

Measuring Software Engineering

A matter of trade-offs

Stephen Moran

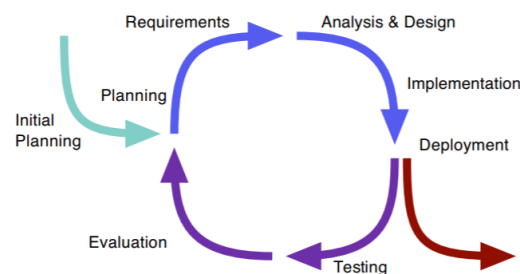
16319039

Table of Contents

Introduction.....	1
Background.....	1
Measurable Data.....	2
Computational Platforms.....	4
Algorithmic Approaches.....	7
Ethics.....	9
Conclusion.....	10
Conclusion.....	11

Introduction

In this report, I will focus on measuring software engineering. Software engineering is a detailed study of engineering to the design, development and maintenance of software (Fig 1). Software engineering was introduced to address issues of low-quality software projects (The Economic Times, 2018). Software engineering usually follows a process that outlines the set of tasks to be completed. The development is usually performed through cycles. Each completed cycle or iteration usually results in added features or functionality from the previous version (Passos, et al., 2011). Software measures are tools that allow us to measure the quality of software (Zuse, 1995).



(Fig 1) Software Engineering Cycle (Passos, et al., 2011)

This report describes the background and history of measuring engineering and how we came to where we are today. It then focuses on the methods and approaches to measuring engineering before looking at some of the ethical impacts this practice may have.

Background

The area of software engineering measurement has been researched for more than thirty years and is known as software metrics. As computers have become more ingrained into our lifestyles there is an increased importance on software measurement in order to enforce quality.

Despite the increase in software engineering activity and research, software projects have become very complex and software projects are hard to successfully complete. In fact, 75% of business and IT executives anticipate their projects will fail and fewer than a third of all projects in 2016 were successfully completed on time and on budget (Medium, 2017). This highlights that there is a lot of room for improvement in software development. In order to improve the development process and consider software development an engineering process it's believed

that rigorous scientific procedures must be applied to the discipline with measurement techniques at the heart of this (Zuse, 1995).

In his article “No Silver Bullet- Essence and accident in software engineering”, Frederick P. Brooks believes that there has been no single development which by itself promises even one order of magnitude improvement in software engineering practices. However, when concluding he explains that our best way to develop is to focus on great designers. These designers are usually produce structures that are faster, smaller, simpler, cleaner and produced with less effort (Frederick P. Brooks, 1987).

I agree with Brooks and believe that improving the way in which developers think about, design and implement software is the key to great improvement. A way of inspiring these improvement is by measuring the progress of developers. As mentioned by analysing it's developer's activities a company can derive a basis for estimates, track project progress, determine complexity, assess quality, analyse defects and validate best practices (Zuse, 1995). However, this practice is far from straightforward and there must be careful considerations made before delving in.

Measurable data:

There are many different types of data that one can use to produce metrics to measure the software engineering process. A company must determine the suitability of the available data for the metrics they are trying to compute before including it in their analysis. These metrics should be easy to compute, consistent, easy to obtain and relevant to the development of high-quality software product (Stackify, 2017).

Line of code (LOC):

Lines of code is a simple metric that is used to indicate the complexity of programs of the efficiency of the software. Dating back to the 1960's' this was one of the first metrics that was used to determine both programmer productivity and software quality. LOC is usually expressed as KLOC, short for thousands of lines of code (Fenton, 1999).

This metric can be used to enforce different company coding standards e.g. limiting the lines allowed in a function to increase readability. LOC will not give those conducting analysis a great insight into the true project productivity or code efficiency for a number of reasons. For example, if comparing against another similar project, something as small as a difference in

coding standards or programming language used would ruin the analysis. For the same reasons we can again see its flaws when comparing two programmers with this metric. When using any of these size metrics it is worth considering what to actually count as a LOC, should we count comments, blank lines etc? However, as managers pick and choose what to include comparability becomes even harder.

