

CSU33012 Software Engineering

Measuring Engineering

How Software Engineering is Measured and Assessed

Cillian Byrne, 18322806

November 1, 2021

Contents

[1. Introduction 2](#_Toc92095685)

[2. Measuring Engineering activity 3](#_Toc92095686)

[2.1 Why companies measure Software Engineering productivity? 3](#_Toc92095687)

[2.2 Software Engineering productivity metrics 3](#_Toc92095688)

[2.3 Benefits of Software Engineering Metrics 3](#_Toc92095689)

[2.4 Difficulties in measuring Software Engineering Activities 5](#_Toc92095690)

[2.5 The Right Engineering Productivity Metrics 5](#_Toc92095691)

[3. Data Gathering and Performance Calculations 6](#_Toc92095692)

[3.1 GitPrime (Pluralsight Flow) 6](#_Toc92095693)

[3.2 WayDev 7](#_Toc92095694)

[3.3 Codacy 7](#_Toc92095695)

[3.4 Personal Software Process 8](#_Toc92095696)

[4. Algorithms 8](#_Toc92095697)

[4.1 Halstead Complexity Measures 8](#_Toc92095698)

[4.2 Cyclomatic Complexity 9](#_Toc92095699)

[4.3 Function Point 10](#_Toc92095700)

[5. Ethics 11](#_Toc92095701)

[5.1 Privacy 11](#_Toc92095702)

[6. Conclusions 12](#_Toc92095703)

[Index 12](#_Toc92095704)

# Introduction

Software engineering can be defined as the systematic, disciplined, and quantifiable approach to the development, operational and management of software. Software Engineers apply this approach to programming that is used to develop software with the aim of improving quality, time or budget efficiency, alongside structured testing. Software engineering is typically used for large and intricate software systems, with software engineers responsible for the design and structure of these systems.

The primary characteristics of Software engineering are its inherent complexity, its lack of industry standard procedures within the discipline and the fact that it is developed and not produced. These features make it very different from other disciplines.

The complexity of software makes it extremely difficult to build and understand, especially as it progresses throughout its useful lifespan, as the original code base becomes older and more mature it often becomes error-prone and harder to change due to the implications it has on other areas of the code.

Another reason for complexity in Software engineering is due to the lack of tractable modelling of the product and process and the lack of the knowledge required to build or understand software systems. A developer or a development team that is highly skilled in the tools and the knowledge of task can be exponential more efficient than that of a more rudimentary level. The problem of visibility in software systems attributes to this and more skilled teams will use their higher levels of knowledge to limit the impact of poor quality, structure and functionality of the software build using good processes, requirements, and testing.

Also, Software is made in a development process, a much newer type of creation than Manufacturing for example. This means that there has been more limited learning in this field and best practices to emphasize quality and efficiency have not yet became wholly standard across the industry. This is yet another are where strong developers and development teams can show their worth.

What I will be looking to explore in this essay is how to turn these aspects of performance and others into measurable data and quantify it. This is a problem with Software engineering as difficult as it is to build and understand it is just as daunting to try and measure individual performance and efficiency of developers. Since the term Software Engineering was defined in the 1960s the there have been attempts to quantity it. There is no industry standard measurement that covers everything, finding one would allow more efficient allocation of resources in companies and development teams. Companies instead have used their own combination of data to try, and analysis software engineering and I will detail some of these in the next section of the report.

# Measuring Engineering activity

## 2.1 Why companies measure Software Engineering productivity?

This is the goal of any Engineering manager or CTO, to be able to understand an evaluate your workforce to allow you to accurately review staff, knowing who is carrying a team and who is being carried. To be able to set timeframes for products, set sprint goals and budget targets.

## 2.2 Software Engineering productivity metrics

These are some of the common metrics companies often rely on Abi Node a Senior Product Manager at GitHub calls them the “Flawed Five” for reasons we will explore.

