**Measuring Software Engineering**

**Jason Yang 18322733 12/11/2020**

**Abstract**

This report considers the ways in which the software engineering process can be measured

and assessed in terms of measurable data, an overview of the computational platforms

available to perform this work, the algorithmic approaches available and the ethical

concerns surrounding this kind of analytics.

**Introduction to Software Engineering**

*“If you can’t measure it, you can’t improve it.” – Peter Drucker*

Companies are spending tens of millions of dollars on their engineering teams even thought they have absolutely no way of tracking or measuring how well they are doing, let alone whether they are getting better or worse.

In just about every other profession, sports, marketing, sales, there are established ways of measuring how your organization is doing and how it can improve. Most of these methods are generated through software. Ironically, in the field of software, we have not solved this problem.

We’ve come a long way since the term ‘Software Engineering’ was coined by Margaret Hamilton. However, the problem of measuring software engineering remains.

This report will explore the problem by assessing measurable data, providing an overview of the current computational platforms available, detail the algorithmic approaches available, and consider the ethics surrounding such analytics.

**Measurable Data**

**Source Lines of Code**

Source lines of code (SLOC), also known as lines of code (LOC), is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

While SLOC shows a high correlation with effort, it is not a perfect measure.

*“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” – Bill Gates*

In software engineering, efficiency is valued, and top developers will be able to develop the same functionalities with far less code. In other words, having more lines of code is often worse.

In an academic paper *titled* [*‘Analysis of Course Lines of Code (SLOC) Metric’*](https://www.researchgate.net/publication/281840565_Analysis_Of_Source_Lines_Of_CodeSLOC_Metric) *by Kaushal Bhatt, Vinit Tarey, and Pushpraj Patel*, the authors outline nine flaws in measuring software by this metric:

1. Lack of Accountability

The coding phase of a project usually accounts for only 30% to 35% of the overall effort, therefore many think it isn’t useful to measure the productivity of a project with Lines of Code.

1. Lack of Cohesion with Functionality

While experiments have repeatedly confirmed that effort is highly correlated with SLOC, functionality is less well correlated with SLOC. As well as this, SLOC is a poor productivity measure of individuals because a developer who develops only a few lines may still be more productive than a developer creating more lines of code.

1. Adverse Impact on Estimation

As mentioned in point 1, estimates based on lines of code can go wrong, in all possibility.