Bugs:

There has been many efforts to measure the bugs in a software system and therefore, the system quality. As mentioned before many have used the approach of counting the number of bugs within a system or per lines of code etc. However, these efforts have been shown to be useless in many instances. Vlad Giverts is the Head of Engineering for Clara Lending offers an alternative solution of using bugs as a measurement. He suggests looking at the amount of time actually spent fixing the bugs. By doing this it's possible to see if a disproportionate amount of time is being spent on fixing bugs and it could signal a quality/ architectural problem (Stackify, 2017).

Test Coverage:

Test coverage is a popular metric used to determine how well a software system has been tested. It looks at the number of lines tested by the test suite in relation to the total number of line of code (Fig 2). This metric is very popular in industry and works off the logic that the higher the code coverage, the better the code has been tested.

$$\text{Code Coverage} = \frac{\text{Number of lines of code called by test suite}}{\text{Total number of lines}} \times 100$$

(Fig 2) Code Coverage Formula

Many argue that a high code coverage is not a sufficient means of testing as all it implies is that the code has been called by the test suite once however, it does not indicate what exactly was asserted. Others argue, that while you can test all your code, it's very likely that the value of your tests will diminish as you approach the 100% limit, this may give developers the tendency to write more meaningless tests for the sake of satisfying the coverage requirement (Clement, 2014).

Function orientated metrics

This is intended to be a measure of the systems functionality. This is difficult to directly measure so we can calculate the function point (FP). The FP is a unit of measurement that quantifies the business functionality of the system (Stackify, 2017). This involves counting the number of inputs and output, user interactions, external interfaces, files used. Then the complexity of each is assessed and multiplied by a weighting factor before its used to predict size, cost or project productivity. Many find this a tedious process so many managers stay completely way from function points (Stackify, 2017).

Happiness:

In 2015 researchers found that happy workers had 37% higher work productivity and 300% higher creativity and therefore, recently there has been research on how we can track and improve worker's happiness. While measuring this qualitative term has seemed an impossible task in the past, now it could be possible with advancements in technology, in particular wearable technology. Although still a very difficult task, researchers were able to relate happiness to physical activity and then to some degree capture an individual's happiness using wearable technology. As this technology improves, we may see it introduced by those measuring software engineering in the future (Yano, et al., 2015).

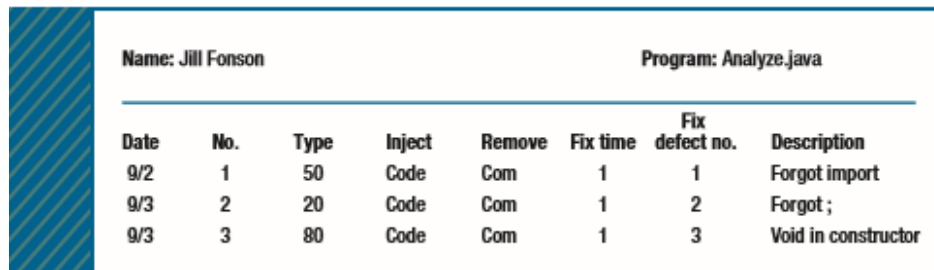
Computational Platforms:

PSP - Personal Software process:

PSP is a rigorous methodology used to monitor software development (Sillitti, et al., 2003). It uses simple spreadsheets, manual data collection, and manual analysis. In one version of the PSP developers must fill out 12 forms including: a project plan summary, time-recording log, defect-recording log (Fig 3), process improvement proposal, size estimation template, time estimation template, design checklist and a code checklist (Johnson, 2013).

Information collected is analysed statistically and this continuous monitoring helps developer to trace their performances, compare them to the schedule, and find out whether specific factors (i.e. environment, behaviours, interactions, etc.) affect them (Sillitti, et al., 2003). Consider a developer who suspects that the number of interruptions he or she experiences each morning directly impacts productivity. The PSP provides explicit encouragement to explore this analytic and make an evidence-based conclusion.

However, the drawbacks of this approach soon became apparent. It wasn't news that collecting and managing this data takes substantial effort and that there are many overheads for developers. However, the real issue was that although the manual nature makes PSP's analytics flexible, the manual nature of the PSP sometimes led to incorrect process conclusions despite a low overall error rate. This leads to significant data quality problems (Johnson, 2013).



