Measuring Engineering

A 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.”

2748 Words.

Jane D’Altuin
Junior Sophister
Computer Science and Language

As the technology industry grows increasingly competitive, it is imperative that the software engineering process be made as efficient and effective as possible. The software engineering process consists of every step taken by software engineers to deliver a product to customers. This involves the raw process of writing code itself, but also the processes like liaising with a team, being assigned to a team in the first place, co-ordinating roles, planning a product, testing software and refining a finished product. As such, there are lots of areas where we see potential for the process to be refined. The issue arises with finding objective methods to quantify the software engineering process. It is certain that the ideal process involves the majority of time writing and testing effective code. The goal of better methods being employed by software engineers is to deliver better code that's more effective in less time. This is a topic that should matter to software engineers themselves to stay on the cutting edge of their fields, but also their project managers and anyone with a financial interest in the company they work for. The customer obviously wants the best service and product, and to be left behind as the industry progresses would foretell a grim future for any company. However, the question of what constitutes 'better' code is not objective. The question of what makes good code is surprisingly esoteric in nature, and one that has proven difficult to quantify. The goal of better methods being employed by software engineers is to quantify. Even when the qualities of good code are identified, one must then figure which metrics that are associated with these qualities, and then devise a method of collecting them.

Fenton and Neil (1999) describe software metrics as "activities [that] range from producing numbers that characterise properties of software code (these are the classic software `metrics') through to models that help predict software resource requirements and software quality. The subject also includes the quantitative aspects of quality control and assurance and this covers activities like recording and monitoring defects during development and testing."[1] This paper will look at both traditional and agile software engineering processes, and methods of software engineering metrics as they apply to both. Finally, it will address the matter of ethics as it relates to this field.

[1] Fenton, Norman E., and Martin Neil. "Software metrics: successes, failures and new directions." *Journal of Systems and Software* 47, no. 2-3 (1999): 149. doi:10.1016/s0164-1212(99)00035-7.

In *Searching Under the Streetlight for Useful Software Analytics*[2], Johnson compares the data which is easily obtained to the data that is worthwhile in measuring the software engineering process. The title comes from an analogy that compares basing one's software engineering analysis on the most easily obtained data is as useful as looking for lost keys under a streetlight far away from where they were lost as is is easier to search there. He acknowledges that companies are likely to prefer the measurement and review of metrics which can be collected "with little social, political or developmental impact"[3]. An example of this could be the metric of lines of code (LOC or KLOC), which has been popular since the very early days of software engineering metrics, as measurements like "LOC per programmer per month" could be used to measure productivity and "defects per KLOC [thousand lines of code]" could be used to measure program quality[4]. However, this is not an effective measurement as the LOC is an easily manipulated variable by the software engineer. With no extra effort, the engineer can make themselves look more productive (more lines of code written) or more efficient (fewer lines of code written) to carry out the exact same process. This is exemplified in the below samples of code.

```
if(x>y)
{
      x = x+1;
}
else
{
      y = y-1;
}
```

Sample 1: 8 lines of code (java)

```
(x>y) ? x+=1 : y-=1;
```

Sample 2: 1 line of code (java)

Such actions carried out on a large scale can affect the LOC value by an order of magnitude. Therefore, the usefulness of LOC is limited as a software engineering metric. That being said, Fenton and Neil (1999) observe that LOC has value as a metric precisely because the

---

[2] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 57-63. doi:10.1109/ms.2013.69.

[3] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 57. doi:10.1109/ms.2013.69.

[4] Fenton, Norman E., and Martin Neil. "Software metrics: successes, failures and new directions." Journal of Systems and Software 47, no. 2-3 (1999): 150. doi:10.1016/s0164-1212(99)00035-7.

ease and lack of controversy with which it can be collected means that it is a value measured across the industry[5], and does on a large scale tend to correlate with complexity metrics and values such as absolute number of faults in the code[6].

Considering the type of data which is more difficult to obtain but richer in analytical value, Johnson writes about the Personal Software Process. PSP involves extensive manual data collection, which would add to project overheads. However, it can provide rich data as values are collected in dozens of areas, with a standard PSP form yielding more than "500 distinct values that developers can manually calibrate"[3]. Despite the time and cost of manual data collection, PSP is still valuable as a method of data collection as the values obtained can be tailored to the specific project at hand. This is part of the trade-off that Johnson describes as an "essential design characteristic" of the practise of software engineering metrics[7].

