Conor Loftus

18326036

Measuring software engineering report

**Introduction**

Software is unlike almost all other kinds of products or things that can be built or assembled. The progress of building a tower can be visualised by how tall it is and the estimated time to build the tower can be accurately estimated to a certain degree like taking into consideration how many bricks can be placed an hour. On the other hand, using these kinds of measurements on software engineering hasn’t been anywhere near as effective or successful since the start of measuring software engineering. Corporations cannot look at where their developers are and judge how close they are to their goals, the only thing that can be determined is how much hasn’t been done.

According to the OECD, productivity is described as the relationship between output and the inputs that are required to generate that output (OCED Measuring Productivity, 2001). This relationship can be expressed as the ratio of an aggregate output to a single input or alternatively an aggregate input used in a production process. Productivity can also be described as the efficiency with which industries convert input into output. When a company increases its productivity by improving the ratio of outputs to inputs the company will have increased profits.



Industries and companies search for areas that need improvement and will invest resources in projects or processes that will result in the company having higher returns. Measuring productivity is a good measure for deciding which areas could be improved on. For traditional industries, this approach works well but when applied in the Software Development industry companies run into issues with measuring the productivity of developers as this calculation is not straightforward.

Software metrics are valuable systems in the software industry as they provide measurement for the software developed including software requirement docs, designs, programs, and tests. This report covers ways in which the software engineering process can be assessed in terms of measurable data and gives an overview of computational platforms available to perform this kind of work. The report also provides algorithmic approaches available and the ethical concerns that arise from engaging in these kinds of analytics.

**History of Software Metrics**

Software metrics was first mentioned back in the late-1960’s. After that Lines of Code measure was regularly used as the method of measuring both programmer productivity as well as program quality. Lines of Code used the size of the code as a metric for work done by programmers. Later on, the quality of the software began to be measured after a proposal from Akiyama that the number of defects per KLOC (thousand lines of code) be used as a metric instead (Akiyama, 1971). Since this proposal, software complexities, function points and many other attributes of code have been identified as potential productivity metrics.

There have been both practical and academic approaches to performing and researching software metrics although many academic metrics are essentially irrelevant to industrial needs. There are two kinds of irrelevance: irrelevance in scope and irrelevance in context.

**Irrelevance in scope**

Much of the academic work was focused on the use of metrics with small programs but it was discovered that this would not provide to be useful as all the reasonable objectives for applying metrics are only relevant for large systems. Irrelevance in scope also applied to the models which require parameters that could never be measured in practice.

**Irrelevance in context**

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

In the last twenty years there has been lots of progress made with the development of software metrics in relation to the fast development of technology. According to the paper “On certain integrals of Lips Software Metrics” the characteristics of good metrics is defined as:

* Objective, to the greatest extent possible
* Easily obtainable
* Valid – the metric should measure what its meant to measure
* Vigorous – relatively insensitive to non-important changes in product
* Simple, accurately definable – so that its clear how the metric can be evaluated

**Measuring and Gathering Metrics**

As mentioned earlier there are numerous different metrics that can be used to derive productivity from a software development team or project.

**SLOC**

SLOC or “Source lines of code” was the first metric developed for quantifying the outcome of a software project.

Advantages

* Easy to measure
* Availability of scope for automatic counting of lines

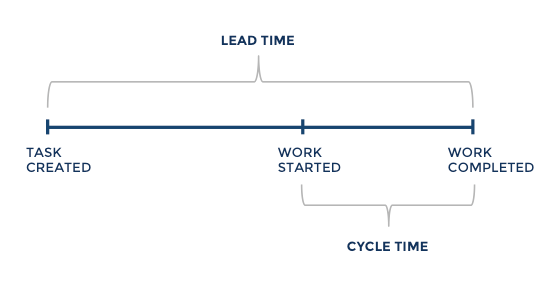
Disadvantages

* The inclusion of non-useable “dead code”
* Inclusion of white spaces and multi-line comments

**Cycle Time**

This metric is used to measure the development team and the process involved in the development of software rather than focusing on evaluating the software itself and its efficiency.

