Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Module: CSU33013
Date: 03/01/22
Name: Darren Aragones

Measuring Software Engineering

1. Introduction

How to deliver value is not obvious within the context of software engineering. One cannot measure software engineering productivity by lines of code per hour, or completed programs per month.

Software engineers epitomize the idea of *knowledge workers,* where workers solely own the means of production. The concept of productivity has evolved from Taylorism or manual-worker productivity, where absolute quantitative measures and motion analysis could drive improvements in efficiency (Drucker, 1999). This type of analysis is inappropriate for software engineers and induces less than favourable practice; 'gaming' the metrics, among other questions of suitability and privacy.

The contemporary issue of measuring software engineering is that knowledge workers must be measured *qualitatively*, as value is not always delivered *quantitatively.* The relationship between quality and the numbers generated by common software engineering activities is ambiguous, quality itself being the extent to which value is delivered. Measurement as a discipline is vital for communication and accountability in a team environment.

This report serves as a discussion of the current state of measurement on software engineering activities. The theoretical and ethical nature of metrics and measurement is then expanded upon to inform and justify potential improvements in the measurement processes that regard software engineers.

2. How to Measure

2.1 - Quantitative Measures

Numbers for engineering measurement can be relatively straightforward *(direct)* or a complex derivation (*indirect)*. Number of **commits** or the **lines of code** are simple numerical measures.

The **cyclomatic complexity metric** measures the number of linearly independent paths in a code section. It is useful for testing code coverage and

Defect/bug counts are surrogate measures for engineer skill and quality of testing. Bug count per unit time (i.e. per week) may include all bugs found in a week, or all open/remaining and indicates an engineer's progress on a project.

There exists a large number of metrics that can be derived from simple quantitative measures.

The amount of time an engineer spends on a section of code leads to a number of essential development metrics. Managers define cycles and ‘tickets’ based on the state of a project or issue, leading to a measure of **cycle time**: time taken to “cycle” from a state to another. The statistics for a specific cycle type or status can be aggregated to help understand a team’s speed. **Velocity** alternatively measures the number of features (value-added work) completed within a period of time. Lastly, **throughput** expresses the total amount of work completed within a time period. Throughput tickets include bugs, chores and tasks as well as features completed and are used to understand workload compositions. Similar metrics based around the concept of time include **mean time to recover, mean time to failure** and **lead time.**

2.2 - Qualitative Measures

Qualitative measures for the work of software engineers span **question rubrics** and design concepts that allow a degree of subjectivity.
The creation of qualitative metrics cannot be automated. Engineers must report and address these metrics on their own basis.

Questions used by managers on such rubrics and quality checklists include;

- ➢ Is the code readable?
- ➢ Is the code extensible?
- ➢ Is the code easily maintained or refactored?
- ➢ Does regularly updated documentation accompany?

Technical debt is a metaphorical measure of the extra work incurred by design choices. Engineers can decide to release a quick and easy design instead of a potentially better design. However it is nearly certain that worse code must be refactored, which is ‘debt’ repaid (Ernst et al., 2015).

It is a design concept that cannot be translated empirically. Engineers garner ‘interest’ on their debt which consists of extra refactoring. The time and difficulty of excess refactoring cannot be expressed strictly quantitatively.

2.3 - Construct Validity

Quality can be defined in terms of attributes, and the possession of key attributes allows the interpretation of quality (IEEE Standard 1061). The purpose of metrics is to output numerical values from data to identify such attributes.

Therefore *construct validity* exists to question how metrics map to attributes. Software engineering attributes are seldom simplistic. Productivity cannot be measured through the real number of commits nor the lines of code pushed out by a developer.

Standard 1061 distinguishes between **direct** metric functions and **derived** functions. The direct metric of mean time to failure assesses software reliability as an attribute of quality, but suffers from uncertainties of how time or failure are defined (Kaner, 2004). A fundamental misunderstanding in the use of metrics is that they fail to become relevant if defined from the possible set of operations and not the desired context. In other words, we try to fit measurable operations to attributes instead of the opposite.