Name: Jill Fonson				Program: Analyze.java			
Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

(Fig 3) Sample defect recording log (Johnson, 2013)

LEAP:

LEAP is a process measurement toolkit developed to try solve some of the data quality issues found with PSP. This toolkit automates the data analysis reducing overhead for developers. The developer still manually enters the data the toolkit automates the PSP analysis. This toolkit replaces enhances PSP by improving data quality and decreasing the manual analysis required. However, despite making some analysis easy to conduct, it also makes others increasingly difficult. Using the example of the hypothetical developer again, who suspects that there are factors affecting their productivity. They would now be expected to know how to design and implement a new LEAP tool-kit rather than a simple spreadsheet. It was soon realised that PSP would never be a fully automated approach and would require manual data entry (Johnson, 2013).

Hackystat

Hackystat was the third generation PSP data collection system developed by the University of Hawaii (Sillitti, et al., 2003). This was developed in order to automate the data collection phase and reduce overhead for developers. Hackystat achieves this automation by attaching sensors to development tools and sending the collected data to a server (Johnson, 2013).

Hackystat has 4 key design features:

1. Both client and server-side data collection
2. Unobtrusive data collection (No interruptions for developers)
3. Fine-grained data collection (Minute by minute analysis)
4. Personal and group-based development

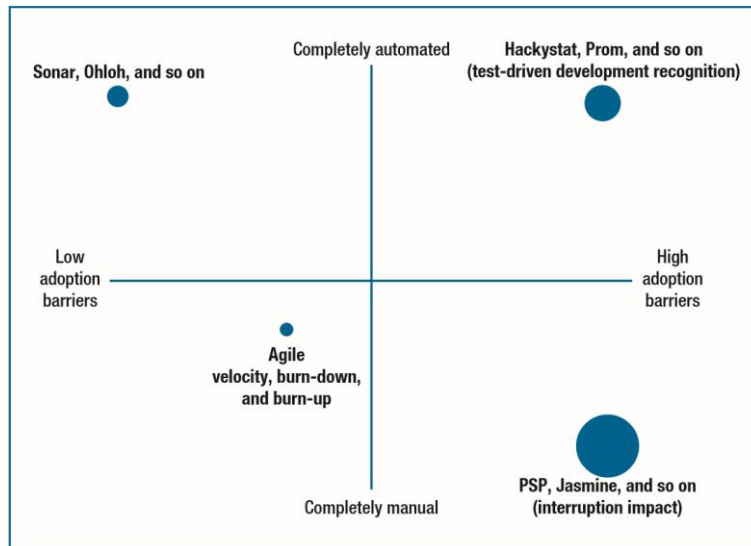
The strengths of this system have not gone unnoticed and Hackystat technology soon became part of a commercial offering. However, after more research some of its problems were uncovered. The problems found were largely centered around the ethics of this type of data collection and how many employees are unhappy with the way in which their data was being collected or used (Johnson, 2013).

PROM:

PROM is an alternative to Hackystat that manages to solve many of the privacy concerns. PROM is similar to Hackystat in that it attaches to developer tools however, the metrics and analysis produced can only be seen by the developer themselves. This allows the developer to spot their own inefficiencies and compare it to the planned schedule. However, managers are not left out of the equation, they are provided aggregated data on the project development teams and this helps them in managing the project (Sillitti, et al., 2003).

Other platforms:

Recently, the number of software packages available for this type of analysis has been on the rise. CodeClimate, DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity, are all types of software product analysis packages (Fig 4). These systems typically consist of a configuration management system, a build system, and a defect-tracking system (Johnson, 2013). Most of these systems have modern user-friendly dashboards. They provide detailed analysis while keeping low overheads for developers with high automation. These services avoid controversy by focusing on product characteristics and very little on developers' behaviours (Johnson, 2013).



(Fig 4) – Computation platform comparison (Johnson, 2013)

Algorithmic approaches:

Algorithmic and predictive approaches have been used in favour of basic measures. The aim is to produce models of the software development and testing process which take account of the crucial concepts missing from traditional statistical approaches. Models need to be able to handle diverse process and product evidence, genuine cause and effect relationships, uncertainty and incomplete information while at the same time not dramatically increasing overheads (Fenton, 1999).

Machine Learning Algorithms:

“Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention” (SAS, 2018).

Coder Productivity:

An example of machine learning used to in measuring engineering is seen with Semmle Inc who produced a report called “Measuring software development productivity: a machine learning approach”. In this report they explain their approach to computing coder productivity using machine learning (Helie, et al., 2018).

