Measuring Engineering

17341914 | [miblack@tcd.ie](mailto:miblack@tcd.ie)

Michael Black

# Introduction

Software engineering is the systematic, disciplined and quantifiable approach to the development, operation and maintenance of software.[[1]](#endnote-1) It is concerned with the techniques used to produce quality software products for use in industry. Like any product, steps (that will be examined in this report) are being taken to analyse and ultimately better the development process.

Initially one would assume that productivity correlates directly to volume of work contributed. However, it is known in developing that solutions of equal merit can be achieved through different methods. One solution may contain more lines of code yet carry out an identical function. In this scenario, rewarding the solution with more lines of code would be rewarding inefficiency. It would also be counterproductive to reward the smaller solution before it is complete, as its efficiency is yet to be understood. This dilemma has spurred a massive debate on what is the best way to measure productivity in relation to software engineering. It would be naïve to assume the source lines of code approach is the most reliable. With vast amounts of trackable data available to software managers, large numbers and variations of metrics have surfaced showing predicting trends of increasing accuracy.

The body of this report tackles the measuring of software engineering under four distinct headings: Measurable Data, Computational Platforms, Algorithmic Approaches and Ethical Concerns. Respectively, they will provide insight into and examples of common software metrics, popular code reviewers, select algorithms and privacy and ethical violations relating to the measuring process. Following the body there is a brief reflection and conclusion that will wrap up everything discussed in this report. A bibliography is included at the end.

1.1 Defining measurement in software engineering

Formally, measuring can be defined as, the process by which numbers or symbols are assigned to attributes of entities in the real world, in such a way as to describe them according to clearly defined rules.[[2]](#endnote-2) Such entities in software may include lines of code, time spent, numbers of errors etc. When quantified, these values are stored and compared to other measurements in the attempt to suggest trends and test hypotheses.

1.2 Why measure software engineering?

Like any profession, the measurement of good progress must be credited and laziness vice versa. In college we measure our engagement with modules by means of a grade the lectures assign. Ones intelligence in a certain topic should coincide, in theory, with the mark they got. They can tell us truths about a person or object by means of an intuitive charting system. They same is true in software engineering. The goal of tracking software metrics is to determine the quality of the current process or product.[[3]](#endnote-3) Benefits within an organisation includes:

* Identified goals and managed workloads
* Credit productivity and acknowledge areas of improvement
* Increased clarity within an organisation
* Increased profits

1.3 In Theory vs In Practice

The Software Metrics Best Practices Report[[4]](#endnote-4) highlights the gap between theorized metrics and its implementation. The report explains how it proves a challenging task to apply principles of measurement to software in practice. One major problem is the assumption that the components of the model are identical or even similar to the project in question. Researched models theorists use may not translate well in practice and might lead to skewed results. To explain this, consider that metrics research involves the following method:

* Collection of data and product measures in a specific development project
* Make a model based off these measurements
* Claim it can predict future project characteristics of similar circumstances

A software manager using this model must assume that the model has a reasonable similarity to their current project. An example of a factor to consider when making this assumption is: Are the N number of developers used in this model more qualified or experienced than my team? If so, how does this affect the model?

# Measurable Data

Software metrics have been proven to better time management and production capabilities. Though there is substantial subjectivity in the measuring of software engineering there are still many implementations of metrics regardless. Software metrics can be broken down into two subheadings, product metrics and process metrics. Product metrics is the measure of various characteristics of the software product. Process metrics is the measure of various characteristics of the software development process.

## Product metrics

Lines of Code

Lines of Code (LOC) is a software metric used to determine productivity based on the volume of code or number of lines written. Though it is naïve to assume volume equates to effort, it tends to be a good indication of work. There are two types of LOC measures, physical and logical. Physical refers to all lines of code excluding comments whereas logical refers to the measures all logical statements. i.e. a logical line is not bound to a physical line but a statement, if a statement or comment spills over more than one line it is still considered just one logical line. Physical LOC is much easier to measure; however, it is prone to logically irrelevant formatting conventions. For this reason, Physical and logical LOC measures can vary hugely even with the same codebase.

