Measuring Software Engineering

- A Report

Anna Helana Maliakal
17324430

Introduction

Technology is developing and transforming at an exponential rate. It is tied to all professions and careers, a drastic change compared to 25 years ago. Teachers use software to keep track of students' attendance, grades and assignments, doctors use video chats to attend appointments with patients, builders and architects use software to design new buildings, the list is endless. But, in a world of rapid changes, discoveries and developments, how is the average tech company to measure the software engineering process.

Productivity is often expressed as the ratio of an aggregate output to a single input or an aggregate input used in a production process, in other words, it is the efficiency with which industries converts input into output. Increasing productivity results in increased profits, which is the main goal of every firm.

$$\text{Productivity} = \frac{\text{Units of output}}{\text{Units of input}}$$

In order to increase profits, industries are continuously searching for areas that need improvement. One way they are attempting to do this is by measuring productivity. However, measuring productivity in the software engineering process has not always been straightforward to calculate. Many questions would be raised about what component of the process could be measured and would prove useful to those analysing it. Software metrics are valuable entity in the entire software life cycle. They provide measurement for the software development, including software requirement documents, designs, programs and tests. Rapid developments of large scaled software have evolved complexity that makes the quality difficult to control. The successful execution of the control over software quality requires software metrics. The concepts of software metrics are coherent, understandable and well established, and many metrics related to the product quality have been developed and used.

This report covers ways in which the software engineering process can be assessed in terms of measurable data, an overview of computational platforms available to perform this work, algorithmic approaches available and the ethical concerns that arise from this kind of analytics.

History of Software Metrics

Software metrics were first mentioned in the late 1960s. After that Lines of Code measure was routinely used as the basis of measuring both programmer productivi0ty and program quality. This was basically using the size of the code as a metric. It was only later on that the quality of the software was being measured, when Akiyama proposed that the number of defects per KLOC (thousand lines of code) be used as a metric. Since then, software complexities, function points and as many as 85 others have been identified as productivity metrics.

Although there had been great success surrounding software metrics (with at least 40 books on software metrics and most software engineering courses teaching about these metrics), much academic metrics research was inherently irrelevant to industrial needs. Irrelevance can be seen at two levels: irrelevance in scope and irrelevance in content.

Irrelevance in scope

Much of the academic work was focused mainly on the use of metrics on small programs. This would not provide to be useful as all the reasonable objectives for applying metrics are relevant primarily for large systems. Irrelevance in scope also applied to the models which require parameters that could never be measured in practise.

Irrelevance in content

Although the need for metrics is to improve processes, much academic work has concentrated on detailed code metrics. In many cases these aim to measure properties that are of little practical interest.

However, in the last 20 years, much development of software metrics has been made corresponding to the fast development of technology. The research paper "*On certain intergrals of Lips Software Metrics"* has defined the characteristics of 'good' metrics as:

- Simple, precisely definable – so that it is clear how the metric can be evaluated.
- Objective, to the greatest extent possible
- Easily obtainable (i.e. at a reasonable cost)
- Valid – the metric should measure what it is intended to measure
- Robust – relatively insensitive to insignificant changes in the process or product.

Measuring and Gathering Metrics

Here we will explore the different metrics and how we can obtain them.

SLOC

"Source lines of code" or SLOC was the first metric developed for quantifying the outcome of a software project. The "lines of code" or LOC has similar meaning and is widely accepted.

Advantages:

- It is easy to measure
- There is a scope for automation of counting

Disadvantages:

- It may include significant "dead code"
- It may include white spaces and comments
- This metric is vague for software reuse

Function-oriented metrics

Function- oriented metrics focus on how much functionality software offers. But functionality cannot be measure directly. Therefore, function-oriented software metrics rely on calculating the function point (FP) – a unit of measurement that quantifies the business functionality provided by the product. Function Point Analysis consists of performing the following steps:

- Determine the type of Function Point count
- Determine the application boundary

- Identify and rate transactional function type to calculate their contribution to the Unadjusted Function Point count (UFP)
- Determine the Value Adjustment Factor (VAF) by using General System Characteristics (GCSs)
- Calculate the adjusted Function Point count

Advantages:

- It stays stable regardless of programming languages used
- It can compute non-coding activities such as documentation
- It can measure non-coding defects in requirements and design
- These are useful for software reuse analysis
- Mathematical conversion of function points into logical code statements is very easy

Disadvantages:

