

CS3012 - Software Engineering

Prof. Stephen Barrett

“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.”

Student Name:	Davy Nolan
Student No:	17330208
Date:	28/10/2019

Introduction

Is it possible to accurately measure the process of software engineering? Is it a “one size fits all” scenario where one metric can simply be applied to all fields and the accuracy will remain consistent? It is only natural to want to have quantitative data as a software engineer, it could keep developers on track and ensure harmony throughout their work as well as allowing them to predict errors before they even occur. The advantages are endless, however I return to my opening question, is it possible? As M. Fowler said, *“Many of these arguments are impossible to resolve because the software industry lacks the ability to measure some of the basic elements of the effectiveness of software development.”* (M. Fowler, 2003).

Measurement of the Software Engineering Process

As Cem Kaner put it *“Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way to characterize them according to clearly defined rules.”* (C. Kaner, W. P. Bond, 2003). Assuming that the measurement and assessment of the software engineering process is possible, how would one achieve this? The measurement and assessment of this process is also known as software metrics. A software metric is a measure of software characteristics which are quantifiable or countable. A metric is a measurement function, and software quality metric is *“a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.”* (IEEE Standard 1061 [8], 1998).

To create a variety of metrics for a project, you must write up a list of important factors for it. With each quality factor there is a connected direct metric that acts as a quantitative representation of a quality factor. A direct metric is a *“metric that does not depend upon a measure of any other*

attribute" (IEEE Standard, 1998). This means that it does not rely on any other piece of data, it is simply its own individual piece of information. For example, a direct metric would be the amount of lines of code a developer has in their project. You must identify multiple direct metrics to associate with each quality factor, "otherwise, there is no way to determine whether the factor has been achieved" (C. Caner, W. P. Bond, 2003).

There are ways that metrics must be validated in order to be legitimised. The IEEE Standard 1061 outlines six different validation criteria:

The first is "*Correlation*" which implies that the metric should be connected to the quality factor linearly as measured by the statistical correlation between the metric and the corresponding quality factor. Basically this means that correlation metrics measure whether or not there is a relationship between two variables.

Next is "*Consistency*". This means that the metrics must provide output data for the quality factor that is always either becoming more accurate and helpful or remain the same.

"*Tracking*" suggests that as a metrics function changes overtime, either from being improved or redesigned, the output data should also change promptly.

The next validation criteria is "*Predictability*". This can be described as if we know the value of the output of a metrics function at some point in time, we should be able to predict the value of the metrics function.

The IEEE Standard characterises "*Discriminative Power*" as "*a metric shall be able to discriminate between high-quality software components and low-quality software components. The set of metric values associated with the former should be significantly higher and those associated with the latter.*"

Lastly, “*Reliability*” signifies that a metric should demonstrate all the criteria; correlation, tracking, consistency, predictability and discriminative power “*for at least P% of the application of the metric*”.

The difference between a direct metric and an indirect metric is that a direct metric has a one-variable domain and an indirect metric as a multiple-variable domain. For example, speed is a function of distance and time, which has a domain of two, therefore it is an indirect metric. Examples of indirect metrics in software engineering include programmer productivity (LOC / programming time) and system spoilage (effort spent fixing faults / total project effort). According to Fenton, “*the Lines of Code measure (LOC or KLOC for thousands of lines of code) was used routinely as the basis for measuring both programmer productivity (LOC per programmer month) and program quality (defects per KLOC).*” (N.E. Fenton, M. Neil, 1999). The reliability of these metrics did not last long with the introduction of more complex program languages such as Pascal in 1970 and C in 1972 - “*After all, an LOC in an assembly language is not comparable in effort, functionality, or complexity to an LOC in a high-level language.*” (N.E. Fenton, M. Neil, 1999). Simply put, these simple standards of metrics just did not stand the test of time when it comes to software development. The software engineering process is a complex process, the act of measuring it varies from company to company and even individual software developer to developer. The metrics need to be able to adapt to fit each situation.

Since direct metrics are only based off one domain (i.e. based off one piece of data), it seems as though we can assume it to be accurate and true. However this is not always the case, direct metrics can be a lot more complex. This is mainly due to the fact that most of the time, there is human interaction with software – “*As soon as we include humans in the context of anything that we measure—and most software is designed by, constructed by, tested by, managed by, and/or used by humans—a wide array of system-affecting variables come with them. We ignore those variables at our peril. But if we take those variables into account, the values of our seemingly simple, "direct" measurements turn out to be values of a challenging, multidimensional function*” (C. Caner, W. P. Bond, 2003).

