

**CSU33012 Software Engineering**

**Measuring Software Engineering Report**

**Ross Donnelly, Std# 19335153**

September 14, 2021

# Contents

1. **Introduction** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1**
2. **Measurement techniques** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1**
3. **Analytical Practices** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **1**
4. **Computational Methods** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1**
5. **Ethical & Moral Concerns** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **1**

# 1 Introduction

Software engineering has seen a shift from craftsmanship to industrialism on a fundamental level that also includes the organization of the complete development process.  The metrics we use to measure our engineering prowess must shift in tandem.  Quality and productivity are indeed the two most important input parameters for controlling any industrial process. Successful control requires some means of measurement.

# 2 Measurement Techniques

The measures used are as varied for each company as the data attained.  However as they all strive toward similar goals in the long run, many of the fundamentals of these metrics remain the same.  Abreu & Carapuça (et. al 1994) lay out 7 criteria for desirable metrics.

1. Metrics determination should be well defined.  Obviously the data obtained must be reliable, clear and plentiful in order to attain a proper result.
2. Non-size metrics should be system size independent.  Essentially, subjective or ineffable metrics must be backed up properly.  This is particularly difficult and led to the early days of software engineering metrics concerning file sizes and lines of code.
3. Metrics should be dimensionless or expressed in some consistent unit system. In order for cross-team and cross business functionality and at the very least clarity these metrics must be identifiable and consistent throughout their life cycle.
4. Metrics should be obtainable early in the life cycle.  The cost of recovering from errors can be exponential through a product's development and metrics ability to identify these errors buried in the design can be monumental.  IN order for them to take full effect these metrics must be attainable early in development in order to eliminate errors sooner rather than later.
5. Metrics must be down-scalable. Whilst it will become clear through this report that team-based metrics are of the utmost importance, proper metrics must be able to be scaled down to unveil the more minute details to both acknowledge individual performance  and reveal errors detailed in the previous criterion.
6. Metrics should be easily computable.  Whilst the benefit of these metrics should be abundantly clear.  Provided that criterion 1 is met, the implementation of these metrics becomes more abstract and less and less detail is given.  This is in part due to the cost of not only developing these metrics but continually using them, reflecting upon them and adjusting course in response to the results. Thus it is paramount that these metrics are relatively easy to attain.
7. Lastly, metrics should be language independent.   Understanding across a business should be a baseline for these metrics.  A background knowledge shouldn’t be hugely necessary and therefore proper metrics raise themselves above a syntactic level.

It is a heavy burden for a metric to attempt to atone to all of these commandments.  Many companies strive towards documenting and evaluating each employee's personal software processes.  PSPs are a long and arduous process and whilst tools such as Timeular have created a gamification element amongst their employees leading to improved personal and collaborative productivity, few have yet to be satisfied with their level of efficiency.  This has led to more extreme examples in the case of companies using CCTV footage or products like SmartBadge coupled with AI to paint a picture of not only their employees productivity but their mood, personal habits and social circle.  The ethical argument of these metrics will come into question later in section 4.

However three companies stand ahead above the rest when it comes to measuring software engineering; code climate, WayDev and plural sight.  Whilst it may seem an insurmountable task to integrate with a company providing useful metrics and a cycle to “align, deliver and improve”, these companies certainly come close.  These companies exemplify the aforementioned criteria for a functional software engineering metric.

The actual hard measurements used by these companies

The following is an insight into the metrics utilised by Waydev.  Waydev’s software development metrics are categorised into three sections; Security or Test Metrics, Production or Operational Metrics and their core focus Agile Process Metrics.

1. **Agile Process Metrics**
   1. Lead Time

A measure of time between task creation and work completion.  It provides a perspective for the client before their requested feature is available.  It may identify issues along the feature supply chain and through historical data a developer can smooth any potential friction once the developer embarks.

* 1. Cycle Time

Similarly, measures the time spent on starting and completing a task giving an idea of a company’s organizational velocity.  It is compiled through examination of the entire development process: coding, pickup, review and deployment.  Waydev uses data points such as time to issue from first commit, time to first review, time to merge from first review, and time to deploy from merge.