Active Days

This charts only the times when a developer contributes to code. It pays no attention to any prior research or administrative tasks. The purpose of this metric is to highlight patterns of interruption. A trend may form showing peak/low productivity during a period.

Code Churn

Code Churn is a measure which shows you the rate at which you code evolves. It represents the number of lines of code that were modified, added or deleted in a specified period. Unlike raw lines of code commits this method credits only the productive code committed and acknowledges the code changed or deleted. Code churn is a very good indicator of general effort and can be a great tell for things like post-release deficits. It is a warning sign if there is a spike in a code churn nearing a deadline as it suggests the code becomes more volatile towards the end. Stabilized code is preferable before delivery.

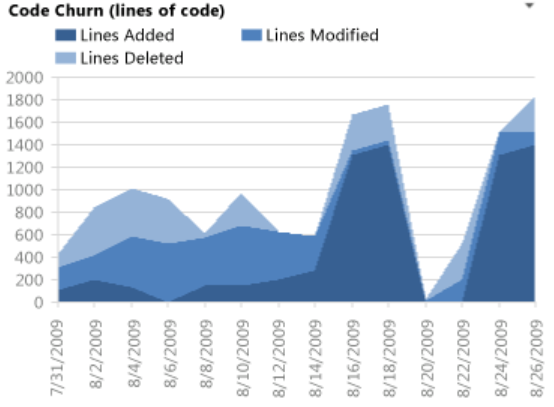


Fig i) – Code Churn

## Process Metrics

Lead Time

This metric’s focus is towards response time to assigned projects. It shows the time elapsed between the identification of a requirement and its fulfilment. Lead time begins when the customer makes a request and ends when the product is delivered, i.e. it is what the customer sees. Note that lead time refers to the time undergone, not the effort undergone. The project may be complete, but the lead time doesn’t stop until it is delivered. The example below is specific to bug detection, when a bug is detected the ticket is created and when bug fix is live the ticket is closed.

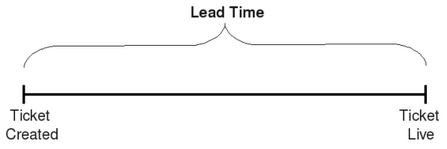


Fig ii) – Lead time

Cycle Time

This describes how long it takes to change the software system and implement the change to production. It is a more mechanical measure of process capability when compared to lead time. Cycle time begins when the work on a request starts and ends at delivery. Cycle time cannot be shorter than lead time. Most companies specify an SLA (Service Level Agreement) which defines the period that the product should be completed in, logically this will be the same at the lead time. Reducing the cycle time can in turn reduce the lead time and therefore we can see why analysing these models is relevant from a company’s perspective when performance is prioritised.

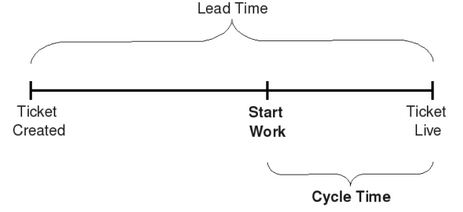


Fig iii) – Cycle time

# Computational Platforms

Code review is a quality assurance technique in which code is analysed for the purpose of accelerating and improving the process of software development. Several platforms have been established that contain all-inclusive management environments to review code. These code review tools typically include reports and visual graphs that display employee effort. Naturally there is regular error or in inefficiency in human written code, so code review attempts to front this issue. There are many platforms available to facilitate this process. Some of which include GitPrime, Collaborator and Phabricator. Each will be discussed below.

3.1 GitPrime

GitPrime is a software analytics company founded by (CEO) Travis Kimmel and (CMO) Ben Thompson in 2015. Based in Durango, Colorado, this platform provides reports on developer work patterns and provides analysis on team productivity. Frustrated by the difficulties in communicating with non-technical engineering stakeholders, engineering Travis Kimmel began creating reports to objectively quantify engineering throughput.[[5]](#endnote-5) From this the ideology behind GitPrime was born.