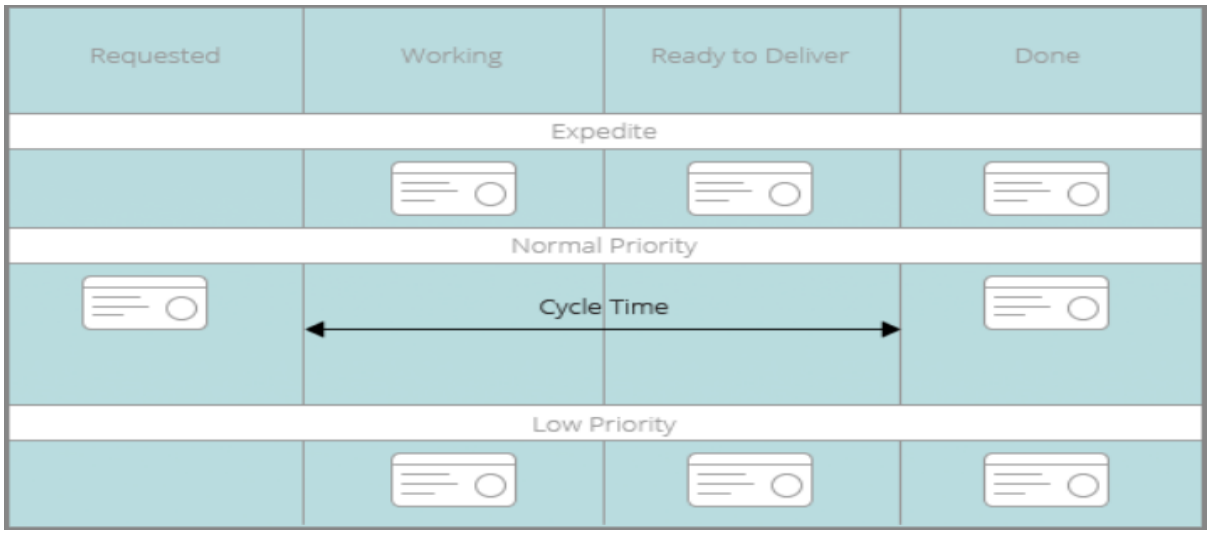
- FP counting requires much experience
- FP counting can be protracted and pricey
- FP counting automation is of indefinite accuracy
- FP counts are unreliable for those projects that are below 15 FP in size.

Errors per FP or Defects per FP can also be used as software metrics. These can be indicators of an information system's quality. Software development teams can use these software metrics to reduce miscommunications and introduce new control measures.

Cycle time

Cycle time a metric that can be used to measure the development team and the process of development rather than evaluating the software itself and its efficiency.

Cycle time describes how long it takes to bring a task or process from start to finish. Within the task or process itself, there can be several individual elements that have their own cycle as shown in the diagram below. Cycle time helps forecast the production process precisely. Shorter cycle times mean an optimized software development process and faster time to market. Longer or lengthening cycle times mean there's a waste or inefficiency in the process and delays for customers.



A project's end-to-end development process can be measured or a particular subset of the process that is significant to the project management. While measuring the cycle time from end-to-end, the time taken to complete each subtask should be recorded, whether it be completing a new functionality or fixing a bug. This allows to pin-point tasks that are lengthening the cycle-time and the software development team can work on how to improve this.

The simplest formula for cycle time is:

Cycle Time = End Date – Start Date

Advantages:

- It is not hard to measure
- Can highlight areas that cause difficulty for the team/engineer
- Gives a more accurate time frame for the project

Disadvantages:

- It is difficult to automate the measuring of the process
- Mathematical calculation does not prove useful in in-depth analysis of the process.

Team Velocity

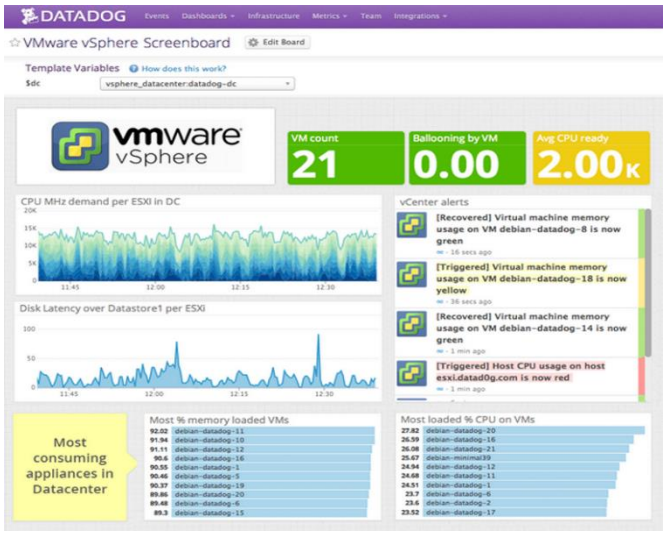
Team velocity measures how many software units a team completes in an iteration or sprint. This is an internal metric that should not be used to compare software development teams. The definition of deliverables changes for individual software development teams over time and the definitions are different for different teams.