* 1. Team Focus

An overview of what work is being done in each team and how these efforts are spent.   Barriers to efficiency and code blockers are identified and promptly addressed.

* 1. Project Timeline

Details the most actionable metrics in code-based activity.  Allows company leaders to see what and how their team is working in a given timeframe.  This is indicated through plenty of simple datapoints, commits per active day, total commits, total impact and less relevant code volume.  These processes should not be broken down to numbers and cents but studied as to whether they align with the period's goals and expectations.

* 1. Code Review

Code review can be an interesting metric to examine the collaborative efforts of a team after a sprint.  Code review, unlike other methodologies such as agile scrums, can allow engineers to iterate more efficiently whilst perhaps slower in the short term as bottlenecks disturbing the development process are uncovered early on.

1. **Production or Operational Metrics**
   1. Active Days

Rather simply the days spent actively on a project which can assign accountability relatively easily.

1. Work Log

Using the same data from a developers Github as before, WayDev manages to demonstrate a developers workflow in a clear and understandable way in order to signal what might be holding back a team's success.

1. Code Churn
   1. A representation of when and how often engineers will rewrite their committed code.  Whilst a certain level of churn may be expected.  Unusual and repeated patterns of this behaviour may lead to inefficiency down the line or skewed specs between a code review and an end-users experience.
2. Productive Throughput
   1. Represents the proportion of code without churn, code with proper forethought.
3. Efficiency
   1. It could be argued that all of these measures show some level of efficiency but WayDev’s measure shows a hard indication of how productive and balanced a developers code is, signalling its longevity and reliability.
4. Impact
   1. This metric begins to step away from outdated datapoints depending on size which any developer will tell you is not a proper perspective on someone’s code.  Impact speaks toward the complexity of output into a product.  It instantiates the cognitive value engineers are capable of providing and the positive change their work has made through the dev cycle.  This can be attained though a developers total edits, work coverage, severity of their changes and how these changes affected other features amongst the product.  Impact is a particularly relevant measure, as WayDev put it “translating engineers’ output into business value and cognitive load{“
5. Mean Time Between Failures
   1. This is a measure of the length of time from a products deployment to the occurrence of a repairable error in a developers code.  The longer this time, normally, the more reliable the code was in the first place.
6. Mean time to Recover
   1. Furthermore, this metric encapsulates the average time taken to recover a product or system failure.  From error to fully operations,  a MTTR can be useful when assessing the stability of a team.
7. **Security or Test Metrics**
   1. Code Coverage
      1. This type of metric determines the percentage of lines of code successfully validated under a test procedure. This pairs nicely with other metrics to interpret a coder's reliability.
   2. Commit Risk
      1. This risk metric measures the likelihood that a particular commit will cause issues.  Risk being minimal leads to higher quality, stable code for end-users.  This is attained through the size of commits, the spread of the changes and the depth of the changes.

There’s a significant overlap amongst the aforementioned companies with WayDev’s processes and these are by no means every form of software engineering measurement.  Although, these methods need to be adjusted to each client depending on goals, company size and other factors their main practice remains relatively the same;  to integrate into a company, make use of their data and present in such a way that any agile company or developer could take the full advantage and become their best most efficient version of themselves.

**A note on teamwork analysis**

Software engineering involves teamwork and communication of many kinds. Specific examples include: eXtreme Programming, pair programming both of which require very close collaboration focussed on the same artifact. In CSE, the pair members need not be spatially co-located. Development activities such as analysis, design, testing and coding may be carried out by different combinations of individuals. CSE-mediated discussions are potentially a valuable way for effective communication and feedback between and within these groups.  With the increase in remote practice we can begin to use data to analyse this collaboration.  In the office this was replicated with companies like SmartBadge but now using analytics from companies like slack and zoom we can excavate physical data from a collaborative effort.

# 3 Analytical Practices

Evaluating large scale software development requires the use of quantitative models attained through proper data analysis.  These models provide insight and support to the development team and a potential client through historical data from the current and similar projects.  This has given rise to various approaches such as investigative methodology, software engineering validation and optimized set reduction  However the core intentions remain the same.