GitPrime aggregates git data into easy to understand insights and reports. It uses data from version control, ticketing systems and the existing system to provide visibility into a software engineering team.[[6]](#endnote-6) An attractive UI displays visually focused reports and analysis. There are four core fundamental metrics that show productivity levels. These are:

* Active days
* Commits per day
* Impact
* Efficiency

Other features include a timeline dashboard that displays a 30-day average as well as custom targets. For an increase in cost, the user can opt for an upgrade which unlocks a review workflow dashboard. This illustrates all pull requests in the last two weeks. A customer review says “Having metrics for software development is a tough problem to solve. They get it right. Pricing on the high side but makes sense given the cost of development salaries.”[[7]](#endnote-7) It is an overall finished and clean application that oozes quality but also cost in equal measure.

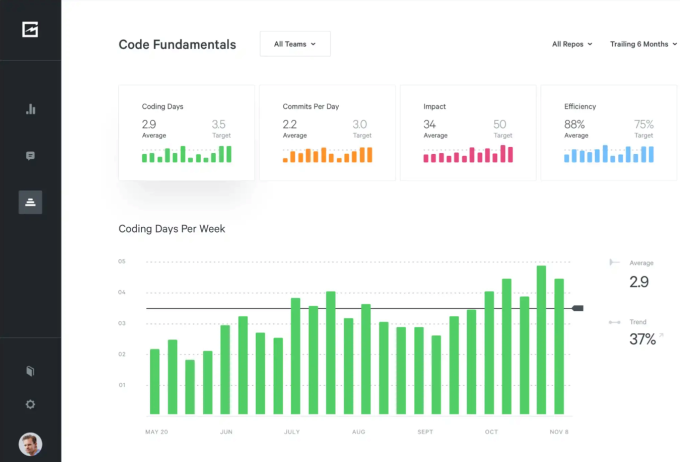


Fig iv) - GitPrime UI

3.2 Collaborator

Collaborator is a code and document review application by SmartBear Software based in Somerville, Massachusetts. Limited not only to software engineering this company has a popular prominence in industries like Automotive, Healthcare, Aerospace, Finance and Embedded systems [[8]](#endnote-8). Founded in 2003 by Jason Cohen, this application was originally called Code Collaborator. The app has since been sold in a merger of Automated QA, Pragmatic software and SmartBear and received a collaboration jolts award for its contribution.

Features of Collaborator include:

* **Comprehensive Review Capabilities** – Review source code, design docs, requirements, user stories, test plans, and documentation.
* **Proof of Review** – Proof with electronic signatures & detailed reports that meet regulatory compliance standards.
* **Support for 11 SCMs** – including Git, SVN, TFS, Perforce, CVS, ClearCase, RTC etc.
* **Integrations** – with GitHub, GitLab, Bitbucket, Jira, Eclipse, Visual Studio etc.
* **Real-Time Updates** – Threaded chat shows conversations as well as highlights changes & defects for visibility during each code review. [[9]](#endnote-9)

An attractive feature of Collaborator is its customizability, allowing for flexible implementation among vast arrays of developments settings. The custom template-based approach is compromised by a few components including:

* **Custom fields and workflow**: Collaborator offers the ability to add custom fields to templates enabling users to capture information specific to their team. This is often used for addressing compliance reports or including links to external tools. Custom review workflows are also supported
* **Participant, Rules, Groups and Subscriptions**: Participant roles can be used to set rules for required review participants. There are four standard participant roles in Collaborator: Author, Reviewer, Moderator, and Observer. All of which are customizable. Participant groups work in a similar manor. You can subscribe individuals or groups to certain review templates, i.e. more experienced development may be assigned to more thorough review templates.
* **Custom review Checklist**: Research has suggested that code reviewers that use checklists outperform those who don’t.[[10]](#endnote-10) Building custom checklists is supported in this application. They can be an effective way to ensure compliance burdens and security best practices.

