Software measurement report

To deliver a report that 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 ethics concerns surrounding this kind of analytics.

Dervla Brennan
17325863

1. Introduction

Software engineering can be defined as “systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software" (IEEE, 2019). Due to its broad definition and complex nature, the process of software engineering is notorious for being inherently difficult to measure. However, as technology rapidly continues to play a greater role in the world of business, measuring the software engineering process has become a monumental task faced by managers across companies spanning all backgrounds, industries and sizes.

This issue is not unique to software engineering. We exist in an era of information, where organisational success is heavily reliant upon highly educated and skilled ‘knowledge workers’ (including accountants, surgeons and indeed software engineers), whose value is linked to what they know. For these kinds of workers, there are far more factors that must be taken into consideration than simply quantity or speed of output as quality and other ‘subjective’ metrics become of greater importance. For instance, a medical surgeon performing a small number of successful surgeries in a day is arguably more valuable and skilled than a surgeon who conducts many unsuccessfully within the same time frame. Similarly, a software engineer who is constantly producing a huge amount of output that is ridden with bugs is likely to be of less value than a developer who contributes less but performs more rigorous testing.

That being said, from a management perspective, measuring worker productivity can provide a number of helpful insights and opportunities, including but not limited to: tracking the evolution of new team members, improving performance and maintaining employee motivation. In today’s hypercompetitive business environment, management must continually seek out new and creative ways to get ahead. The familiar phrase, “What gets measured, gets managed,” infers that we cannot improve things we are not measuring.

In this report, I aim to provide a general overview of the different kinds of measurable data that can be used to measure and assess the software engineering process in particular. I will also consider the computational platforms and algorithmic approaches available to perform this work. Finally, I will discuss the ethical concerns surrounding the measurement and analysis of software engineering as a practice.

2. Measurable data

Measurable data is of huge interest to firms seeking to measure and assess productivity in that it provides quantifiable and objective information about employee performance which can aid planning and inform decisions about process improvement. There are a wide variety of measurable types of data that can be used to assess the productivity of a software engineer, of varying levels of meaningfulness and accuracy, which will be explored later in this report.

Although the below list is not exhaustive, I aim to provide a general overview of the different ways the software engineering process can and is being measured quantifiably in workplaces today under the following headings:

- Activity
- Time
- Failure/Success

2.1 Activity

Lines of code (LOC):
Physical LOC is simply a count of how many lines of code that have been contributed to a program. It is an easy to understand metric that is accessible to managers regardless of technical knowledge. It was one of the first software metrics ever used as a basis for programmer productivity measurement and was thought to be a key factor in the effort and cost of developing software. LOC can be further broken down into logical lines of code which attempts to improve the meaningfulness of LOC as a metric by eliminating the counting of comment lines and blank spaces. Logical LOC instead counts the number of executable expressions written (e.g. functions, operators, etc). However, this is not without flaw and varies hugely depending on the programming language utilised (Stackify, 2017).

Commits:

In a version control system, such as git, a commit describes an uploaded or 'committed' change made to a file (or set of files) (Github, 2019). Monitoring commits is another way that activity in the process of software engineering is commonly measured. This could involve simply counting the frequency of commits or the consistency of commits over time of developers within a repository, which may be indicative of how instrumental individuals are in that project. This could also extend to deeper analysis of the commit including code churn, i.e. the amount of deleted/added lines within a commit.

2.2 Time

Time is an important metric in software measurement as a demonstration of efficiency and productivity. There are several different times involved in the software engineering process that can be measured, including:

Working Hours:

One way of measuring the time involved in the software engineering process is simply the hours that developers work. If monitored, this metric could give insight into the connection between worker fatigue and productivity.

Leadtime:

Leadtime is a measure of how long it takes to go from original idea to delivered software.

Cycle time:

Cycle time measures how long it takes to make a change to your software system and successfully deliver that change into production.

Measuring the lengths of time projects and elements of a project take can help to make future estimates more accurate and can also indicate which developers are more skilled and responsive.