The IEEE Standard 1061 suggests that MTTF (Mean time to failure) is an example of a direct measure of reliability. Studying the work of Kaner and Bond, we can see that this measure is not direct. They use the example of “*a professional secretary and an occasional typist using a word processor*” (C. Caner, W. P. Bond, 2003). Both of these positions possess completely different “*operational profiles*” and levels of experience and skill, which causes metrics to fluctuate. There may also be a chance of the product failing more frequently for the occasional typist than for the secretary. It just not possible for there to be simply one function that can be applied to both of these users to obtain accurate metrics as it’s like comparing apples and oranges. The paper then continues to explain the dilemma of measuring the “*mean time to first failure or mean time between failures*”. Totally different results could be acquired from either of these options as a program may not function correctly at all according to the first failure but it could then improve drastically according to the mean time between failures. More ambiguity lies within the term “failure”. A failure could be classified as something minor like a program crash or something major like data corruption. There needs to be a scale for failure, not a blanket statement.

Even though software metrics are the most popular method to measure the software engineering process, metrics still continue to have downsides and to be ineffective to an extent. Remarking the efforts of Fenton and Neil in software engineering metrics research, “*much academic metrics research is inherently irrelevant to industrial needs*”. Most of the research done regarding metrics focused primarily on small programs not large systems, which formed an “*irrelevance in scope*”. This research is also generally concentrated on detailed code metrics and not metrics that are necessary for process improvement.

Metrics to this day are looked down upon in most industrial settings. They are only ever really employed when things aren’t looking well or when an external audit or assessment body requires them. According to *Managing the Software Process* by Humphrey, “*in the US the single biggest trigger for industrial metrics activity has been the CMM*” (Humphrey, 1989).

Any work done on a software project produces overheads. Therefore, employing metrics activities does indeed generate overheads for a software development team. Hall and Fenton estimated in 1997 that metrics overheads would be typically 4%-8% for a software team. It's difficult to motivate staff into keeping metrics up to date, especially when other tasks are prioritised, especially near deadlines when stress and tensions are high.

Useful Metrics and Computational Platforms

One of the most modern and popular approaches to software development metrics is Agile. Agile software development is more than frameworks such as scrum, extreme programming and feature-driven development. There's also a lot more to it than practices such as pair-programming, test-driven development, stand-ups, planning sessions and sprints. Agile refers to a group of software development metrics and methodologies based on iterative development, where requirements and solutions evolve through collaboration between teams. Agile methods promote a disciplined project management process that encourages frequent inspection and adaptation. It encourages teamwork, self-organisation and accountability and has a business approach that aligns development with customer needs and company goals.

In Agile, velocity is the amount of work completed during a sprint. This metric provides an indication of how far a software development team has come in comparison to their previous sprint and how much work they have left to reach the sprint objective. Velocity allows a team to understand how long it will take to finish a whole backlog.