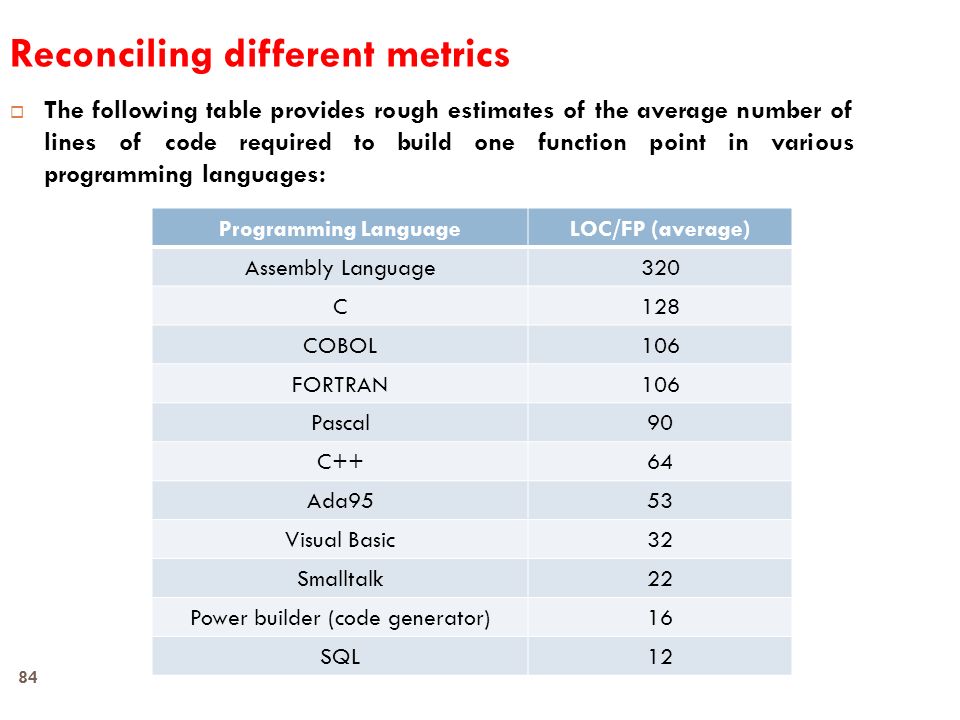
1. Developer’s Experience

An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience, even if they are using the same language.

1. Difference in Languages

Depending on the programming language used, the same functionality could take up more or less lines of code. As a consequence, the amount of effort required to develop the application would be different.

The difference of LOC in different programming languages is nicely summarized in a [project management presentation](https://slideplayer.com/slide/10787293/) published in 2016 by Corey Henderson.



1. Advent of GUI Tools

With the advent of GUI-based programming languages and tools such as Visual Basic, programmers can write relatively little code and achieve high levels of functionality.

This results in variation between languages; the same task that can be done in a single line of code (or no code at all) in one language may require several lines of code in another.

1. Problems with Multiple Languages

Today, software is often developed in more than one language. Very often, a number of languages are employed depending on the complexity and requirements.

1. Lack of Coding Standards

There is no standard definition of what a line of code is. Do comments count? Are data declarations included? What happens if a statement extends over several lines?

1. Psychology of Programmer

A programmer whose productivity is being measured in lines of code will have an incentive to write unnecessarily longer code. This is undesirable as increased complexity can lead to increase cost of maintenance and increased effort required for bug fixing.

The in-depth analysis provided in this paper clearly outlines that source lines of code is not an adequate measure of the productivity of a software engineer. The paper seems to recommend the use of function point analysis.

**Function Point Analysis**

Function Point Analysis (FPA) is a sizing measure which was first made public by Allan Albrecht of IBM in 1979. The FPA technique expresses the amount of functionality a program provides to a user and is an alternative to lines of code for measuring software size.

From an excerpt of *Advances in Computers in 1997* titled [*‘Software Cost Estimation: A Review of Models, Process, and Practice’*](https://www.sciencedirect.com/topics/computer-science/function-point) *by Fiona Walkerden and Ross Jeffery*, the authors discuss the relevance of measuring software by function points.

A function point value for a system is derived by summing and weighting counts of five features of the system: inputs, outputs, queries, logical files, and interfaces to other systems.

A function point value for a system is derived by summing and weighting counts of five features of the system: inputs, outputs, queries, logical files, and interfaces to other systems. To add these counts together, there must be models that can convert the counts to a common unit.

Function point analysis is a powerful source of data. A drawback of function point analysis is that although it deals well with the size and usability of output produced by a programming team, it doesn’t always measure other aspects of production quality such as maintainability, reliability, stability and legibility.

**Number of Commits**

*“Not everything that can be counted… counts.” – Albert Einstein*

Another metric for measuring the software engineering process is based on the number of commits performed. If no commits are done then no work is done, and in general, small, frequent commits support greater transparency, collaboration, and continuous delivery.

However, counting the number of commits does not tell about the value being delivered and if anyone wanted to, developers could do more commits while providing little value.

In other words, the size and frequency of commits are not highly correlated with the completion of a project and therefore, commits between programmers should not be compared.

The number of commits is a flawed way of measuring the software engineering process. Instead, it should only be viewed as an indication of activity. Even still, different developers will have varying personal preferences as to what should be committed.

**Pull Request Count**

Following from the number of commits, we can also count the number of pull requests. Through pull requests we can measure the pull request lead time, time to merge, pull request size, pull request flow ratio, and pull requests discussion. SourceLevel.io is an example of a SaaS company that uses these metrics to measure efficiency.

The problems with counting pull requests is that it doesn’t factor in size or effort required, and encourages unnecessarily small, chunked pull requests. Again, this metric is better suited to being viewed as an indication of activity.