2.3 Failure/Success

Ideally, the software we build would never fail, but practically speaking, that is highly improbable. Conversely, instead of measuring failure, the success of software can also be measured quantifiably. Hence, various aspects of failure/success can be used to measure the performance of software engineers, including:

Mean time between failures (MTBF)

MTBF describes the average amount of time that elapses between inherent failures of a system.

Automated code coverage tests

Grades can be automatically generated based on the coverage that unit tests provide to the source code.

Business success metrics

Satisfying the customer is the ultimate goal when it comes to software engineering as a service. It is crucial to ensure that customer feedback e.g. ratings/number of downloads, is measured and taken into account so as to check that what developers are producing is in line with what the customer wanted.

3. Computational platforms

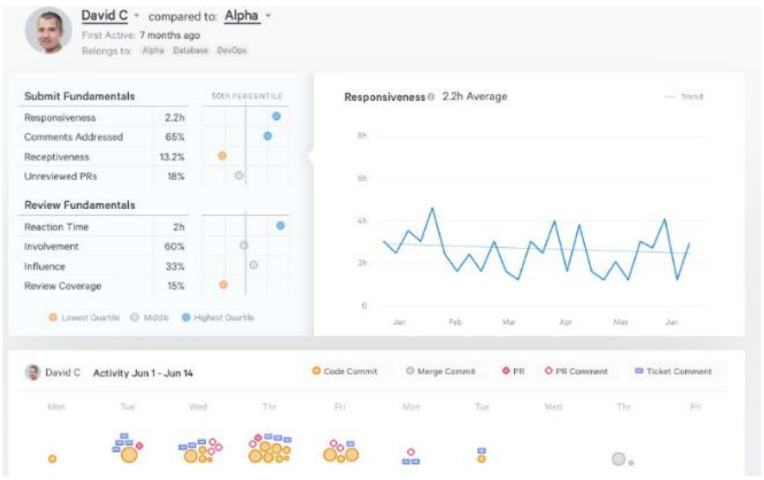
In order to assess the software engineering process, it is not enough to merely collect measurable data types that were outlined in the previous section. Once this information has been gathered, it has to be processed and organised in order to make it usable. There are several services currently available that provide computational platforms to organisations seeking to assess the software engineering process, some of which are proprietary tools and others which are open source. These applications have been created to help organisations extract value from large amounts of data without having to build the necessary software to do so in house.

GitHub

GitHub is an open-source web-based version control system that is owned by Microsoft. As of August 2019, GitHub hosts over 100 million repositories, 40 million developers and 2.1 million businesses worldwide (GitHub, 2019). GitHub provides the functionalities and features of git, along with extra services of its own including bug tracking and wikis. Moreover, GitHub provides an API that allows anyone to extract data relating to aspects of the software engineering process such as commits, contributors and pull requests from all publicly available repositories which can then be utilised to analyse activity.

GitPrime

GitPrime is a popular computational platform that has the ability to mine and aggregate data from any Git based code repository, ticketing system and pull request and transform this into insightful reports (GitPrime, 2019). This platform is made accessible to non-technical managers by presenting the data in a meaningful way with a focus on key metrics instead of the more complex software work. Figure 1 below is a snippet of an example that demonstrates how GitPrime allows managers to visualise an individual's key metrics and progress over time using a 'Player Card':



David C compared to: Alpha

First Active: 7 months ago

Belongs to: Alpha Database DevOps

Submit Fundamentals	
Responsiveness	2.2h
Comments Addressed	65%
Receptiveness	13.2%
Unreviewed PRs	18%

Review Fundamentals	
Reaction Time	2h
Involvement	60%
Influence	33%
Review Coverage	15%

Responsiveness 2.2h Average

David C Activity Jun 1 - Jun 14

Figure 1 (McGrath, 2019).