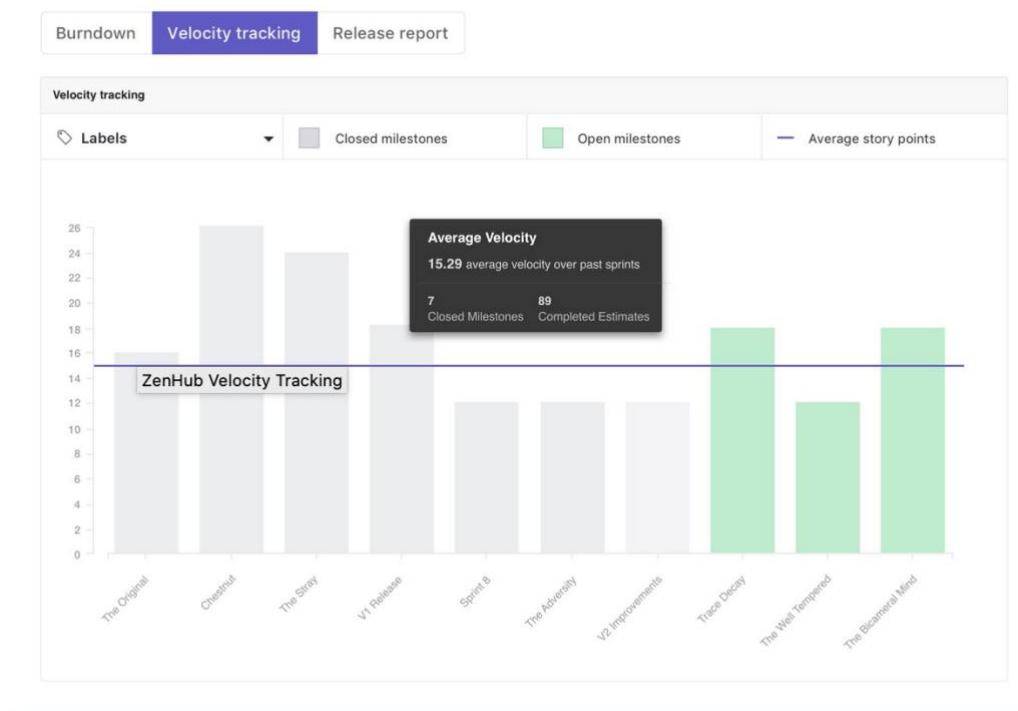


Fig 1: Zenhub Velocity Tracking graph

The above image displays the “Velocity Tracking” feature on a computational platform called Zenhub. This is an Agile project management add-on for Github which tracks team activity and velocity. It’s an easy-to-use computational platform and is very efficient as it is all built into your Github repositories. However, it’s an obvious deal-breaker if a team chooses to not use Github for project development.

When undertaking an agile transformation, there is a need to collect data to demonstrate progress and show improvement. Common Agile metrics approaches do well at measuring team velocity and throughput but can sometimes overlook the requirements of executive sponsors, product management, and other key stakeholders. This problem is often rooted in a lack of understanding about what business goals are driving decision making throughout the organization and what questions we should be answering with the metrics we collect.

The Goal-Question-Metric (GQM) approach is a proven method for driving goal-oriented measures throughout a software organisation. This

approach came from the work of Basili, Rombach and colleagues in 1988. They proposed a simple scheme for ensuring that metrics activities were always goal-driven. With GQM, we start by defining the goals we are trying to achieve, then clarifying the questions we are trying to answer with the data we collect. Thus it is important to make clear, at least in general terms, what informational needs the organisation has, so that these needs for information can be quantified whenever possible, and the quantified information can be analysed to whether or not the goals are achieved.


Measurement Program at ABS/IntelliCar	
	GQM Goal
	Analyze the software development process in order to improve the reliability from the viewpoint of the software developer at ABS/IntelliCar
	GQM Questions
	Q1. What is the total number of defects detected before delivery? Q2. What is the distribution of defects? Q3. Does the type of inspections have an impact on their effectiveness? Q4. Does the experience of developers have an impact on number of faults introduced in the system? ...
	Quality Models
	<i>Effectiveness of inspections</i> Context: company IntelliCar, automobile domain Assumptions: The defect density is comparable across documents. Computation: $\text{effectiveness} = (\text{number of defects detected in inspection}) / (\text{size of document} * \text{training duration})$ Attributes: number of defects detected in inspections; size of document; duration of training

Fig 2: Goal-Question-Metric example diagram

In the above goal-question-metrics example, we can see that the goal is to analyse the software development process in order to improve the reliability from the viewpoint of the software developer at ABS/IntelliCar. Some suitable questions that were raised include “What is the total number of defects detected before delivery?” and “Does the experience of developers have an impact on number of faults introduced in the system?”. From some simple analysis using the goal-question-metric approach we have gone from thinking of an initial goal, to asking necessary questions about the goal and then finally to developing a metric for the “Effectiveness of inspections”.

The use of the goal-question-metric approach in the software development process is advantageous as it aids in understanding and

monitoring the organisation's software practices. It also allows for improvement; a goal can always be expanded upon when needs be. Another benefit of using this method is that it helps the user remain on track. The user is forced to think of a goal and stick to it, elaborating it into questions and eventually (hopefully) solutions. This decreases the amount of ambiguity and confusion around tasks especially in team scenarios where everyone is trying to understand each other's goals.

Another computational platform that I researched is "Source{d}" (pronounced Sourced). This platform's main objective is to allow you to "gain visibility into your software portfolio, engineering team, and processes". Source{d} gives developers visibility into codebases, teams and processes. The codebases function allows users to make wider use of the existing technology and reduces the duplication of software assets.