One approach to measuring productivity is by breaking it down into code output and code quality. To measure coding time models are created that create datasets that associate code

changes with coding times. After being trained, the model can predict the coding time required to produce a code change. This model would then give a view of the benchmark coder.

Code quality is hard to define and is in many ways subjective to whoever sanctioned the project. For example, a project low in errors may still be full of unnecessary complexity and therefore, difficult and costly to maintain. Quality for the productivity metric is calculated using LGTM, which is a service that queries code and produces ‘alerts’ based on potential errors. The queries have been developed over several years by language experts and encode domain knowledge about code quality. LGTM puts the alerts in different categories. For each of these categories a non-linear quantile regression is fit that predicts the alert distribution for this project with the given LOC. A model is then used to give a quality score for each category:

$$\{0 = \text{worst}, 0.5 = \text{as expected}, 1 = \text{best}\}$$

The final quality score is the weighted average of all categories. Weighting is determined by the severity of the alert.

Bayesian belief nets (BBNs):

Bayes’ theorem describes the probability of a test result based on prior knowledge of conditions that might be related to the result (Wong, 2017). BBN is a graphical network associated with a set of probability tables used to predict software defects (Fig 5). This works off the concept of Bayesian probability. This same solution is used in areas such as medical diagnosis and is behind the help wizards in Microsoft Office. Each node represents an uncertain variable and the arcs represent the relationships between the variables. The probability tables for each node provide the probabilities of each state of the variable for that node. The graph contains a mix of variables that we are interested in predicting. The BBN can give the probabilities of certain outcomes given certain evidence. For example, when we enter evidence about the number of defects found in testing all the probabilities are updated. This will always compute the probability of every state regardless of the amount of evidence provided. Lack of evidence is reflected with a greater uncertainty of values. This method’s success is reliant on the stability and maturity of the development process. Organisation must collect the basic metrics data and carry out systematic testing to be able to apply this model effectively (Fenton, 1999).



(Fig 5) - Bayesian belief net (Fenton, 1999)

Ethics

“At its simplest, ethics is a system of moral principles” (BBC, 2014). Privacy and ethics are key considerations when conducting analysis of software engineers. Today consumers, markets and employees are becoming increasingly intolerable to unethical practices and will punish companies in one way or another for any practices deemed unacceptable. The key trade-off when conducting analysis on the software engineering process is between obtaining rich and meaningful information and impeding on developers’ rights to privacy.

In the paper “Searching under the Streetlight for Useful Software Analytics”, Phillip M. Johnson believes that no silver bullet will be yielded from future research without social and political implications (Johnson, 2013). The challenge for employers and researchers is being able to find the optimal balance between the two, information that inspires improvement while also providing transparency to employees and making ethical considerations.

Many people may feel uncomfortable by the fact that they are being constantly monitored. A study at John Hopkins University determined that people perform tasks better while being watched and that the parts of the brain associated with social awkwardness and reward invigorate another part of the brain that controls motor skills, improving their performance. However, it was also noted that those with social anxiety performed better and therefore, it could be argued that this performance is largely down to fear (Knapton, 2018). Putting someone in a constant anxious/paranoid state for the whole day may be very unpleasant and exhausting for the worker and this must be taken into account when choosing a suitable system for observation.

In his article “Software Metrics: Successes, Failures and New Directions” Fenton gives a rather pessimistic outlook on the motivations behind software measurement, he says “For all the warm feelings associated with the objectives for software metrics (improved assessment and prediction, quality control and assurance, etc.) it is not at all obvious that metrics are inevitably a ‘good thing’” and describes the use of measurement programs as a grudge purchase made when things are bad or to satisfy an external body (Fenton, 1999).

We can see clearly the negative impact these systems can have when implemented without taking workers attitudes into account when we look at Hackystat. This platform was once described as “hacky-stalk” and was seen as overly intrusive by some developers who felt that their information was being collected without them being notified. Secondly, the fine-grained data approach can create tension in working groups and teams, where some developers were unimpressed with the transparency it provided regarding team members work (Johnson, 2013).