3.3 Phabricator

Phabricator is a collection of software development tools including code review, repository hosting bug tracking, project management and more. It is largely based on Facebook’s internal tools. It was set up in 2010 by principle developer Evan Priestley. The concept was initially discovered during Priestley’s time at Facebook. Upon realising the potential, he left to concentrate on the development of this project. Phabricator is completely open source and can be freely downloaded from the website.[[11]](#endnote-11) A payment plan exists optionally for hosted Phabricator. The software runs on PHP, however, the server does not work on the windows operating system.

There are four distinct components in Phabricator:

**Differential:** This is the pre-push code review tool and workflow. Differential makes use of the technology Arcanist, a wrapper script that provides simple command-line API to manage review. Code review in Phabricator uses a slightly different methodology from pull requests. Simply put:

1. The author sends his changes for review
2. The reviewer inspects the change and either approves it or requests changes
3. When both parties are satisfied, the author may push upstream

**Diffusion:** This is the post-push code audit tool and workflow. Can host repositories locally or observe remotely. Audit is a less powerful workflow than review though it has some of the following benefits:

* Users do not need install Arcanist
* Little training needed
* Useful to keep a tab on issues discovered after review

**Manifest:** The task management software used to track issues or bugs throughout the development cycle. It is built to be intuitive while allowing users to remix items through custom fields, custom forms, API integration and workboard organisation.

**Phriction:** A wiki. Documents here are organised as a hierarchy, similar to a file system.

# Algorithmic Approaches

Viewing metrics from another light, this section discusses algorithmic approaches to measuring engineering. Namely, this section will discuss Halstead’s unique contribution to software measurement and secondly will touch on what promise AI can bring to this industry.

4.1 Halstead’s Complexity Measures

In 1977, Maurice Howard Halstead[[12]](#endnote-12) introduced metrics to measure software complexity. The measures were developed to calculate the complexity of source code by means of the operators and operands in the module. According to Halstead, “A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands”.[[13]](#endnote-13) His metrics statically evaluate testing time, vocabulary, size, difficulty, errors, and efforts for C/C++/java source code. His metrics are now included in a variety of commercial code reviewing tools.

By counting the tokens and determining which are the operators and operands the following parameters can be collected.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| *n1* | Number of distinct operators |
| *n2* | Number of distinct operands |
| *N1* | Number of operator instances |
| *N2* | Number of operand instances |

Fig v) – Metrics parameter and meaning table

From these numbers, several measures (below) can be calculated. The table below can optionally be viewed in the metrics viewer report.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Meaning** | **Formula** |
| *n* | Vocabulary | *n1* + *n2* |
| *N* | Size | *N2* + *N2* |
| *V* | Volume | *N* \* log2 *n* |
| *D* | Difficulty | *n1*/2 \* *N2*/*n2* |
| *E* | Effort | *V* \* *D* |
| *B* | Errors | *V* / 300 |
| *T* | Testing time | *E* / *k* |

Fig vi) – Metrics Report

Where k is the ‘stroud’ number with a default value of 18.

4.2 Artificial Intelligence and Machine Learning

Measuring software engineering has been an immensely challenging task. It must constantly change and adapt to new technologies. With artificial intelligence (AI), the purpose of the testing discipline changes entirely. AI now requires continued measuring and feedback way after the product has been launched. It must gather feedback from the real users as it is the premise on which AI operates. This is dis-similar to other contemporary technologies that cease measurement once the development progress has been finished (though analysis can be carried out on the already published code).

Artificial Intelligence is the branch of computer science concerned with the simulation of intelligent behaviour in computers.[[14]](#endnote-14) Machine-learning is an application of AI that provides systems the ability to improve from experience without being explicitly programmed.[[15]](#endnote-15) It is focused on the development of computer programs that can cleverly interact with its data. Machine learning are often categorised into the following headings:

Supervised Learning: The application of previously learned data to predict new future events. In other words, starting from a know dataset, the learning algorithm makes a function to produce predictions on output values. The algorithm compares the output with the correct intended output and makes to the model accordingly until it achieves a desired level of accuracy. An example of supervised learning is Linear Regression.