**Code Coverage**

Code coverage measures the number of lines within the code that is covered by the devised test cases. High code coverage means more informative data since more of the work is examined by tests. This metric can help management obtain an insight into the thoroughness of the code from each developer.

A possible shortcoming is that the tests must be sufficient in covering all eventualities, otherwise optimal code coverage does not translate into a bug free software application.

**Velocity**

Velocity is a metric for the work done which is often used in agile software development. The main idea is to help teams estimate how much they can complete in a given time period based on how quickly similar work was previously completed. Velocity is a relative measure meaning that the raw numbers have little meaning, it’s the trend that matters.

The problem with velocity as a metric is that the sizing is done before work is completed, not after, and this undermines the usefulness of the estimation process. People are incentivised to inflate the number of points they deliver. Finally, velocity destroys the careers of highly productive people who are working on very difficult problems.

**Lead Time / Cycle Time**

This metric measures the time elapsed between the identification of a requirement and its fulfilment. It can be easily measured by taking note of the project start and completion dates.

This metric can be used to assess the efficiency of the software engineering process and evaluate the capability of individual developers. When the software development team obtains a new project, they can evaluate the results of these metrics from previous similar jobs and provide a data-driven estimate for the time required for completion.

**Computational Platforms**

**GitHub**

GitHub is a web-based version control and collaboration platform for software developers with over 50 million users worldwide. It is an open-source platform where managers can evaluate data from developers and track the software engineering process.

By interrogating the GitHub Rest API, you can find plenty of metrics, for example:

* Number of commits
* Frequency of commits
* Number of changes and contributions
* Lines of code (using the cloc package)

Platforms such as Screenful Metrics, Pull Panda, and many more can be integrated with GitHub to allow management to create visual dashboards and automated reports based on the metrics obtained.

**Personal Software Process**

The Personal Software Process (PSP) is a structured software development process that is designed to help software engineers better understand and improve their performance by bringing discipline to the way they develop software and tracking their predicted and actual development of the code. It clearly shows developers how to manage the quality of their products, how to make a sound plan, and how to make commitments.

In other words, the Personal Software Process acts as a framework for self-evaluation and best practice in software engineering. The PSP improvement process can be outlined by the following steps:

1. Define the quality goal
2. Measure product quality
3. Understand the process
4. Adjust the process
5. Use the adjusted process
6. Measure the result
7. Compare the results with the goal
8. Recycle and continue improving

The PSP uses forms to collect information from developers, which includes a project plan summary, defect-recording log, design checklist and code checklist among others. To provide flexibility, these forms are not automated. Unfortunately, this has ultimately resulted in problems with data quality.

**Velocity - Code Climate**

Velocity by Code Climate is a software product that performs automated analysis of code to identify and flag potential code errors, security vulnerability and instances of flawed coding methodology. It can collect and present quantitative and qualitative metrics describing and summarising various aspects of the code.

Velocity integrates with tools like GitHub, BitBucket, Jira, Git, and Slack to help:

* Pinpoint exactly where pull requests get stuck from open to deploy
* Distinguish productive work patterns to know when to help and when to avoid interrupting engineers
* Give data to support important decisions
* Use data as a common language for communicating engineering successes

**Flow – Plural Sight**

Flow by Plural Sight manages pull requests at scale, provides a powerful visualization of code review dynamics, and measures and tracks how collaborative a group is over time.

This powerful tool is trusted by start-ups and large enterprises such as Amazon Web Services alike. It’s easy to see why. Flow allows you to:

* See how much time is spent refactoring old code vs. new work
* Recognize project bottlenecks and remove them
* Get concrete data around commit risk, code churn, and impact to better guide discussions
* Know if code reviews are productive and positive
* Observe team dynamics and patterns in the code review process

**Waydev**

There are currently over 300 companies relying on Waydev. Their tool helps automatically measure engineers’ performance by analysing their codebase, pull requests and tickets. Similar to CodeClimate, Waydev integrates smoothly with GitHub, BitBucket, GitLab, Jira and more.