1. To collect relevant, reliable and relatively extensive data from a source
2. To build models of the software process, product, and other forms of experience based upon common characteristics of the purpose of prediction.
3. The recognition and quantification of these influential factors on various issues of interest for the purpose of understanding and monitoring development
4. To evaluate the resultant products and processes from different perspectives by comparing them to projects with similar characteristics,.
5. To identify the strengths and weaknesses of the current work environment
6. To understand what we can and cannot predict and control so we can monitor it more carefully.

Thankfully (for most), in this day and age there are frameworks in place to monitor and aid in these intentions.

**To collect relevant, reliable and relatively extensive data from a source**

For a software engineer there’s no better place for data interrogation than a user's code repository.  A surplus of data is available from an engineer's bitbucket or more commonly their Github.  So much so that these are requirements on par with a resume and cover letter when applying for jobs in the industry.  An engineers Github allows data based on code volume, time spent, commit risk, quality and impact of code to be collected.  This is of course under the assumption that all coding collaboration takes place on Github and is available to those retrieving the data.  More often than not, however, this information is freely and extensively available through the Github API.  Whilst at times this data can be inaccurate especially in the case of code churn, it’s relevance and reliability has allowed it to become the industry standard.

Less commonly, employees under review will detail personal software processes in order for an employer to measure their performance but these can be cumbersome and often biased.  It is unrealistic to have an examiner standing over the shoulder of each and every developer measuring their performance and thus the hard data obtained from their repos should be used.

**To build models of the software process, product, and other forms of experience based upon common characteristics with the purpose of prediction.**

This process is often a collaborative, ‘whiteboard’ experience and has become harder and harder amongst the developer workforce with the rise of remote working however these processes also including;  Risk Management, Software Quality Assurance (SQA), Software Configuration Management (SCM), Measurement, Formal Technical Reviews (FTR) are often a collaborative experience with normally top management and their staff as well as third part companies such as WayDev and flow.   Whilst much of this development stems from personal experiences and is one of the core roles of a VC in an early stage start-up within software engineering, much of it can also be computed analytically with the data in hand.  Companies make use of the aforementioned engineering metrics and use software visualisation tools such as React or d3 in order to make clear to their client their processes and prowess.  These visualization linked to the relevant data can then be analysed (which will be covered in more detail in section 3) to give a prediction of the workflow and life cycle of the client’s development.  This prediction is estimated through a series of formulas as well as experience but primarily through AI which can chart potential inaccuracies through the life cycle and lay them out for a given future project much like an investor would project their portfolio.

**The recognition and quantification of these influential factors on various issues of interest for the purpose of understanding and monitoring development**

Visualisation tools like those mentioned play a greater role in this task.  The reason being that proper visualization can make clear to anyone their role in the development.  If it appears that a developers engineering performance is in a downtick or worse if their performance is curtailing the metrics of the entire development then their instinct will be to improve.  Similarly a client can track the progress of their potential asset.  However this is only possible through proper understanding and development of visualisations of development.

**To evaluate the resultant products and processes from different perspectives by comparing them to projects with similar characteristics.**

Again this can be done effectively through visualizations but a company may decide to take a different approach.  This is often at a product launch when a company can use these metrics in order to gain introspection, rewarding innovation and smoothing bumps.  This takes the form of far more in person collaboration through surveying and questionnaires from various perspectives.  However from a data perspective the company may align their performance along various projects and embark upon creative destruction keeping what works and fixing or throwing out the rest.

**To identify the strengths and weaknesses of the current work environment**

This ties in strongly with the last point, all companies desire to identify their pros and cons, not only software development companies.  Companies such as Equalture use their platform to observe at first but then allow the strengths and weaknesses of a team to peak their head before visualizing them for proper understanding and using an industry benchmark to find the best next steps.

**To understand what we can and cannot predict and control so we can monitor it more carefully.**

Obviously the purpose of this data and its subsequent development and analysis to create a better overall performance tomorrow.  Further visualizations and the use of machine learning practices should manage to summarize both the past and future performance and clue the management team on how to maximise their performance.