Unsupervised Learning: Differs from supervised learning in so far as there is no intended correct output from which the learning algorithm can compare. The training information is neither classified nor labelled. The system does not arrive at the correct output; however, it can draw inferences from a dataset to describe hidden structures. Example – Apriori Algorithm.

Semi-Supervised Learning: Lies somewhere in between the two above as they both utilise labelled and unlabelled data for training. Typically uses a small amount of labelled data with a large amount of unlabelled data. The systems that use this method can considerably improve learning accuracy. Example – Pseudo Labelling.

Reinforcement Learning: This method interacts with its environment by producing actions and receives rewards or errors based on these actions. The learning curve is developed through trial and error and delayed reward approach. Simple reward feedback is required for the agent to learn which action is best. This fosters the path to an ideal behaviour within a specific context. Example – Markov Decision Process.

# Ethical Concerns

So far, the focus of this report has been on performance and accuracy in measuring engineering. In this section, we arrive at the ethical implications that these metrics induce. We will touch on an issue hugely prevalent in ethics called data protection and explore what bearing excessive measurement can have on recipients. This is will be accompanied by a reflection section in which I will add personal input.

5.1 Data protection

Individuals have privacy rights in relation to the processing of their personal data. Stored data can have severe implications on a person’s career when handled incorrectly. In software metrics, (specifically process metrics) data obtained on an employee must be used in accordance with data protection laws.

As of May 2018, there has a new European framework known as the General Data Protection Regulation (GDPR) that has changed the rules of on data protection.[[16]](#endnote-16) It offers a more uniform interpretation and application of data protection standers in the EU. Additionally, May 2018 saw the signing of the Data Protection Act 2018 which revises the previous acts in1988 and 2003. This act established a new data protection commissioner and gave further effect to the GDPR in areas where member states have flexibilities.

The UN Declaration of Human Rights, the International Convenant on Civil and Political Rights and other international and regional treaties recognize privacy as a fundamental human right.[[17]](#endnote-17) A representative from privacy international says “Privacy protects us from arbitrary and unjustified use of power by states, companies and other actors. It lets us regulate what can be known about us and done to us, while protecting us from others who may wish to exert control.”[[18]](#endnote-18) Being of such vital importance to an individual, it is clear to see why any violation of this is as much a legislative breach as it is an ethical one.

5.2 Case Study

A case study featured in the Annual Reports published by the Data Protection Commission[[19]](#endnote-19) reveals an example of when you are within your rights to prosecute an organisation by means of the DPC. In 2017 the DPC received a complaint from an individual who received unsolicited marketing messages from the AA Ireland Limited. He informed the DPC that he needed renewal on his motor insurance and decided to shop around for quotes. One such company he called was the AA. They offered him a quote that he didn’t seem persuaded on. Before he ended the phone call, he asked assurance from the agent who answered his call that his information would not be used for marketing purposes. The agent assured him it would not.

The next day however, the complainant encountered aggressive selling practices by means of a text message from AA offering a discount on the previously quoted price. The subsequent day he received another from the same company. Aware that this was a clear violation of his instruction not to receive any marketing communications, he filed a complaint to the Data Protection Commission.

During the course of the investigation, the AA admitted that they did not receive consent to send the messages to the complainant. As the DPC had previously issued a warning in separate circumstances to the AA in relation to unsolicited marketing communications, in this instance the DPC decided to initiate prosecute proceedings. At Dublin Metropolitan District Court on 14 May 2018 AA Ireland Limited entered a guilty plea to one offence. It also agreed to cover the prosecution costs incurred by the DPC.

5.3 Reflection

Though the above case study was a trivial example of data misconduct this should highlight the rights we hold in relation to this matter. Organisations such as the DPC prevent bullying from business giants and withhold the power from these corporations.