Velocity

CodeClimate is a company that aims to enable data-driven engineering through their computational platform, Velocity. Velocity works in a similar way to GitPrime in that it mines data from repositories and presents the findings in a more user-friendly format. While GitPrime focuses mainly on individual's metrics, Velocity places more of an emphasis on the overall project/team. Velocity allows the flexibility of custom reporting, giving users the ability to build charts that are more valuable and appropriate for them. Also, this framework provides features to allow management to put their learnings into action. Upon growing familiar with how a team performs over certain periods of time, contributor-wide, team-wide, or organisation-wide targets can be set to encourage improvements in prioritised criteria (Rezvina, 2019).

Hackystat

Hackystat is an open source framework that was designed by academics in the University of Hawaii for the “collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data” (Hackystat, 2019). The framework focuses on ‘unobtrusive data collection’ and works by attaching sensors to developer’s tools, such as an IDE, and passing that raw data to a server to interpret it at regular intervals. Hackystat moved from a client-server architecture to a service-oriented architecture (SOA) in 2007. One major advantage that came with this move was that the ability to process the HTTP networking protocol became the only constraint on build language choice. The HTTP networking protocol is provided for in all modern programming languages, which therefore allows users of the framework more flexibility in the creation of native client-user interfaces that can interface to Hackystat’s web-based services and develop more innovative 3D visualisation tools (Johnson et al., 2009).

4. Algorithmic approaches

Once data has been collected and processed successfully, the next stage involves interpreting and analysing this data to give it meaning and value. In the case of assessing the software engineering process, this will commonly involve some sort of algorithmic approach to characterise performance. The following list briefly discusses a handful of the vast number of approaches that may be taken, including:

Process Quality Index

The Process Quality Index (PQI) is a simplistic algorithmic approach to quality in the engineering process. It consists of five elements that are normalised to a ratio between 0-1 and are multiplied together to return a figure that indicates the quality of the software product, with a result of 0.5 and lower indicating poorer quality:

- Design Quality – design time : coding time
- Design Review Quality – design review time : design time
- Code Review Quality – code review time : coding time
- Code Quality – compile defect : size
- Program Quality – unit test defects : size

(Pomeroy-Huff et al., 2009)

Halstead's Complexity

Maurice Howard Halstead introduced his method of using a composite metric to determine aspects of software complexity. It works by initially extracting the following four parameters:

- n1 – the amount of distinct operators
- n2 – the amount of distinct operands
- N1 – count of the occurrence of operators
- N2 – count of the occurrence of operands

Formulas can then be constructed using these parameters to determine a number of different metrics, including:

Metric		Formula
Vocabulary	(n)	n1+n2
Size	(N)	N1+N2
Volume	(V)	Nlog2n
Difficulty	(D)	(n1/2) * (N2/n2)
Effort	(E)	D*V
Error	(B)	V/3000
Testing time	(T)	E/18 seconds

This algorithmic approach is straightforward and can be helpful to indicate whether a project is in line with forecasted predictions. However, these numbers alone do not tell the full story of the software engineering process as this approach neglects to consider certain aspects such as branching or jumps that can make for a more complicated program (Tashtoush et al., 2014).

Cyclomatic complexity

Cyclomatic complexity refers to a popular algorithmic approach that aims to determine the complexity of the software engineering process. The cyclomatic complexity approach calculates the number of linearly independent paths in a program using a control flow graph (CFG) such as the graph representing Insertion Sort in Figure 2 below:

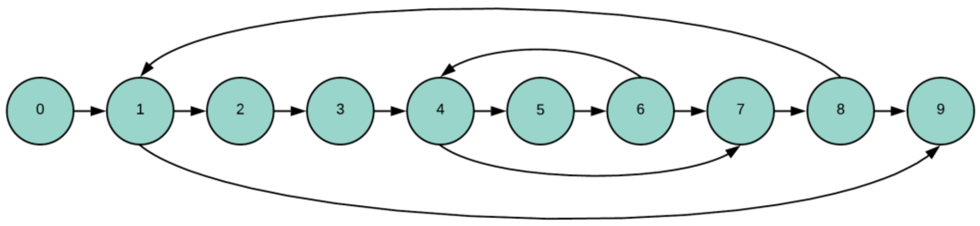


Figure 2.