# 4 Computational Methods

The difficulty of software engineering metrics is their relevance.  It could be argued that without proper goals and team management these metrics mean nothing.  Someone may run faster than Usain Bolt, but if they’re running the wrong direction, what does it matter.   Proper computation of the data mines is vital for the engineering metrics to blossom their full potential within the company.   These processes can then be altered and used to motivate an engineer or team toward company goals; team integration, value maximisation employee engagement and risk minimisation

Simply, the core challenges to success include both the size (millions of lines of code, thousands of classes) and complexity of the software under development.  Quality metrics must be utilised to avoid the three main pitfalls of engineering monitoring.

1. Focusing too much on individual metrics can lead to a streetlight effect.  Revering the few stallions of a team is useless.  Like all collaborations a software engineering is only as strong as its weakest member.  The metrics we use must reflect this
2. Attaching metrics to rewards or incentives can lead to a sort of agency problem.  Teams begin to think in a carrot-stick way.  Yearning for their incentive instead of achieving what should be the overarching goal of all these teams i.e. breeding innovation and striving for efficiency.
3. Using good metrics in the wrong way.  This most often occurs due to a breakdown of communication between the software engineering team and another faction of the business, organisation etc.  For instance agile velocity works well in an agile development sprint and may be used to exemplify the hard work of a software team over a month or quarter.  But these metrics can give a skewed representation when interpreted by different people, perhaps not with a skill base in software development.

In response to these problems computational methods come into play.  I spoke briefly about data acquisition earlier through the industry standard, i.e. employees repositories.  However despite being the industry standard there are plenty of other practices that should be of relevance when examining a firm's software engineering performance.  Many companies continue to merely use statistical analysis.  However the extent of programs such as excel, MATLAB and Plotly can solve and align future plans with the given data especially in a quantitative sense falls short.  However many frameworks have been developed that provide both statistical, computational and quantitative analysis of software engineering performance.

1. **PluralSight Flow**

I detailed the measurements efforts of the company WaydeV.  Similarly PluralSight Flow (formerly GitPrime) provides companies with an opportunity to truly enhance themselves through performance metrics.  They claim to advocate for the work your engineers are doing and help remove roadblocks so they can focus on what they really love to do: solve problems.   By computing and administering the relevant performance metrics and aligning with a firm’s goals and objectives Flow can push a company to new heights.

1. **Velocity 2.0**

Flow unfortunately partially falls to the streetlight effect.  Velocity 2.0 manages to solve this issue.  Similarly they analyse an employees Github account as the bottom line of their data.  However velocity focuses on processing constraints within individual departments.  They give careful consideration to visualizing their raw engineering data to appeal and inform all perspectives, regardless of their background.

1. Gitalytics

Gitalytics manages to take this a step further by providing an even more comprehensive demonstration of Github analytics.  This however does reduce the effect their visualisations can have due to their simplicity and whilst the computation of Pluralsight may cause an adaptive company to shift and realign their objectives Gitalytics is normally used in a more demonstrative way.

Increasingly however these companies have noticed a rise in the relevance of algorithmic approaches and AI and machine learning grow in might and capabilities.  If AI can be used for image recognition and coupled with machine learning to ensure proper social distancing or flagging anti-social behaviour in our day to day, then why not utilise it in the analysis of our developers.  There are three core ML training methods to be put to use.

1. Supervised Learning

These algorithms learn from the past data that is inputted, called training data, runs its analysis and uses this analysis to predict future events of any new data within the known classifications.  Whilst a significant of data is necessary for a level of accuracy, supervised learning shows the potential for a company’s prowess to go far or for bottlenecks to become bloodbaths.

1. Unsupervised Learning

This method concerns providing the algorithm with a large amount of unlabelled data to discover the correlations between them.  This has been put to use in many advertising firms but may in the future inform a company of professional decisions when provided with engineering performance data.

1. Reinforcement Learning

Similarly this algorithm is provided with only unlabelled data.  However it is provided with a form of trial and error, each data point has a relative positive or negative impact.  This has a significant impact on measuring engineering performance as the algorithm merely solves the problem in the best way it sees fit without any previous input.

# 5 Ethical and Moral Concerns

Ethical concerns in the tech industry are a controversial topic to this day.  Despite the history of all other industries display of workers rights over moral and ethical concerns never has there been seemingly greater cause for alarms for data security and privacy.  There has yet to be a societal consensus on what is right due to the infancy of the industry.