I feel like the over emphasis of metrics can be counterproductive. More and more we are seeing team-based projects that are of laissez-faire implementation. Research has shown job satisfaction equates to a better motivated and more productive workforce. Therefore, it seems only intuitive that companies employing harsh monitorisation will in turn induce unnecessary stress and resent. An example[[20]](#endnote-20) of failed autocratic leadership style was Howell Raines time in charge of hugely popular New York Times. His callousness and dismissive nature led to distress and dissention among staff. Morale fell and so too did the quality and quantity of product. He got fired 21 months after attaining the position.

I hold the opinion that most current software metrics and computational platforms do aid the performance of software companies. A healthy pressure has always done well in providing an incentive to work. Clarity within an organisation, is too, vital towards its success. Computational platforms like the aforementioned above provide this with increasing accuracy. When employees are over-monitored, they become disincentivised and the company ultimately suffers. When a company abuses their responsibilities with holding data, there are organisations like the Data Protection Commission to counter this. There is a balance associated with everything and the measuring of software engineering is no different to this.

# Conclusion

This aim of this report was to give a brief insight into the measuring of software engineering. A variety of metrics were explored each showcasing very different approaches. We mentioned that metrics need to be increasingly thorough and specific in order to yield sufficient accuracy and that even with this, practitioners must asset the cost of these metrics and estimate whether the they are useful enough to justify these costs. Some examples were provided of popular code reviewing platforms and insightful algorithms alike. Finally, I expressed my satisfaction with the current software metrics in place though maintained that abuse of metrics can be catastrophic ethically as well productively.

# Bibliography

1. *IEEE Standard Glossary of Software Engineering Terminology*, [IEEE](https://en.wikipedia.org/wiki/IEEE) std 610.12-1990, 1990 [↑](#endnote-ref-1)
2. <https://www.tutorialspoint.com/software_quality_management/software_quality_management_basics_of_measurement.htm> [↑](#endnote-ref-2)
3. <https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th> [↑](#endnote-ref-3)
4. P. Kulik and M. Haas, Software Metrics Best Practices 2003, tech. report, Accelera Research, 2003. [↑](#endnote-ref-4)
5. <https://www.gitprime.com/> [↑](#endnote-ref-5)
6. <https://en.everybodywiki.com/GitPrime> [↑](#endnote-ref-6)
7. <https://www.producthunt.com/posts/gitprime/reviews> [↑](#endnote-ref-7)
8. <https://en.wikipedia.org/wiki/Collaborator_(software)> [↑](#endnote-ref-8)
9. <https://smartbear.com/product/collaborator/overview/> [↑](#endnote-ref-9)
10. <https://smartbear.com/product/collaborator/features/custom-review-templates/> [↑](#endnote-ref-10)
11. <https://secure.phabricator.com/book/phabricator/article/installation_guide/> [↑](#endnote-ref-11)
12. Halstead, Maurice H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*0-444-00205-7*](https://en.wikipedia.org/wiki/Special:BookSources/0-444-00205-7). [↑](#endnote-ref-12)
13. <https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm> [↑](#endnote-ref-13)
14. <https://www.merriam-webster.com/dictionary/artificial%20intelligence> [↑](#endnote-ref-14)
15. <https://expertsystem.com/machine-learning-definition/> [↑](#endnote-ref-15)
16. <https://www.citizensinformation.ie/en/government_in_ireland/national_government/standards_and_accountability/data_protection.html> [↑](#endnote-ref-16)
17. <http://gilc.org/privacy/survey/intro.html> [↑](#endnote-ref-17)
18. <https://rightsinfo.org/the-right-to-privacy-and-why-it-matters/> [↑](#endnote-ref-18)
19. <https://www.dataprotection.ie/en/pre-gdpr/case-studies#201802> [↑](#endnote-ref-19)
20. <https://www.brighthubpm.com/resource-management/77233-examples-of-companies-with-autocratic-leadership/> [↑](#endnote-ref-20)