**Algorithmic Approaches**

**Computational Intelligence**

Computational Intelligence refers to the ability of a computer to learn a specific task from data or experimental observations. It is particularly useful where mathematical approaches are inadequate in solving a problem and it helps us to better understand the software engineering process.

There are three main areas under Computational Intelligence:

1. Neural Networks

Neural Networks are not algorithms themselves. However, they provide a framework for machine learning algorithms to work together and process data inputs. Artificial intelligence and machine learning can provide an insight into trend and commonalities in the investment process and as such, neural networks can be compared to the human brain learning from past examples.

In other words, neural networks are interconnected groups of nodes that use examples to generate identifying characteristics from the learning material they process.

1. Fuzzy Systems

Fuzzy systems can be used to model uncertain problems such as linguistic imprecision by altering the use of traditional logic to include things that may be partially true. As a result, we can use this output to perform approximate reasoning.

1. Evolutionary Computation

This is used to solve optimization problems by generating, evaluating and modifying a population of possible solutions. Evolutionary Computation include genetic algorithms, genetic programming, evolutionary programming, evolution strategies, multi-objective optimization and more.

**Multivariate Analysis**

Multivariate analysis deals with the statistical analysis of data collected on more than one dependent variable. There are two families of techniques that one can use under multivariate analysis – supervised and unsupervised learning methods.

*Supervised learning methods* are when you have an input variable (x), an output variable (y), and you use a method to study the mapping function from the input to the output such that y=f(x).

The task is to estimate the mapping function so well that when you have new unlabelled input data that you can forecast the output variables.

In other words, you are already aware of the underlying groupings within the data, and supervised learning methods can be used to classify new data points added to the cluster map.

Some examples of supervised learning methods that can be used to measure software engineering process are linear discriminant analysis, k-nearest neighbours and logistic regression.

*Unsupervised learning methods* involve taking in input data (X) and no matching output variables (Y). The aim is for the algorithm to model the underlying structure, groupings or distribution in the data in order to learn more about it.

Some examples of unsupervised learning methods include cluster analysis, hierarchical analysis and k-means clustering.

**Bayesian Belief Network**

In 1999 *Fenton & Neil* published their paper [*‘Software Metrics and Risk*’](https://www.academia.edu/23492845/Software_metrics_and_risk) where they highlighted the fact that utilizing the Bayesian Belief Network model as a substitute for the issue of imperfect metrics results in the most success.

The Bayesian Belief Network model applies well established statistical theory known as Bayesian statistics to software engineering and intentionally accounts for uncertainty, incomplete and subjective information while making explicit assumptions that have previously been hidden from traditional statistical approaches.

The model can collect data such as the defect counts at different stages of testing, the size of complexity metrics at each development phase, and the approximate development and testing effort.

The Bayesian Belief Network algorithmic approach has been utilized for a wide variety of applications, from empowering the Microsoft Office wizard to decision-making on the International Space Shuttle.

**Ethics**

**Data Collection**

The main objective for the collection of data should be to improve decision-making, aid in identifying problem areas and improve internal operations. Developers should be in favour of the data collection to improve the overall software engineering process of the organization.

However, when management implement software that measures their software engineering process, it is common for developers to feel under constant scrutiny. (And understandably so.) To avoid negative repercussions and conflict due to implementing such tools, it is imperative that there is transparency with regard to the data that management are collecting from its developers.

Regardless of transparency, companies must be wary that metrics tend to make developers’ lives miserable and harm teams.

**Data Regulations**

Recently, society has begun concerning itself more with these issues surrounding the measurement of professional productivity. GDPR (General Data Protection Regulation) was approved by the European Union in 2016 and will dictate part of how to correctly collect data on employees.

There are prudent regulations to follow when it comes to collecting personal information, data decision making, storing data for long periods and being transparent about access to information and how it is stored as well as much more.

Failure to comply with these social standards of behaviour can lead to huge penalties. For example, GDPR authorities are able to issue fines of up to EUR 20 million or 4% of annual worldwide turnover.