1. **Lines of code**

This is a self-explanatory metric for productivity the more lines of code an engineer writes the more productive they as an individual are.

1. **Pull requests**

The amount of submitted contributions to an open development project.

1. **Commits**

A change or revision to an individual file.

1. **Velocity points**

Velocity is a term used in agile software development to illustrate the “rate of progress” for a team or a set of teams (i.e., a project/program).

1. **“Impact”**

This measures the impact any change has on the progress of the software development project.

## 2.3 Benefits of Software Engineering Metrics

The benefits of using each of these different measures to evaluate software engineering is very different. The gamification of these metrics is inevitable but you hope to incentivize the right type of activity so each different metric will change the way a software engineer operates.

**Lines of code:**

This will ideally result in software engineers writing more good code, but there is no way in guaranteeing this. This metric is also easy to track and measure taking little additional effort other than logging the number of lines of code committed.

**Pull requests:**

This is like lines of code commits in it being an easily tracked form of measurement. By simply logging GitHub contributions form a personal account. This in theory will benefit software engineers that complete the most tasks but can be gamed by tackling smaller tasks more often or breaking up tasks into smaller requests rather than larger ones but in general this incentivizes more activity.

**Commits:**

This is very similar to pull requests in gathering and performance incentives.

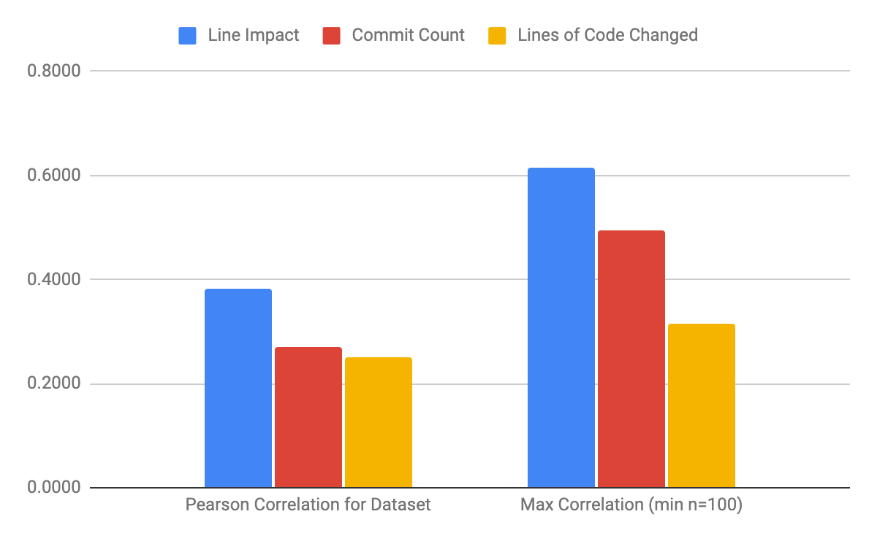
**Velocity Points:**

This is one of the harder metrics to track as you need to predefine points of development before starting work. You need points evenly spaced in work effort to accurately track the velocity or rate of progress. But with good planning and understanding of the task ahead this is one of the harder measures to game as the only way to make positive gains is to complete goals and progress towards project completion. The only thing to be wary of is b-lining towards stated goals and leaving other unstated tasks incomplete. Velocity points benefit the team by incentivizing a quick rate of progress in project completion.