Moral Issues

Two major concepts have been gaining traction in the field of data science ethics; informed consent and data ownership.

Informed Consent stems from the medical research world and boils down to the rule that subjects need to know and agree to be studied. This may seem obvious but it does not seem clear or perhaps it does and remains irrelevant to Facebook.  Facebook was manipulating the newsfeeds of users in a study in conjunction with Cornell University and the University of California-San Francisco on emotional contagion based on reaction to primarily positive or negative posts on the topic.  Data Ownership gets called into question when we ask whether the developer owns their performance metrics or their firm.  This may not seem as relevant to analysis of a company's own employees as naturally they consent under the employ of the firm.  However, in my research I was obviously only able to break down the metrics a firm discloses.  A company may use metrics that it knows may be morally or ethically grey and refuses to disclose them publicly.  Whether it is concerns over the methods or depth to which they analyse their employees.  The technology, especially AI, is in place to predict every aspect of our lives and why shouldn’t a company use this to their advantage; adjusting health plans, lining up promotions, or establishing incentives. However this is arguably an overstep morally and a company would not disclose that publicly to the dismay of both their employees and shareholders. This then becomes a legal issue, not only for the employees who may have private data sold or utilised against but also for the company as data legislation slowly edges its way to the courtroom. Not to get too foreboding, but a continued fostering of this systemic disregard for the importance of our data could reign in the singularity.  With the proper data analysis we are morally charged with a classic trolley problem; On one track we have a team of developers, whose efficiency is lacking according to their data with ambiguous consensus and ownership.  On the other track we have peak performance of an AI developer capable of replicating it’s human counterparts work with peak performance.  This moral dilemma will not go away anytime soon, and data privacy and analytical morality are merely one piece of the puzzle.

**Verdict**

The question at hand is: is it wrong to analyse software engineers as they go about their work.  In my humble opinion, I believe with proper management it should be perfectly acceptable.  What should be under scrutiny, I believe, is the astronomical targets, pressure and conditions these engineers are put under.  The industrialisation of software development has brought us many things.  It has made daily innovation the norm.  It has made the writing of papers all the more difficult as many publishers find themselves made irrelevant by the next leap forward before reaching peer review. It has also built a society that expects incredible technological advancements in every crevice of their daily lives and for the perfect functionality of this technology.  This ideal is far more harmful than any concerns I believe over data analysis.  All workers are scrutinised by analytical management.  Granted the wealth of data provided by a developer is perhaps more telling than the amount of coal brought in by a miner but the issue stands.  In a capitalist, product/performance driven, toxic work environment supported by a lack of unionisation and ethical accountability of management, a developer can experience a crunch upon their workload which ripples out into all aspects of their life.  Software engineering can unearth this problem, however it may also lead a company to double down leading to further problems down the road.  It is therefore our choice to use the measurement of software engineering to our advantage and ingratiate ourselves firmly as part of the solution.

# References

[1] Carl Cook and Neville Churcher. 2005. Modelling and measuring Collaborative Software Engineering. In Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38 (ACSC '05). Australian Computer Society, Inc., AUS, 267–276.

[2] Kaner, C., & Bond, W.P. (2004). Software Engineering Metrics: What Do They Measure and How Do We Know?

[3] Brito e Abreu, Fernando. (1997). Object-Oriented Software Engineering: Measuring and Controlling the Development Process.

[4] pluralsight.knowledgeowl.com. (n.d.). Get your organization started using Pluralsight Flow | Pluralsight Help Center. [online] Available at: https://help.pluralsight.com/help/get-your-organization-started-using-pluralsight-flow [Accessed 4 Jan. 2022].

[5] Waydev. (2021). Software Metrics Leaders Can Measure to Track the Development Process. [online] Available at: https://waydev.co/software-development-metrics/ [Accessed 4 Jan. 2022].

‌[6] Basili, Victor R. and David M. Weiss. “A Methodology for Collecting Valid Software Engineering Data.” *IEEE Transactions on Software Engineering* SE-10 (1984): 728-738.

[7] Lim, Y. (2019). Doing Data the Right Way. [online] Medium. Available at: https://towardsdatascience.com/doing-data-the-right-way-dbf5ba658177.

‌

‌