Cycle time reports the length of time it takes for a process to finish. There can even be cycle times of individual tasks within the process being done. Cycle time is a forecasting tool that helps the user plan the production process precisely. Shorter cycle times suggests that the development process has an optimized software development process and faster time to market while conversely longer cycle times suggests that there is some kind of waste or inefficiency in the process which leads to delays for the customer. There are numerous formulas to represent cycle time but one common method to calculate it is: Cycle Time = End Date – Start Date.



While measuring the cycle time from end-to-end the time taken to finish each task or subtask should be noted. This includes noting down features implemented, functions written, and bugs fixed. This allows the development team to quickly figure out which genre of task is lengthening the cycle time and the team can tend to this issue if it becomes an issue.

Advantages

* Highlights areas that are creating difficulty for team
* Provides accurate time frame for the project
* Ease in calculating and measuring

Disadvantages

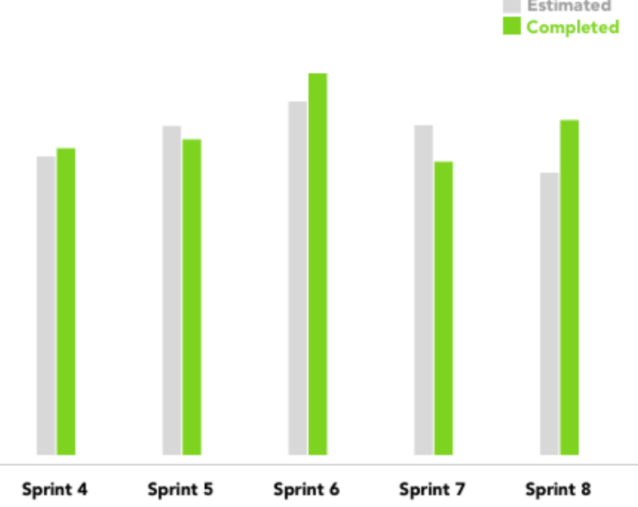
* Difficulty in automating the measuring process, has to be done manually
* The calculation of cycle time doesn’t provide in depth analysis of project progress

**Team Velocity**

Team Velocity is a metric mostly used in Agile or Scrum software development methodologies. This metric measures the number of software units a team completes in an iteration or sprint. This is an internal metric and very specific to the team that the value was calculated from, so this metric cannot be used to compare development teams.

Project velocity is calculated by taking the average of the total completed units over the last few sprints. For example, a team’s velocity could be: (82 + 85 + 100+ 80 + 90) / 5 = 87.4

What this figure indicates is that the product manager should expect the development team to complete at least 87.40 units worth of work in the next sprint.



Advantages

* Assists team make better predictions about teams capacity
* Highlights issues before they become problems

Disadvantages

* Not a precise unit of measurement as new members join, and project scopes change
* Inability to compare the velocity of multiple teams

**Function-orientated metrics**

Function-orientated metrics focus on how much functionality the software offers but functionality itself cannot be measured directly. As a result, function-orientated software metrics rely on calculating the function point (FP). Function point is a unit of measurement that quantifies the business functionality provided by the product. Function point analysis performs the following steps:

1. Determine the type of Function Point count
2. Determine the application boundary
3. Identify and rate transactional function type to calculate their contribution to the Unadjusted Point count (UFP)
4. Determine the Value Adjustment Factor (VAF) by using General System Characteristics (GCSs)
5. Calculate the adjusted Function Point count

Advantages

* Useful for software reuse analysis
* Remains stable regardless of the programming language used in the project
* Availability to compute non coding activities such as writing documentation
* Ease of mathematical conversion of function points into logical code statements
* Availability to measure non coding defects in the requirements and design

Disadvantages

* Function point counting requires adequate level of experience to perform
* Function point counting isn’t a cheap option due to the layers of work involved
* Function point counts are unreliable for projects that are below 15 FP in size.

Errors per FP or Defects per FP can also be used as software metrics and they can be indicators of an information systems quality. Software development teams can use these software metrics to reduce errors in communication and bring in new control measures.

**Computational Platforms Available**

With the creation and research into many different forms of software measuring metrics there has come a demand for platforms to help developers and managers track these metrics.

**GitHub**

GitHub is a popular repository hosting service for version control using Git and as of 2018 they reported having more than 28 million users. GitHub offers a web based graphical interface for parsing through commit history as well project files and other repository data. The service also grants access controls as well as a number of collaboration features like tools for task management. GitHub gathers metrics such as lines of code added or deleted from a commit. It hosts source code projects with support for many different programming languages and tracks the changes that occur during each iteration with the availability for a “rollback” or step back to a previous state of the project.