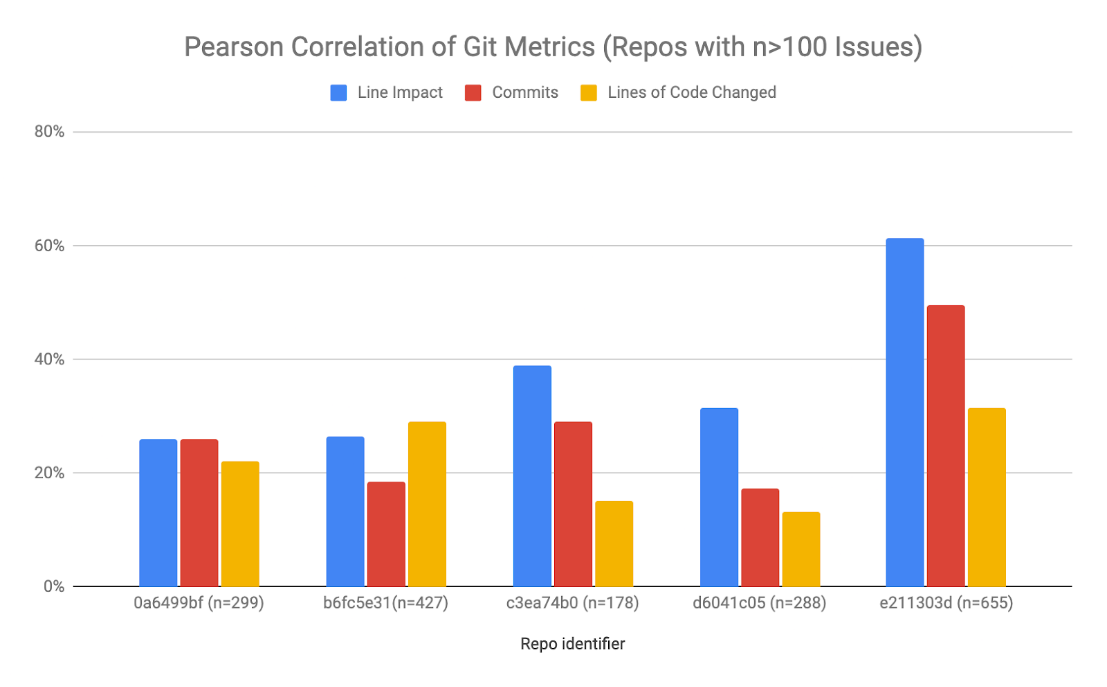
**Line Impact:**

It measures how much each commit evolves your project's code base. This metric is very gaming resistant as it tracks how much a line of code influences the total project in a whole. "Lines of code" is a distant cousin to Line Impact in that each changed line of code has the potential to accumulate Line Impact. The biggest difference between "lines of code" and Line Impact? The "lines of code" is much noisier, whereas Line Impact is built for the expressed purpose of measuring changes that matter.

This can be seen in the below study which measures the correlation of some of the metrics discussed above with story points (points hit that signify progress in a sprint).



Average software metric Pearson correlation with Story Points, and max correlation among repos with at least 100 issues assigned Story Points.



## 2.4 Difficulties in measuring Software Engineering Activities

Many output metrics are hard to tie to individual contributors, they may be more suitable for team performance. These include

* Bug fixes
* Change requests
* Uptime

The same is also true for input metrics like:

* Staff hours
* Hardware
* Technical debt

This means that individual performance is hard to separate from the team and it is much easier to organize the development group into small teams where you can more accurately evaluate them.

The gamification of metrics is also a problem in evaluation. Whatever metrics a company uses ends up creating incentives to preform well in these statistics. This is good but what you want as a company is for those incentives to line up with your own. This ends up not being the case for many of these “flawed five” detailed in section 2.2. In the case of lines of code this will incentivize individuals to write unnecessarily long and imprecise code, which is hard to decipher and could create issues downstream. Pull requests incentivize small changes that require less work. With Velocity points you can over-estimate the points to completion. This all creates an environment where people are working to not work and could quickly result in an unproductive software development team.

## 2.5 The Right Engineering Productivity Metrics

A good metric should do most of the following:

* Truly reflects productivity as defined above/correlates closely with revenue
* Includes all work output
* Incorporates non-Engineering work
* Gaming-resistant
* Objective and independently verifiable
* Language-agnostic
* Can be compared across projects
* Gives the best people the hardest assignments
* Easy and cheap to measure

Its better to measure a process rather than an output. When the number of process completed increases, productivity also increases. This means tracking:

* Code review turn-around time
* Pull request size
* Work in progress
* Time-to-open

Every measure is imperfect and its our goal to try and find the most accurate measure to evaluate performance. Its best to use multiple measures so we can cancel out the pros and cons of individual measures and get a more accurate overall picture.