Computational Platforms Available

With many metrics available now to measure the software development process, there are equally as many platforms available to assist software engineers in tracking these metrics.

Datadog

Specifically designed for IT operations and development teams, Datadog is a cloud-based solution, which helps manage the lifecycle of application development. Datadog lets firms automate data logs (allowing them to monitor performance), detect performance issues and send notifications via social channels to alert clients about critical issues. The AI-based solution enables professionals to visualize system events and metrics, annotate changes to documents and store the updated data in a unified repository for future reference.



Advantages:

- Easy setup/Hassle-free
- Allows integration with various third-party applications
- Rich reporting capabilities

Disadvantages:

- Not mobile friendly
- Monthly subscription fee required

GitHub

GitHub is widely known throughout the software development industry. It is a repository hosting service for Git that also has a web-based graphical interface. The service includes access controls as well as a number of collaboration features like tools for basic task management and for all projects that are being handled. It can also gather metrics such as lines of code added or deleted. It hosts source code projects in a variety of different programming languages and keeps track of the various changes made to every iteration.

As of 2018, GitHub reported having more than 28 million users. This also allows for companies to identify well-capable, well experienced software engineers.

Features include:

- Having your code reviewed by the community
- Collaborate and Track Changes in your code across versions
- Use Multiple Integration Options
- Develop and Implement a Management Strategy
- Total contributor count and number of commits per contributor
- Track number of issues submitted (and length of time they remain open)

With GitHub, employers can observe the work being completed by an employee. If working in an agile environment, one can take note of the length of time it takes for a task to be completed and how much a certain employee is contributing to the project. The task management tool available also allows to calculate cycle-time, team velocity and FPs.

Raygun

Raygun is a cloud-based networking monitoring and bug tracking application. It is suitable for large and midsize organizations across a variety of industries and offers crash reporting, user monitoring, user tracking, deployment tracking and integrations with other software. On-premise deployments are also available.

Raygun's crash reporting features allow users to identify where clients encounter errors, crashes or performance issues and alert appropriate team members to problems. Workflow management tools help users to organize issue resolution.

Raygun also provides full stack application monitoring. Users can view lists of clients, track session information and analyse how end users interact with applications. Diagnostic information and workflow tools help users identify, reproduce and resolve problems.

This software provides comprehensive technical information around bugs, errors and crashes which allow for faster debugging of the issues.

Algorithmic approaches available

Halstead's Software Physics or Software Science

These are software metrics introduced by Maurice Howard Halstead in 1977 as part of his treatise on establishing an empirical science of software development. Halstead observed that metrics of the software should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. These metrics are therefore computed statically from the code.

Calculation:

For a given problem, let:

- **n1** = no. of distinct operators in program
- **n2** = no. of distinct operands in program
- **N1** = total number of operator occurrences
- **N2** = total number of operand occurrences

From these numbers, several measures can be calculated:

- Program Length: **N = N1 + N2**
- Program volume: **V = N log 2 (n1 + n2)** (represents the volume of information (in bits) necessary to specify a program.)
- Specification abstraction level: **L = (2 * n2) / (n1 * N2)**
- Program Effort: **E = (n1 + N2 * (N1 + N2) * log2 (n1 + n2)) / (2 * n2)** (interpreted as number of mental discriminations required to implement the program.)
- Time required to program: **T = E/18 seconds**
- Number of delivered bugs: **B = V/3000**

COCOMO (Constructive Cost Model)

This is a regression model based on LOC. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers", each with a number of subsidiary attributes: -

- **Product attributes**
  - Required software reliability extent
  - Size of application database
  - Complexity of the product
- **Hardware attributes**

    - Run-time performance constraints
    - Memory constraints
    - Volatility of the virtual machine environment
    - Required turnabout time
- **Personnel attributes**
    - Analyst capability
    - Software engineering capability
    - Applications experience
    - Virtual machine experience
    - Programming language experience
- **Project attributes**
    - Use of software tools
    - Application of software engineering methods
    - Required development schedule

Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an **effort adjustment factor (EAF)**. Typical values for EAF range from 0.9 to 1.4.

The Intermediate COCOMO formula now takes the form:

$$\mathbf{E} = a_i(\mathbf{KLoC})^{(b_i)}(\mathbf{EAF})$$

where:

- **E** is the effort applied in person-months
- **KLoC** is the estimated number of thousands of delivered lines of code for the project,
- **EAF** is the factor calculated above.
- The coefficient $\mathbf{a_i}$ and the exponent $\mathbf{b_i}$ are given in the next table.