Features

* Availability for community review of code
* Implement a management strategy
* Display all contributors and number of commits per contributor
* Allow users to collaborate and track changes in code across versions and states
* Track issues and bugs submitted and display thread for discussion of said bugs

GitHub allows employers to observe the work being committed by an employee. When working in an agile setting, GitHub can track the length of time it takes a task to be completed and can measure how much a certain employee is contributing to the project. The task management tool also provides repository owners with the ability to calculate certain metrics such as Cycle-time, Team velocity and FP’s.

**Datadog Cloud Monitoring**

This software was designed for IT operations as well as development teams. Datadog is one of many cloud based solutions which helps their users manage the lifecycle of application development. The service lets teams automate data logs which allows them to monitor performance, detect performance issues and send notifications through social media to alert clients about issues. Datadog also offers an AI-based solution enabling developers to visualize system events and metrics, add changes to documents and store updated data in one of its repositories for future reference.

The benefits that come with using Datadog is that the service is simple to set up locally or virtually, allows the integration with numerous third-party applications and has abundant metric reporting features.

The main downside to the service is that unlike GitHub the service requires a subscription fee to visualize most metrics generated.

**Raygun**

Raygun is another cloud-based network monitoring and bug tracking application similar to Datadog. The service is aimed at large and midsized branches of companies and operates in numerous different industries. Raygun offers many different metrics such as crash reports, user usage monitoring and tracking, deployment tracking and integration compatibility with other software.

Raygun promotes their service as a way to “Discover poor performing parts of your application, diagnose the root cause and prioritize engineering resources for the biggest performance gains”. The crash reporting feature enables users to find places where clients encounter errors and alert appropriate team members to problems. Workflow management tools are also offered and help users to organize issue resolution which boosts work efficiency.

Raygun offers comprehensive technical information around errors, bugs and crashes which allows for more rapid debugging of the issues. Although Raygun offers very in-depth analysis and monitoring this comes at a high cost which is far more expensive than Datadog and GitHub.

**Algorithmic Approaches**

**Constructive Cost Model**

The Constructive Cost Model which can be abbreviated to COCOMO is a regression model constructed around the principle of LOC (lines of code). It’s a procedural cost estimate model for software projects and often used as a process of reliability predicting the various parameters associated with making a project such as effort, cost, size, quality, and time.

Intermediate COCOMO computes software development effort as a function of program size and a set of cost drivers which include the following: hardware, subjective test of product, project characteristics and personnel involved in project. This extension considers a set of four cost drivers each having a number of lesser attributes:

* **Personnel attributes**
  + Ability of analysts
  + Software engineering capability
  + Experience with virtual machines
  + Experience with applications
  + Experience with different programming languages
* **Hardware attributes**
  + Run time performance limitations
  + Memory limitations
  + Changes in the virtual machine environment
  + Requirements for turnabout time
* **Product attributes** 
  + Required software reliability extent
  + Size of application database
  + Complexity of project
* **Project attributes** 
  + Use of software tools
  + Application of software development methods
  + Required development schedule

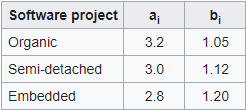
Each of the mentioned attributes get a rating on a six-point scale which ranges from “very-low” to “extra-high”. Then an effort multiplier is applied to the rating. The product of all the effort multipliers results in a figure known as the effort adjustment factor (EAF) and typical values for EAF range rom 0.9 to 1.4.

The formula for intermediate COCOMO takes the form:

**E=ai(KLoC)*(bi)*(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 effort adjustment factor mentioned above
* The coefficient ai and the exponent bi are given in the following table.



The development time D calculation uses the value E in the same way as in the basic COCOMO. By using this method, the size and time of a project can be estimated and used as a basis when measuring the software engineering process. This can be then used as a goal for teams for when the project should be completed.

**Halstead’s Software Physics**

These software metrics were introduced by Maurice Halstead in 1977 as part of his report on creating an empirical science of software development. Halstead observed that software metrics should mirror the implementations and expressions of algorithms in different languages but should be independent of their execution on a specific platform. All of these metrics are then computed statically from the code.