Each team will also need to be looked at in a unique manor, some teams may have more emphasis on turn-around-time in an environment where time is critical, other may highlight work-in-progress and look for teams to have lots of projects on the go at the one time. A way to help incentive different areas is a grading system. Ranking teams on a 1-5 scale of productivity from a terrible-excellent rating on how they are preforming in each individual area. Once all metrics are scored giving reviews based on multiple measures will help to prevent gamification. Generally speaking, the better the evaluation on performance, the happier developers will be and greater work effort you’ll get from engineers as they know their effort is properly being tracked.

# Data Gathering and Performance Calculations

In the current software engineering environment, we have platforms such as GitHub that track a vast amount of data. We need to gather his data and draw meaningful conclusions from it. There are lots of different Platforms that provide this service the ones I’ll be detailing in this report are as follows:

* GitPrime
* Waydev
* Codacy

These are popular tools among developers and let’s find out why.

## 3.1 GitPrime (Pluralsight Flow)

GitPrime is a popular Git analytics platform, which offers visibility into engineering performance, showcases efficiency, and helps improve productivity.

The Key Metrics Tracked:

* Churn – Code that is rewritten in the first 21 days after being committed.
* Impact – The complexity of code changes.
* Influence – The amount of follow-up commits made after a review left on a pull request.
* Reaction Time – The amount of time it takes for a reviewer to answer a comment addressed to them.

## 3.2 WayDev

Waydev is the leading Development Analytics tool that helps engineering managers gain better visibility into their software development team. Waydev focuses on increasing team productivity and helps managers become data driven.

WayDev is evenly focused on providing code-level analytics, code review metrics, and project management insights, with a wide array of functionalities and flexibility.

The Key Metrics Tracked:

* Work Log – This feature helps you keep track of what everyone is doing individually or as a team. The Work Log report is especially helpful when managing large teams.
* Daily Update – The commit activity spread across four work types: New Work, Legacy Refactoring, Help others and Churn.
* Developer Summary – provides an aggregate view of individual developers’ activity, over the course of a month or a week, across all work types. Can easily spot work patterns and progress, as well as blockers.
* Project Timeline – The Project Timeline is an overview of how work focus and volume changed over time, and how certain events affected your team’s output.
* Velocity Report – The Velocity Report will help you determine the amount of work your team can cover and visualize how much value each sprint delivers so you can accurately predict the team’s workload for future sprints.
* Sprints – The Sprints report helps engineering managers to monitor the health of their sprints.
* Review Collaboration – aims to provide a perspective of how engineering teams work collaboratively. The Review Collaboration report is split into submitter and reviewer metrics. The balance between speed and thoroughness in code review becomes easier when you can zoom in into the team’s collaboration stats.

## 3.3 Codacy

Code Climate Velocity is a development analytics tool focused on providing custom reports and a complete view of engineering teams’ performance. They do this by inspecting all the data from your Git repositories and then compressing it into real-time analytics.

The Key Metrics Tracked:

* Coding Days – measures how much time per week developers spend coding.
* Review Cycles – measures how much a pull request has gone back and forth between the submitter and the reviewer.
* Cycle Time – measures the time elapsed between a developer’s first commit to the final delivery.
* Time to Merge – measures the entire time spent reviewing the code.

# Algorithms

More and more companies are looking to implement analytics into the measurement of software engineering. Once we have collected the data discussed above in part two, we can use many different types of algorithms we can use to interpret this data in a more beneficial way. The algorithms I will be looking at more in depth are the following:

* Halstead Complexity Measures
* Cyclomatic Complexity
* Function Point Analysis

## 4.1 Halstead Complexity Measures

Complexity is the measure of a state of events or things that have interconnected links and complicated structures. In Software Engineering as design becomes more complex the interconnections are immense. This makes it difficult to understand all at once. Software design complexity is difficult to assess without complexity metrics.