2.4 - Improving Measurement

Estimations and Expectation

Computer science educator Frederick Brooks remarked that techniques for estimating often confuse effort with progress and poorly monitor such progress. While techniques certainly have developed, the fallacies of effort and manpower within estimation remain.

Problems arise from ‘gaming’ of measurements by some employed engineers. It is possible that these issues lie within the scope of estimations by any one metric. Net improvements could be

found in grounding expectations within the actual capabilities of an individual or team.

A solution is to ask teams or individuals to set personal quantitative goals such as *time to open,* or *turnaround time.* Developers can measure against their goals and managers are able to understand the aggregate level of productivity, and gain insight into the social dynamics of the team[s]. Aggregating the numbers can instead become a measurement of the team leader's management skills and not productivity. Complex behavioural relationships can be found between the establishment of metrics and how they are measured.

Framework for Metric Evaluation
The questions that engineers ask about metrics can become more comprehensive, using a theoretical framework.

Kaner and Bond set 10 questions to evaluate proposed metrics, including novel ideas;

- *Natural scale of ambiguous attributes?* What scale makes sense for measuring engineering skill?
- *Natural variability of an attribute?* Understanding degrees of variation between objects/attributes.
- *How is the attribute we want to measure, related to the metric value?* The question of construct validity.
- *What are the side effects of measuring?* Measuring creates superficial incentives punishing engineers that take their time to solve problems, or supporting other engineers.

2.5 - Other Measurements

Correlations between engineering productivity have been explored with employee wellbeing and co-worker productivity. The company Hitachi conducted a study using wearable sensors to find patterns of physical activity. The intuition behind this study was to show there is a correlation between **health** [as a proxy for happiness] and productivity (Yano et al., 2015).

Its authors found a high statistical correlation between happiness, concentration, good sleep and appetite was found with better sales in a call centre. The authors of this study claim that happiness metrics possess applications for improving engineering productivity and quality. This type of measurement has been theorised to be particularly suited to creative industries (Shawn, 2012).

Furthermore studies on **peer effects** show increases in productivity metrics as a result of experimental changes to worker collaboration, in which the beneficial outcome is observed in increased interactions. Negative effects may even be observed from deprivation of an impactful social contact (Lindquist et al., 2015). There is a rising justification to study "*people analytics*", a recent term encompassing metrics that focus on the interplay between co-workers (Leonardi, 2018).

3. Infrastructure

Infrastructural tools for software engineers that display graphical information are sometimes known as *dashboards* within industry. Certain services specialise in automated code review, but the scope of measurement extends to data gathered from general task and project management software. Services can be distinguished by their usability by individuals or groups, and what attribute it intends to describe with metrics.

3.1 - Automated Code Review

Useful and efficient code review tools perform automatic analyses on a repository input. *Static analysis* reasons about the run-time properties of code without actually executing anything.

Codacy is a code analysis for Git platforms drawing data from commits and pull requests. It was created as a plugin for Github and Bitbucket and enforces a user-generated standard of code quality. **Code Climate** offers a *Quality* product that provides its own automated assessment of technical debt and test coverage, and is integrated as a browser extension that connects to GitHub. **CodeGrip** is a web service that lists and suggests solutions for code issues, and provides graphs for reliability and maintainability. Its selling point is in code security, where the tool will never store any code. **Jira** is a dashboard for project workflows that has integration with Bitbucket to automatically update its lists of tasks upon a commit, and is focused on the Agile and DevOps methodologies.

3.2 - Non-Specific Infrastructure

Non-specific tools that do not analyze any code or commits can still provide useful information for assessing productivity.

Trello is a visual tool that organizes projects as 'cards' on a virtual board. **Flow** and **Slack** are task and project management software that offer features for enumerating and graphing tasks. These services can be integrated with the Harvest browser extension **Harvest** to track time spent on various tasks using timers. From a qualitative perspective they can also be used to identify causes and effects of coding-related issues.

Physical *peripherals* provided by companies such as Steelcase or Fitbit engage in employee health monitoring. Office equipment such as chairs or badges can be augmented with sensors and microphones that generate data based on voice, location or posture.