For a control flow graph, *G*, the cyclomatic complexity, *V*, can be calculated using the formula: *V(G) = E – N + 2P*, where *E* and *N* represent the number of edges and number of nodes in the CFG respectively and *P* represents connected components. This approach to evaluating the software engineering process returns a single number that indicates the complexity of the program and implies how many test cases would be required to adequately cover all possible paths through the code. It is generally better to have a lower number as it has been found that the likelihood of errors increases with greater cyclomatic complexity (Tashtoush et al., 2014).

Bayesian Belief Networks

Bayesian Belief Networks (BBNs) are a probabilistic algorithmic approach, based upon classifications of Bayes' Theorem, used to estimate the quality of the software engineering process. Unlike Halstead's or cyclomatic complexity approaches, which rely on imperfect size and complexity metrics, BBNs takes into consideration more diverse factors implicit in defect prevention, detection and complexity. This approach allows for the creation of causal models based on existing simple metrics. One major advantage of using the BBN approach is the explicit modelling of uncertainty and ignorance in its estimates (Fenton and Neil, 2000).

5. Ethical concerns

There are several major ethical concerns surrounding the measurement and assessment of the software engineering process. As was discussed in the first section of this report, there is a wide range of data relating to the software engineering process that can be measured. The following two sections focused on the various existing methods of collecting and interpreting that data. The question is not about whether or not we can measure the software engineering process but instead becomes whether or not we should when there exists debate about whether such analysis can deliver meaningful results.

Inaccurate/Demotivational
Assessing employee worth based solely on numerical data like statistics, trends and algorithms can be extremely harmful to a company's culture. The nuanced nature of the software engineering process makes it ethically questionable to apply the same metrics and analytics across individuals, teams or projects in that doing so may inaccurately and unfairly reflect the work involved and could lead to unintended consequences. For instance, an organisation emphasising productivity metrics that focus on volume of code and errors produced, may result in developers who contribute extensive amounts of simplistic code may end up with a great productivity score but may not necessarily have great technical skills when it comes to software development. Also, consequently, engineers could ultimately try to avoid more difficult problems that arise in order to maintain high LOC and low error counts.

Numerical information may also fail to take into account soft skills such as teamwork, creativity or communication skills that are vitally important to achieving successful results. Of course, these characteristics are more difficult to measure due to their subjective nature, however, if ignored, may leave employees feeling taken for granted, under-appreciated and demotivated. Moreover, the act of overly intrusive monitoring can erode at the intangible feelings of trust and fairness in an employee-employer relationship, especially as the lines of acceptable forms of surveillance are pushed further and further out (Burt, 2018). Employee retention is of high concern in modern organisations and not granting engineers autonomy over their work can lead to driving good employees and potential future managers out of the business.

Privacy/Legality concerns

Measuring employee productivity is by no means a new concept and has been around since the turn of the 20th century when Taylor (1911) proposed his theory of scientific management to reduce worker inefficiency and improve productivity during the Industrial Revolution. Over one hundred years on, the world is a drastically different place, however the core principle has endured. Monitoring now abounds in modern society through technologies like email and browser history, CCTV and GPS. Workplace surveillance has also advanced far beyond sign-in sheets to include far more invasive metrics. Tools to facilitate surveillance in the workplace are a fast-growing market. For example, 'Humanyse' is a company that provides biometric staff ID badges. Real-time tone of voice analysis can be conducted through microphones and motion detectors can chart movements around the office (Burt, 2018).

However, the general public has grown far more aware of how data is collected and monitored in recent years, helped by conversations started by big news events like Snowden's NSA whistleblowing and the Facebook/Cambridge Analytica scandal. Public opposition to privacy invasion also led to the introduction of regulation like GDPR to protect people's right to privacy. Under GDPR, companies within the EU will have to ensure that employee consent about data collection should be freely given, informed and revocable. When measuring and assessing the software engineering process, management should be aware that a trade-off that exists when carrying out this kind of analysis, between obtaining rich insights and impeding upon employees' rights to privacy.