In 1977, Mr. Maurice Howard Halstead introduced metrics to measure software complexity. Halstead has said, “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”. His measures allow us to evaluate testing time, vocabulary, size, difficulty, errors, and efforts for C/C++/Java source code.

He has classified these as indicators to check complexity:

|  |  |
| --- | --- |
| Parameters | Meaning |
| n1 | Number of unique operators |
| n2 | Number of unique operands |
| N1 | Number of total occurrence of operators |
| N2 | Number of total occurrence of operands |

Above we look at the Halstead Complexity Measures we are moving on to look at the details of Halstead Metrics:

|  |  |  |
| --- | --- | --- |
| Metric | Meaning | Mathematical Representation |
| n | Vocabulary | n1+n2 |
| N | Size | N1+N2 |
| V | Volume | Length\*Log2 Vocabulary |
| D | Difficulty | (n1/2)\*(N1/N2) |
| E | Efforts | Difficulty\*Volume |
| B | Errors | Volume/3000 |
| T | Testing Time | Time = Efforts/S where s = 18 seconds |

Advantages of Halstead metrics:

* Simple to calculations.
* Measures overall quality of the programs.
* Predicts the rate of error.
* Predicts maintenance effort.
* Does not require the full analysis of programming structure.
* Useful in scheduling and reporting projects.
* Can be used for any programming language.

Halstead measures do depend on the complete code to be calculated, which has issues code that isn’t in a functioning state yet an It also cannot be used in a predictive manor, estimations cannot be made by using it.

## 4.2 Cyclomatic Complexity

McCabe, in 1976, proposed Cyclomatic Complexity Measure to quantify complexity of a given software. Every program is made up of statements that are executed in order to preform a task, and other statements that make decisions on which statements need to be executed. These statements change the flow of a program. The more of these flow changing statements are in a piece of code the more complex it will be.

* To make a flow control graph we do the following:
* Break program in smaller blocks, delimited by decision-making constructs.
* Create nodes representing each of these nodes.
* Connect nodes as follows:
  + If control can branch from block i to block j and draw an arc.
  + From exit node to entry node and draw an arc.

Diagram

Description automatically generated

Example of a Flow Control Graph

Cyclomatic Complexity formula:

|  |
| --- |
| V(G) = e – n + 2P |
| e = total number of edges |
| n = total number of nodes |
| P = the number of connected components |

Cyclomatic Complexity analysis is very useful in determining the independent path executions for testing, making sure each path has been tested once. It can also help with risk associated with the program. It helps to use this type of analysis early on in for it to be more beneficial.

Advantages of Cyclomatic Complexity:

* It can be used as a quality metric, gives relative complexity of various designs.
* It can compute faster than the Halstead’s metrics.
* It is used to measure the minimum effort and best areas of concentration for testing.
* It can guide the testing process.
* It is easy to apply.

## 4.3 Function Point

Function Point Analysis is used to measure the size of the software. This type of complexity analysis focuses on the features and functionality of the code base. Function point counts five different parameters:

|  |
| --- |
| External Input |
| External Output |
| Logical Internal Files |
| External Interface Files |
| External Inquiry |

**External Input**

This is every unique input to the system from the outside. It is considered simple if the input count is low affects less internal files, Complex if it has a high input count and affects more internal files and Average if in-between.

**External Output**

All outputs from the systems are counted in this category, the output must also be unique. It is considered Simple if the output count is low, Complex if the count is high and Average if the count is in-Between.

**Logical Internal Files**

Software systems maintain Logical internal files, to maintain functional information and to function properly. This logical data may contain both functional and control data. It is considered Simple if the number of record types is low, Complex if high and Average if its in-between.