4. Computations

4.1 - Halstead Complexity Measures

The introduction of the Halstead complexity measures by Maurice Halstead in 1977 was made on the observation that software-centric metrics should be platform independent.

Formulas of volume, difficulty and effort are built from operators and operands:

- Counts of operators η1
- Total operator number N_1
- Counts of operands η2
- Total operand number N_2

The formulas attempt to capture natural properties of software similar to properties of matter. The Halstead metrics are used to describe maintainability and will interpret large portions of code as intrinsically complicated. Interestingly, an estimate for the time required to program (T) is included in these measures:

$$T = \frac{E}{18} seconds$$

Where $effort(E) = D \times V$, difficulty (D) and volume (V) are arithmetic operations on operator and operand values. A highly empirical value has been assigned to the attributes of *difficulty* and *effort.* These attributes are derived only from the coding done by engineers and exclude important work in requirements engineering, which limits the scope of Halstead metrics to code complexity.

4.2 - Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) automating statistical modeling. Machine learning is used in the software engineering process to identify bugs, bug causes and risk prediction.
Development of ML integration into the software engineering process has been partially driven

by the need for greater support for ML development itself (Amershi et al., 2019). Uncertainty and component entanglement in data-driven learning algorithms imposes a new level of complexity.

Productivity and risk suggestions implemented in the systems discussed in [Section 3.1] for code review use ML techniques.

4.3 - Case-Based Reasoning

Analogy or case-based reasoning (CBR) algorithms reason about the potential cost of software. The aim is to "reuse experience" (Keung et al., 2008). Software engineers' previous work in CBR is broken down into mathematical components to reason for future projects.

Existing algorithms for CBR include ESTOR and Analogy-X. They serve as alternatives to linear regression as existing project data can be used as the 'analogy' which has more intuitive value for team management. Whether a CBR technique is useful or not cannot be proven on a statistical basis unlike regression analysis; selection of analogy code is largely human-based, although the selection can be automated too.

CBR algorithms assume that the presence of project features implies a certain level of effort. This level of effort remains constant between analogous projects.

5. Ethics and Measurement

5.1 - What is ethics?

The Institute of Electrical and Electronics Engineers (IEEE) lists eight principles in its Code of Ethics that software engineers should adhere to, relevantly including the aspiration to ethical Management.

"5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance."
[Preamble- Software Engineering Code of Ethics]

The IEEE Code of Ethics concerns itself mainly with [1] the public interest, and [2] treating human beings with *due respect*. The side effects of managing using measurement on engineers should be considered an ethical matter.

Firstly, engineers must consider how the public, "*if reasonably well informed*" would view their actions and act accordingly. Measurement systems can create incentives to 'massage the numbers' (*distortion*) or cause engineers to provide less value than without measuring (*dysfunction*). Inadequate measurement affects the adequacy of the product and consequently harms the relationship between stakeholders, including the public and environment.

Secondly, ethics addresses the values and principles held by individuals. Methodologies employed to measure software engineers can impact their values positively or negatively. Dignity and respect are common human values that come into question when measuring. There is a set of normative pressures to enshrine such values; derived from the virtue of one's humanity, but the boundaries of ethicality are undefined and this leads to disagreement between engineers and managers.

5.2 - Distortion and Dysfunction

Creating a culture where teams use metrics to reflect on their everyday decisions is not straightforward. Properties found through static analysis tools are generally *undecidable.* It is impossible in theory to determine the existence of runtime problems or uncaught exceptions, but to simply estimate them. There is little

determinism in the art of measuring software engineering.

For instance, aligning programmers with management is difficult without metrics for developer productivity. Alignment in a corporate sense is the question of whether the individuals in a team are best working towards the business' [or client's], goals (Trevor et al. 2016). A method is needed to express achievements and verify them. Brooks remarks that "one rarely controls the circumstances of his work, or even its goal" in reference to developers. *Other people* set the objective and engineers must be capable of interfacing with them.