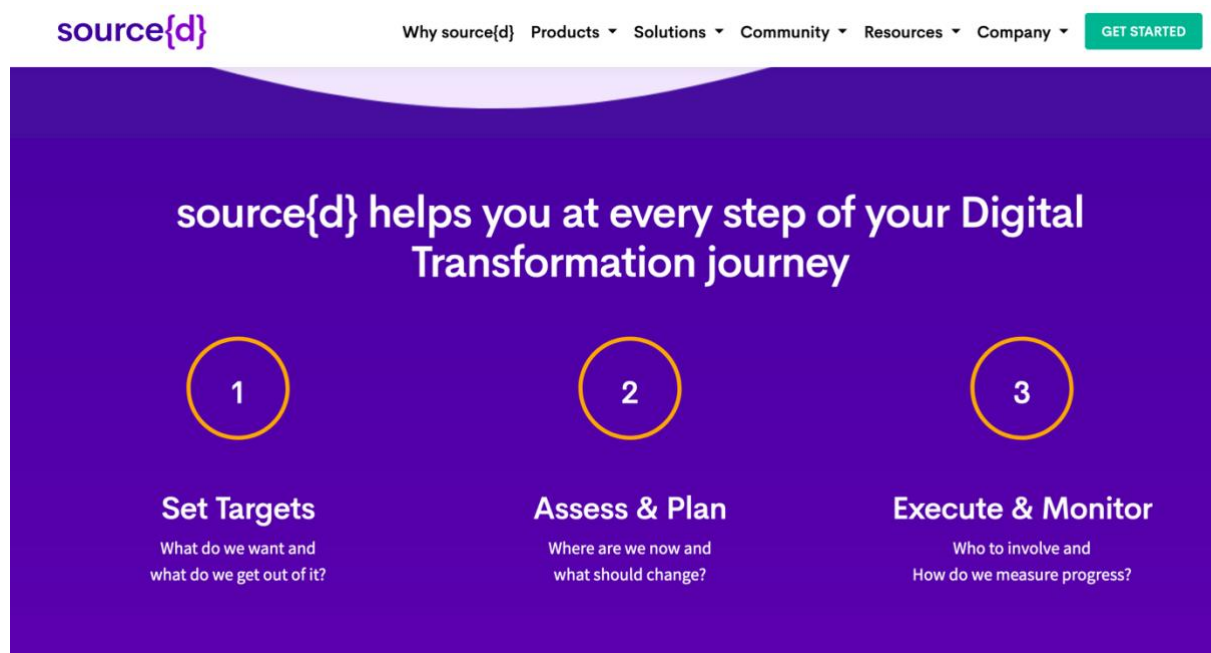


Fig 3: GQM steps outlined in Source{d}

As you can see in the screenshot of the Source{d} website above, this computational platform follows a goal-question-metric approach. It allows users to "set targets", as in make a goal. It then suggests that the user should "assess and plan" which is the same as the question step in GQM. Finally the last step is "execute and monitor" which is equivalent to thinking of a metric.



Fig 4: Example metric visualisations in Source{d}

An interesting case is the Japanese company Hitachi who employ the use of wearable technology to read the happiness levels of their employees. They believe that happiness significantly influences performance in the workplace so they simply decided to monitor each staff member's happiness. In the article on Hitachi by Kazuo Yano and colleagues, they state that "people who are happy have 37% higher work productivity and 300% higher creativity" and that "companies with a large number of happy people have higher earnings per share".

The "Happiness sensors" measure "group happiness" in real time and "physical activity" is measured using three-dimensional acceleration data. The use of people's personal data for the overall welfare of the company raises some ethical red flags.

Ethics

As mentioned previously in this report, measuring the process of software development is a very complex practice. It causes an issue when software development companies are judging and labelling their employees based on simple metrics such as Lines-of-code (LOC) which is difficult to apply fairly to each employee as everyone has different roles and skillsets. It's also controversial for companies to be utilising employee data in first place, personal or not.

However, this becomes even more of a complex debate when the main goal of an employer comes into play – profit. Of course a company will try to do everything in its power to become successful and produce a hard working workforce that create excellent products/services. This sometimes entails the use of employee data to create metrics for the good of the company. Metrics help a company thrive by increasing productivity and locating issues before they become more of an issue. Furthermore, they even help the individual employees to better themselves.

In the case of Hitachi where employee's personal data is being used is understandably a complete breach of privacy. As mentioned in the point above, this data breach is indeed for the overall good of the company and the employee's welfare but it just seems like a step in the wrong direction to be monitoring people's physiological measurements. In my opinion, data use that extends past the workplace is a step too far.

References

- > Software Engineering Metrics: What Do They Measure and How Do We Know? (Cem Kaner, Walter P. Bond, 2004)
<http://www.kaner.com/pdfs/metrics2004.pdf>
- > Software metrics: successes, failures and new directions (N.E. Fenton, M. Neil, 1999) p.149-157
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf>
- > The IEEE Standard 1061-1998 <https://standards.ieee.org/standard/1061-1998.html>
- > Measuring Happiness Using Wearable Technology (Hitachi Review Vol. 64 (2015), No. 8)
http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf
- > <https://ieeexplore.ieee.org>
- > Information on agile project development <https://www.trustradius.com/agile-development>
- > Metric research <https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams>
- > <https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/>
- > Source{d} website <https://sourced.tech/why-sourced/>