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

The Development time **D** calculation uses **E** in the same way as in the Basic COCOMO.

By using this method, the time and size of the project can be estimated and used as a guideline when measuring the software engineering process. Teams can use this to set a goal of when their project should be completed why and if it is not, see what caused the project to lengthen.

Ethical Concerns in Measuring the Software Engineering Process

Measuring employees at a workplace is tolerated as any employer would want to know that the they have made valuable investments at their firm and these will increase returns. However, the question that has to be asked is what are limits to measuring an employee's progress on a given task.

Employee autonomy

Firms have to embrace employee autonomy. Management expert and author Peter Drucker writes that the modern knowledge economy "demands that we impose the responsibility for their productivity on the individual knowledge workers themselves. Knowledge Workers have to manage themselves. They have to have autonomy."

Drucker presents this autonomy as one of the primary reasons why the old manufacturing model falls flat. It's also one of the primary reasons our ownership of our individual productivity becomes more important.

We all come up against some of the same challenges when trying to measure our own productivity, of course, just writ small. But the "autonomy" Drucker talks about means knowledge workers can – and should – have the opportunity to shape their workdays in ways that they know make them more successful and productive.

Automation and delegation free us up to do higher-quality work in the same amount of time, which is a pretty straightforward boost to productivity, even if we can't necessarily measure the objective quality of that work.

Creating these kinds of systems gives employees the autonomy to set their goals alongside managers in alignment with company goals. It provides individual, team-wide and company-wide benchmarks, but for the purpose of reaching common goals.

Workplace Privacy

Workplace privacy describes the extent to which employers monitor and collect information on the activities, communications and private lives of workers.

Workers are often accustomed to personal privacy in their private lives, but employers aren't necessarily obligated to give workers privacy while they're on the job. Employers hire workers to perform specific tasks – time that employees spend taking care of private matters may seem wasteful in the eyes of the employer, so employers may monitor employee activities to determine which workers are wasting time or are engaged in activities that may raise legal issues or security concerns.

Different employers have different workplace privacy policies and employee expectations. For instance, a certain employer might not allow workers to use social networking websites at work and may monitor Web activity to ensure that workers adhere to the rules, while another company might encourage social networking.

Employers may monitor employee activities in a variety of ways, including by using the metrics we have discussed above. According to Bankrate, other possible methods of employee monitoring include tracking Internet usage, archiving computer files, storing employee emails and instant messages, logging keystrokes, recording phone conversations, testing for drugs and maintaining

video surveillance. Businesses may also track the use of key cards or use satellite technology that keeps track of the use of company property such as cars and phones.

Employers that are worried about employees wasting time visiting inappropriate websites or making personal phone calls can block content. For example, a business might block all social networking sites or websites with adult content from its network, or it may block employees from calling certain phone numbers from their work phones. Content blocking may reduce unproductive behaviour without an employer actually monitoring workers and impinging on their privacy.

When working in the technology field, sometimes the development process cannot be quickened as it involves dealing with new software or issues that have only been discovered. Constant monitoring of the employees and forcing many restrictions on them will only impede them from working at their full potential. Employees are more likely to feel under constant pressure to prove to the metric system of their ability that they cannot perform their best.

Employees should be made aware of an employer's policies regarding personal activities at work and the use of technology for activities that aren't related to work. This will bring to their attention what is expected of them in the work place and whether they agree with the workplace’s rules and enforcements.

Conclusion

Nowadays, there are many ways that the software engineering process can be measured. These metrics can prove useful to increase productivity rate and assist software engineers to tackle where they are falling short. However, the metrics can also invade an employee’s privacy and take away form their freedom at the workplace. Therefore, how the results of these metrics and calculations should be used in order to positively affect the worker and the workplace is the responsibility of the employer.

References

Measuring and Gathering Metrics

- https://www.pc.gov.au/news-media/pc-news/previous-editions/pc-news-may-2015/productivity-and-how-measured
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
- https://stackify.com/application-performance-metrics/
- https://www.totalmetrics.com/function-point-resources/what-are-function-points
- https://stackify.com/track-software-metrics/
- https://pdfs.semanticscholar.org/c427/581420e139ce2134e65d9e294e1dafa575e8.pdf

Computational Platforms Available

- https://www.datadoghq.com
- https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946
- https://raygun.com/platform/apm#apm-startup

Algorithmic approaches available

- https://en.wikipedia.org/wiki/Halstead_complexity_measures
- https://en.wikipedia.org/wiki/COCOMO