Conclusion:

Overall, it’s clear to see that there is plenty of data available to measure the software engineering process. I believe measuring the software engineering process is inherently complicated and the biggest challenge is understanding the results and limitations of the metrics and methodologies being used. Metrics are very favourable for management as it portrays a complex discipline as a set of simple figures however, it’s important that the management understand what exactly the metric is portraying and be careful not to ignore the most important factors simply because they are not easily quantified.

Approaches to this practice have evolved over time to keep up with new demands of the industry and I don’t foresee this evolution slowing down anytime soon. However, it’s apparent that we are still yet to find a ‘silver bullet’ in this area that manages to get the right balance in the different trade-offs involved with these systems, such as detailed analysis vs ethical practice. As demands increase and software engineering becomes increasingly important, companies are pushing hard to avoid some of the common mistakes found in software projects. Utilising the latest measurement tools may give companies an edge however, it’s also very important that a balance between gaining quality insights and workers privacy is struck. Failing to do so may have a greater negative impact on worker trust and productivity that far outweighs the benefits of collecting fine-grained data.

Bibliography:

- BBC, 2014. *Ethics: a general introduction*. [Online]
Available at: http://www.bbc.co.uk/ethics/introduction/intro_1.shtml
[Accessed 06 11 2018].
- Clement, T., 2014. *Are Test Coverage Metrics Overrated?*. [Online]
Available at: <https://www.thoughtworks.com/insights/blog/are-test-coverage-metrics-overrated>
[Accessed 15 11 2018].
- Fenton, N. E., 1999. Software metrics: successes, failures and new directions. *The Journal of Systems and Software*, Issue 47, pp. 149-157.
- Frederick P. Brooks, J. P. B. J., 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20(4), pp. 10-19.
- Hayes, W., 2014. *Agile Metrics: Seven Categories*. [Online]
Available at: https://insights.sei.cmu.edu/sei_blog/2014/09/agile-metrics-seven-categories.html
[Accessed 16 11 2018].
- Helie, J., Wright, I. & Ziegler, A., 2018. *Measuring software development productivity: a machine*. 2018, Semmler Inc.
- Johnson, P. M., 2013. Searching under the streetlight for useful software analytics. *IEEE Software*.
- Knapton, S., 2018. *Why people perform better when they are being watched*. [Online]
Available at: <https://www.telegraph.co.uk/science/2018/04/20/people-perform-better-watched/>
[Accessed 10 11 2018].
- Medium, 2017. *10 reasons why software development projects fail*. [Online]
Available at: <https://medium.com/specestimate/10-reasons-why-software-development-projects-fail-7200e7c9ae2e>
[Accessed 08 11 2018].
- Passos, E. B., Medeiros, D. B., Neto, P. & Clua, E., 2011. Turning Real-World Software Development into a Game. *Games and Digital Entertainment (SBGAMES)*, pp. 260-269.
- SAS, 2018. *Machine Learning: What is it and why it matters*. [Online]
Available at: https://www.sas.com/en_ie/insights/analytics/machine-learning.html
[Accessed 20 11 2018].
- Sillitti, A., Janes, A., Succì, G. & Vernazza, T., 2003. Collecting, Integrating and Analyzing Software Metrics and Personal Software. *EUROMICRO*.
- Stackify, 2017. *Development Leaders Reveal the Best Metrics for Measuring Software Development Productivity*. [Online]
Available at: <https://stackify.com/measuring-software-development-productivity/>
[Accessed 15 11 2018].
- Stackify, 2017. *What Are Software Metrics and How Can You Track Them?*. [Online]
Available at: <https://stackify.com/track-software-metrics/>
[Accessed 10 11 2018].

The Economic Times, 2018. *Software Engineering*. [Online]
Available at: <https://economictimes.indiatimes.com/definition/software-engineering>
[Accessed 06 11 2018].

Wong, W., 2017. *What's the Difference Between Machine Learning Techniques?*. [Online]
Available at: <https://www.electronicdesign.com/automotive/what-s-difference-between-machine-learning-techniques>
[Accessed 20 11 2018].

Yano, K. et al., 2015. Measuring Happiness Using Wearable Technology. *Hitachi Review*, 64(8), pp. 97-104.

Zhang, D., 2000. Applying Machine Learning Algorithms In Software Development.

Zuse, H., 1995. History of Software Measurement - Software and Systems Engineering.