2/Background_and_Articles_on_Functional_Size/Introduction_to_Function_Point_Analysis/What_is_Function_Point_Analysis_(FPA)_.htm. Accessed 12 Nov. 2019.
37. “Software Engineering | Functional Point (FP) Analysis - GeeksforGeeks.” GeeksforGeeks, 9 Apr. 2019, www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/. Accessed 15 Nov. 2019.