**Data Usage**

Management must ensure that the analysis of the data is fair whilst avoiding the promotion of unwanted working practises. There is a huge responsibility placed on management to utilize the data in an ethical and non-discriminatory manner.

The data should not be used to reward or punish teams or people. Especially as even good metrics can be harmful when they are not presented in the best way.

**Conclusion**

In this report, I have discussed four aspects of measuring the software engineering process – the metrics, platforms, algorithmic approaches to both measure and best utilize data while also addressing the ethical issues surrounding such data analysis.

There remains countless challenges that make it virtually impossible to create a perfectly consistent and accurate analysis of the software engineering process.

Through advancements in machine and deep learning technologies, we can expect an improvement in our ability of measuring the software engineering process in the years to come.

**Bibliography**

**Introduction**

[Margaret Hamilton – Wikipedia Page](https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)#:~:text=She%20later%20founded%20two%20software,%2C%20both%20in%20Cambridge%2C%20Massachusetts.&text=She%20is%20one%20of%20the,the%20term%20%22software%20engineering%22.)

**Measurable Data**

[GitHub Universe Talk 2019](https://www.youtube.com/watch?v=cRJZldsHS3c)

[Source Lines of Code – Wikipedia Page](https://en.wikipedia.org/wiki/Source_lines_of_code)

[Analysis of Source Lines of Code Metric Paper](https://www.researchgate.net/publication/281840565_Analysis_Of_Source_Lines_Of_CodeSLOC_Metric)

[Project Management Presentation – Henderson 2016](https://slideplayer.com/slide/10787293/)

[International Function Point Users Group](https://www.ifpug.org/about-function-point-analysis/)

[Software Cost Estimation: A review of Models Process, and Practise – Walkerden, Jeffery, 1997](https://www.sciencedirect.com/topics/computer-science/function-point)

[5 Metrics Engineering Managers Can Extract from Pull Requests – Source Level](https://sourcelevel.io/blog/5-metrics-engineering-managers-can-extract-from-pull-requests)

[Code Coverage - Wikipedia](https://en.wikipedia.org/wiki/Code_coverage)

[Code Coverage - IBM](https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.15/gtpd3/d3ccodecovben.html)

[GitHub Universe Talk 2019](https://www.youtube.com/watch?v=cRJZldsHS3c)

[Velocity - Wikipedia](https://en.wikipedia.org/wiki/Velocity_(software_development))

**Computational Platforms**

[GitHub](https://github.com/)

[Screenful](https://screenful.com/)

[Pull Panda](https://pullpanda.com/)

[Personal Software Process – Wikipedia Page](https://en.wikipedia.org/wiki/Personal_software_process)

[Introduction to the Personal Software Process – Humphrey 1997](https://www.win.tue.nl/~wstomv/quotes/humphrey-psp.html)

[What is Personal Software Process – ExplainAgile.com](https://explainagile.com/agile/personal-software-process/)

[Code Climate](https://codeclimate.com/)

[Flow by Plural Sight](https://www.pluralsight.com/product/flow)

[Waydev](https://waydev.co/)

**Algorithmic Approaches**

[IEEE Computational Intelligence Society](https://cis.ieee.org/about/what-is-ci)

[Multivariate Analysis – Toward Data Science](https://towardsdatascience.com/an-introduction-to-multivariate-data-analysis-ece93ceb1ed3)

[Supervised vs. Unsupervised Learning – Toward Data Science](https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d)

[Supervised and unsupervised machine learning algorithms – Jason Brownlee 2016](https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/)

[Bayesian Network – Wikipedia Page](https://en.wikipedia.org/wiki/Bayesian_network)

[Bayesian Belief Networks – Jason Brownlee 2019](https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/)

[Software Metrics and Risk – Fenton & Neil 1999](https://www.academia.edu/23492845/Software_metrics_and_risk)

**Ethics**

[GDPR – General Data Protection Regulation](https://gdpr-info.eu/)

[All you need to know about GDPR – Shrivastava 2018](https://hackernoon.com/all-you-need-to-know-about-gdpr-explained-8e336a1987ea)