6. Conclusion

Overall, as has been explored throughout this report, measuring and assessing the process of software engineering is unquestionably difficult and complex. However, this is not to say that it is unnecessary and of no benefit. Indeed, I believe this is what makes it an even more vital practice.

Measurement of workers' productivity can be invaluable and result in improvements not only at an organisational level but also for the software engineering industry as a whole. Measurement to some degree is required in order to facilitate noticeable advancement. Regardless of whether people are comfortable with the idea of being monitored or not, it is a part of modern business that does not seem to be going anywhere soon. Practically every single industry in the 21st century now requires at least some element of software and this number is ever-increasing which means that software engineers are inevitably going to have to get used to the idea of being measured and assessed.

Furthermore, metrics are not an inherently bad thing, however, as discussed previously, when implemented inappropriately, they can be. It is this very issue that calls for further development of more holistic metrics that represent the software engineering process in a more accurate way. While metrics should remain logical, data-driven and uninfluenced by biases, exercising context and common sense is important. Numbers cannot tell the full story, not yet anyway.

By continuing research into determining what data sources, gathering techniques and algorithmic assessment combinations adequately reflect the qualitative and quantitative toils of software engineers that satisfy management and developers alike, all will benefit.

7. Bibliography

- Burt, E. (2018). *Has employee monitoring gone too far?* People Management. [online] Available at: https://www.peoplemanagement.co.uk/long-reads/articles/employee-monitoring-gone-too-far [Accessed 8 Nov. 2019].
- Fenton, N. and Neil, M. (2000). *Software Metrics: Roadmap*. Proceedings of the Conference on the Future of Software Engineering.
- GitHub. (2019). *Build software better, together*. [online] Available at: https://github.com/ [Accessed 3 Nov. 2019].
- Gitprime. (2019). [online] Available at: https://www.gitprime.com/ [Accessed 3 Nov. 2019].
- IEEE (2017). Systems and software engineering — Vocabulary. ISO/IEC/IEEE 24765:2017(en).
- Johnson, P., Zhang, S. and Senin, P. (2009). Experiences with Hackystat as a service-oriented architecture. *Tech Report Department of Information and Computer Sciences*, *University of Hawaii* [online]. Available at: https://www.researchgate.net/publication/228731372_Experiences_with_hackystat_as_a_service-oriented_architecture [Accessed 3 Nov. 2019].
- McGrath, J. (2019). Visualize core individual metrics with the Player Card. [Blog] *GitPrime Blog*. Available at: https://blog.gitprime.com/core-engineering-metrics-player-card/ [Accessed 3 Nov. 2019].
- Pomeroy-Huff, M., Cannon, R., Chick, T., Mullaney, J. and Nichols, W. (2009). *The Personal Software Process*. Body of Knowledge, Version 2.0. [online] Carnegie Mellon University: Software Engineering Institute. Available at: https://www.researchgate.net/publication/235116544_The_Personal_Software_ProcessSM_PSPSM_Body_of_Knowledge_Version_20 [Accessed 4 Nov. 2019].
- Rezvina, S. (2019). *Velocity vs. GitPrime: Choosing an Engineering Intelligence Tool*. [online] Code Climate. Available at: https://codeclimate.com/blog/velocity-vs-gitprime/ [Accessed 3 Nov. 2019].
- Stackify. (2017). *What are Software Metrics? Examples & Best Practices*. [online] Stackify. Available at: https://stackify.com/track-software-metrics/ [Accessed 30 Oct. 2019].
- Tashtoush, Y., Al-Maolegi, M. and Arkok, B. (2014). The Correlation among Software Complexity Metrics with Case Study. *International Journal of Advanced Computer Research*, 4(15).
- Taylor, F. W. (1911). *The principles of scientific management*. New York: Harper & Brothers.