**External Interface Files**

If a Software System needs to share its files with an external Software system, it will need to pass the file for processing or as a parameter to a function. It is considered Simple if the number of records in a Shared file are low, Complex if they are high and average if they are in-between.

**External Inquiry**

An inquiry is a combination of inputs and outputs, a user will send information to inquire about and the system will respond with and output. The Complexity of a query is greater than that of a Eternal input and external output. It is considered Simple if a query needs low processing and yields small amount of output data, Complex if it needs high processing and yields a large amount of output data and Average if it is in-Between.

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Simple | Average | Complex |
| Inputs | 5 | 3 | 1 |
| Outputs | 8 | 6 | 4 |
| Enquiry | 3 | 7 | 8 |
| Files | 6 | 2 | 3 |
| Interfaces | 9 | 3 | 10 |

Example of raw Function Point data

There are 14 Different characteristics used to adjust these values based upon the environment complexity. All ratings are then summed up as N. The value of N ranges from 0 to 70 (14 types of characteristics x 5 types of ratings). It is used to calculate Complexity Adjustment Factors (CAF), using the following formulae:

**CAF = 0.65 + 0.01N**

This in turn can be used to calculate other various metrics such as:

* Cost = $ / FP
* Quality = Errors / FP
* Productivity = FP / person-month

Advantages of Function point analysis:

* Determine the size of a purchased application package by counting all the functions included in the package.
* Discover the benefit of an application package to their organization by counting functions that specifically match their requirements.
* Measure the units of a software product to support quality and productivity analysis.
* Estimate the cost and resources required for software development and maintenance.
* A normalization factor for software comparison.

# Ethics

## Privacy

One of the biggest unseen markets in the world today is that of personal data. Many businesses collect your data to analysis and proliferate. This is a big concern of many individual particularly in the software engineering world. It is a topic that becomes more and more troubling with more knowledge in that area of expertise and if software engineers are good at one thing it is their ability to analysis data. What we have been discussing today is how to track and gain insights on the work of individual engineers to an extremely accurate level and we have yet to really discuss how the individual affected will feel with every single keystroke and piece of code written being tracked and analysed. This is a very intimate invasion of privacy and I wonder how it would affect workers in the long run knowing that there is the constant pressure of an unseen algorithm calculating your performance level as a worker. This may be seen as a motivation factor by managers, but it could quickly spiral into an all-consuming work-environment where every promotion, bonus and redundancy will reference your performance metrices. Are workers entitled to the mystery surrounding their individual performance? What if your impact isn’t truly caught by the metrics? These are question that will be answered in the coming years as the measurement of software proliferates throughout the sector.

# Conclusions

This report details several software engineering metrics that are used currently in tracking software engineering performance. It details how they perform their strengths and their weaknesses. We then go on to discuss the ideal traits of a software engineering metric and some lesser-known alternative metrics that could be better at meeting the requirements laid out. We then move on to discussing the platforms we could use to gather the data from sources like GitHub. The report lays out the strengths of each and some of there unique features. These platforms make it easier than ever to analyse software engineering. Looking at some algorithms that can help to further form our opinion on the value of work done. I focus on Complexity analysis as in my opinion it is a good measure of how detailed a good is as software systems tend to become more and more complex as their scope grows. We then finally look at the Ethics of all this data tracking and if it is actually a good idea to monitor employees so closely. Through the points stated we get a clearer picture of how and why software Engineering is measured and assessed.

# Index

References:

Story (2021). Story Point Correlation Research (February 2021, Unabridged Version). [online] Google Docs. Available at: https://docs.google.com/document/d/18j\_LC-u\_rhToVp3GBHLiheTOmaVdqKz0DqIUT84ZslI/edit#

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

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.

‌ Silverman, Rachel (Nov 2, 2011). "Latest Game Theory: Mixing Work and Play — Companies Adopt gaming Techniques to Motivate Employees". Wallstreet Journal.