For a given problem let:

* n1 = number of distinct operators in program
* n2 = number of distinct operands in program
* N1 = total number of operator appearances
* N2 = total number of operand appearances

For all of these numbers, numerous measures can be computed:

* Program Length: **N = N1 + N2**
* Program Effort: **E = (n1 + N2 \* (N1 + N2) \* log2 (n1 + n2)) / (2 \* n2)**
* Time required to program: **T = E/18 seconds**
* Specification abstraction level: **L = (2 \* n2) / (n1 \* N2)**
* Program Volume: **V = N log 2 (n1 + n2) Bits**
* Number of delivered bugs: **B = V/3000**

**Ethics of Measuring Software Engineering**

Ethics are the concerns that humans have had for figuring out how best to live. Measuring employees who do software development has become an increasingly debated topic where employers argue that they want to monitor what their employees are doing while in the workplace in order to figure out if employees have done what they are supposed to. Others argue what are the limits to this approach of measuring employees and observing employees daily.

**Employee independence**

Management expert Peter Drucker explains 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”. He also presents autonomy as one of the main factors why the previous manufacturing model for engineering fails when there is an attempt made to apply it to software development. Hence why it’s one of the major reasons why our responsibility for individual productivity becomes far more essential.

The independence that Drucker speaks about means that knowledge workers should have the chance to shape their workdays in a manner that leads to them being more successful and productive in their approach to work. This type of independence further promotes the idea of automating the boring stuff, cutting out meaningless repetitive tasks that hamper productivity which frees up the work force to do higher quality work. Although this kind of work would be difficult to measure objectively the boost in productivity has unbounded benefits.

**Privacy in the workplace**

Workplace privacy is defined as the extent to which employers observe and collect information on the activities, communications, and private lives of their employees.

One of the benefits of living in the current age is the right to privacy and often employees take this belief into the workplace where employers are not required nor compelled to provide the same access to privacy when they are in work. When a employer hires an employee there are two expectations on both sides, the employer expects the employee to perform to the necessary standard in the job that they were hired to do while the employee expects to be paid for doing said job. Employees spending time on private matters while on the job would be seen as wasteful by employers so to observe the severity of this activity and ensuring that work actually gets complete firms and employers may set up monitoring systems to determine which employees are productive and which are not.

Depending on the country and the laws and even the firm, different employers may have different workplace privacy policies and expectations from employees. To provide an example, an employer may prohibit employees from using social media in work and might actively monitor web activity going on to ensure the rules are adhered to. The company down the street may have the complete opposite approach and encourage the use of social media in terms of boosting productivity.

Employers have a variety of methods in which they can monitor and collect information on employees and these methods are getting more sophisticated as technology improves. Employers can use the metrics that have been discussed earlier to get an overview of the work done in a software development team. Other methods include tracking internet usage, archiving files on employee computers, storing employee emails and messages, logging keystrokes from employee computers, recording phone conversations, testing for drug use, and storing CCTV footage. In more sophisticated businesses key cards and GPS location is used to monitor the same activities.

In the software development industry, the development process cannot be quickened as it involves dealing with new software or issues that have only been discovered. Constant monitoring of employees and forcing unnecessary restrictions on them would only impede and hamper progress. Employees are more likely to feel under constant pressure to prove to the metric system of their ability that they can perform their best.

**Conclusion**

In recent years there has been an increase in the ways that software engineering processes can be measured. These metrics can prove useful in increasing the productivity rate and can assist software developers figure out and deal with areas that they are falling short on. Conversely, these metrics invade employees privacy and take away the sense of freedom that other professions have. Therefore, the results of these metrics and calculations should be used to positively affect the worker and the workplace which is the responsibility of the employer.

**Bibliography**

A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analysing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

Akiyama, F. (1971). An Example of Software System Debugging. IFIP Congress.

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

Lindquist, Matthew & Sauermann, Jan & Zenou, Yves. (2015). Network Effects on Worker Productivity.

OECD Measuring Productivity (2001). [online] Available at: <http://search.oecd.org/economy/outlook/1959006.pdf> [Accessed 10 November 2020].

P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, and W. E. J. Doane, “Beyond the personal software process: Metrics collection and analysis for the differently disciplined,” in Proceedings of the 25th international Conference on Software Engineering. IEEE Computer Society, 2003.