Values considerations for software engineering may also concern the end-product of software development as much as the impact on engineers themselves. Facebook's "*social by default*" graph API at the centre of the Cambridge Analytica scandal is often seen as a result of engineers' ethical standards, or the lack thereof.

Externalizing social consequences through impactful use of measurements is an imperative for modern software engineering (Winter et al. 2018). There is a category of **Professional ethics** that applies to the technical professions. Engineers' claim to exclusive knowledge is justified by an assumed obligation to serve society (Bayles, 1981).

The 1999 IEEE code enshrines the social consequence of this professional commitment;

"6.08. Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work."

Ethical advocates note that software engineers owe their primary obligation to society rather than to a client. Technical decisions lead to radically different products with effects beyond satisfying requirements. Professional engineers should be aware of the many technical solutions to problems, and metrics allow elucidation of problem sets.

Gotterbarn illustrates an example of applying ethics considerations to the end-product;

"*Consider designing a journal file for a library check-out system to determine the popularity of books and the number of additional copies that should be ordered, if any. The association of the patron's name with the book checked out has a potential for the violation of several moral rules, such as the violation of privacy and the deprivation of pleasure because one does not feel free to read what one wants, and possibly causing psychological pain."*

Code complexity in what are possibly life-critical systems according to Gotterbarn could put people at risk. Violating system design principles is perhaps synonymous with ethical violations. Measures of code complexity have been discussed in this report [Section 2.1] and assist in mitigating design issues.

5.3 - Measurement and Values

An example of a human value is dignity. Dignity is an innate worth or status that is not earned and cannot be forfeited. Moral philosophical principles like the Categorical Imperative ask humans to treat each other as an end in itself, and not simply a means (Hill et al., 2014).

Privacy is an individual value challenged by the development of technology. The anonymity of data collected by workplace analytics has been challenged by *content privacy* studies (Yu, 2016). Information that is anonymized or encrypted can still identify an individual.

A lack of privacy can lead to discrimination, which is an unethical behaviour. One definition of discrimination is "*the process by which a member, or members, of a socially defined group is, or are, treated differently (especially*

unfairly) because of his/her/their membership of that group" (Krieger, 1999). The consequence that software systems diminish users' sense of personal moral agency and shift the accountability to the computer has also been studied (Friedman, 1997).

Commentators in the legal field remark there is little lawful protection against data analytics revealing very sensitive information; such as the employees likely to take parental leave, employees likely under stress at home or employees more likely to leave (Areheart, 2019). A classic example is of the insurance company that denies credit to members of a neighbourhood if there is a correlation with that area and ethnic minorities.

Considerations for the dataset generated from software engineering should be informed through an ethical standpoint regarding values. The field of study on ethical workplace analytics is not academically rigorous yet (Hullmann, 2020), but some companies have made attempts to externalise negative social effects. The Humanyze badge product integrates ethics in data collection, collecting only aggregated metadata to ensure companies cannot see individual statistics.

6. Conclusion

This report has aimed to establish and explain the inherent non-determinism of metrics to any attribute that developers or managers try to measure. This non-determinism is linked to the misuse and ambiguity of metrics to assess software engineers.

The actual use of metrics is context-specific. It can be dependent on team size, experience and company purpose. Contemporary misuse of metrics comes from misunderstanding the number-attribute association. For instance, superficiality; assigning quality to bug counts or lines of code. The hypothetical scenario being if a company aims to release infrequent but robust products then low bug counts will become idealistic. However this may not matter to a company with frequent and numerous releases, and this quality assumption fails to hold.

The existence of the Hawthorne effect and gaming are human effects. In combination with the disincentives posed by dashboard mandates, measurement advocates have understandably warned against individual measurements. Software engineers are knowledge workers and quality assessments of this category of workers should not become rigid and mechanistic.

It can be concluded that measurement in software engineering should be done solely for improving engineer skill sets, or reactively to investigate issues (Grady, 1987). A study of 20 developers using static analysis tools suggests that their underuse comes from result understandability, weak support for team collaboration and bad outputs. The participants expressed more negative opinions of their tools in every single aspect of a tool; its output, team support, customizability, understandability and workflow integration (Johnson et al., 2013).

Furthermore, a global online survey conducted by SmartBear Software over more than 740 software developers shows a strong correlation between peer code reviewing and improved software quality. About 24% of participants automated with static analysis, but peer reviews are just as prevalent at 25% (SmartBear Software, 2020). Sharing knowledge is about coaching and not direct mandates. It is the only correct way to treat software engineers.

References

Lindquist, M.J., Sauermann, J. and Zenou, Y., 2015. Network effects on worker productivity.

Winter, E., Forshaw, S. and Ferrario, M.A., 2018, October. Measuring human values in software engineering. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-4).

Yano, K., Akitomi, T., Ara, K., Watanabe, J., Tsuji, S., Sato, N., Hayakawa, M. and Moriwaki, N., 2015. Measuring happiness using wearable technology. Hitachi Review, 64(8), pp.97-104.

Achor, S., 2012. Positive intelligence. Harvard Business Review, 90(1), pp.100-102.

Lindquist, M.J., Sauermann, J. and Zenou, Y., 2015. Network effects on worker productivity.

Leonardi, P. and Contractor, N., 2018. Better people analytics. Harvard Business Review, 96(6), pp.70-81.

Drucker, P.F., 1999. Knowledge-worker productivity: The biggest challenge. California management review, 41(2), pp.79-94.

Brooks, F.P., 1974. The mythical man-month. Datamation, 20(12), pp.44-52.

Kaner, C., 2004. Software engineering metrics: What do they measure and how do we know?. In In METRICS 2004. IEEE CS.

Emanuelsson, P. and Nilsson, U., 2008. A comparative study of industrial static analysis tools. Electronic notes in theoretical computer science, 217, pp.5-21.

Ernst, Neil A., Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Ian Gorton. "Measure it? manage it? ignore it? software practitioners and technical debt." In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 50-60. 2015.

Trevor, T., Varcoe, B., 2016. A Simple Way To Test Your Company's Strategic Alignment. Harvard Business Review

Hill, T. E. (2014) "Kantian perspectives on the rational basis of human dignity," in Düwell, M., Braarvig, J., Brownsword, R., and Mieth, D. (eds) The Cambridge Handbook of Human Dignity: Interdisciplinary Perspectives. Cambridge: Cambridge University Press, pp. 215–221. doi: 10.1017/CBO9780511979033.027.

Yu, S., 2016. Big privacy: Challenges and opportunities of privacy study in the age of big data. IEEE access, 4, pp.2751-2763.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B. and Zimmermann, T., 2019, May. Software engineering for machine learning: A case study. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 291-300). IEEE.

Grady, R.B. and Caswell, D.L., 1987. Software metrics: establishing a company-wide program. Prentice-Hall, Inc..

Areheart, B.A. and Roberts, J.L., 2018. Gina, big data, and the future of employee privacy. Yale LJ, 128, p.710.

Ebrahimi, S., Ghasemaghaei, M. and Hassanein, K., 2016. Understanding the role of data analytics in driving discriminatory managerial decisions.

Gotterbarn, D., 2001. Software engineering ethics. Encyclopedia of Software Engineering, 2.

Lister, K. and Analytics, G.W., 2014. What's Good for People?. Moving from wellness to well-being.

J. W. Keung, B. A. Kitchenham and D. R. Jeffery, "Analogy-X: Providing Statistical Inference to Analogy-Based Software Cost Estimation," in IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 471-484, July-Aug. 2008, doi: 10.1109/TSE.2008.34.

Smartbear "The State of Code Review 2020", 2020.

Johnson, B., Song, Y., Murphy-Hill, E. and Bowdidge, R., 2013, May. Why don't software developers use static analysis tools to find bugs?. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 672-681). IEEE.

Hüllmann, J.A. and Mattern, J.A.N.A., 2020. Three Issues with the State of People and Workplace Analytics. Proceedings of the 33rd Bled eConference, pp.1-14.

Weller, C., 2016. Employees at a dozen Fortune 500 companies wear digital badges that watch and listen to their every move. INSIDER Magazine.