In an effort to address both the labour of collecting data for PSP and the data errors that can come from so much manual data collection, Leap was developed as a tool to automate and normalise data analysis[8]. This proved helpful in terms of lowering the manual effort needed in data collection after the initial PSP entry and performing some automatic analysis of data collected, but also made the process less flexible than the PSP. Johnson's team eventually came to the conclusion already popularised by the community that encourages agile software development, which is that such an increase in overhead as implementing Leap won't necessarily (and often doesn't) justify itself in gains.

'Agile' software development exists as a reaction to the traditional, extensively planned and documented software engineering method. While the decreased documentation is typically considered one of the advantages of this method of software engineering, it also means that traditional methods of assessing software engineering metrics tend to not be applicable as they require documents that don't exist for these projects. Practitioners of agile software development pride themselves on prioritising being people oriented, being able to embrace change, focusing on the product, keeping tasks simple where possible, working in

---

[5] Fenton, Norman E., and Martin Neil. "Software metrics: successes, failures and new directions." Journal of Systems and Software 47, no. 2-3 (1999): 151. doi:10.1016/s0164-1212(99)00035-7.
[6] Fenton, Norman E., and Martin Neil. "Software metrics: successes, failures and new directions." Journal of Systems and Software 47, no. 2-3 (1999): 153. doi:10.1016/s0164-1212(99)00035-7.
[7] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 57. doi:10.1109/ms.2013.69.
[8] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 58. doi:10.1109/ms.2013.69.

self-organized teams rather than under managerial structures, fast delivery of products, and continually improve the quality of the product through constantly releasing new iterations of it [9]. ASD doesn't engage in the traditional method of using an extensive user requirement document. It instead considers separate user requirements as "user stories" and carries out most of its documentation through the lens of addressing these user stories[10]. In agile communities, the rate of productivity is referred to as "velocity" and is measured in terms of how long it takes to complete separate US. This is an effective way to measure what combinations of engineers work well in teams and identify what phases of product development teams get stuck on. Taghi, et al. do warn that as soon as changes are made to a team, any prior velocity data is rendered irrelevant for predicting future velocities of the team[11]. US also produce ongoing metrics for agile teams such as burn down charts (US left to complete) and burn-up charts (US completed). Another effective method of monitoring the software engineering process used in agile communities is the use of the cumulative flow diagram. This documents the percentage of the work that is in different pre-determined stages (eg. Designed, Written, Tested, Released) over different dates. This allows for easy visualisation of the progress of the project. Agile communities then also take measurement of the time spent fixing defects and adjusting to new US and log that as time spent 're-working' the project.

Considering again traditional processes, Johnson asks us, "What kinds of useful software analytics could we obtain if both collection and analysis were "free"?"[12]. It is here that he starts describing his team's development of Hackystat. The Hackystat project flew in the face of conventional wisdom which dictated that one defines one's high-level goals before deciding what metrics are needed to measure achievement of that goal and then how to obtain these metrics. What this team decided was that they would focus on methods for collecting data on the software engineering process in a way that had minimal overhead cost for developers, then figure out what high-level goals these measurements could be used to

[9] Javdani, Taghi, Hazura Zulzalil, Abdul Azim Abd Ghani, and Abu Bakar Md Sultan. *On the Current Measurement Practices in Agile Software Development*. Report. Faculty of Computer Science and Information Technology, University Putra Malaysia. University Putra Malaysia. 2.
[10] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 58. doi:10.1109/ms.2013.69.
[11] Javdani, Taghi, Hazura Zulzalil, Abdul Azim Abd Ghani, and Abu Bakar Md Sultan. *On the Current Measurement Practices in Agile Software Development*. Report. Faculty of Computer Science and Information Technology, University Putra Malaysia. University Putra Malaysia. 3.
[12] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 59. doi:10.1109/ms.2013.69.

guide[13]. The first step here was to collect data from both clients and developers. Johnson acknowledges that modern software engineering metrics included collecting data on individual developers at their local workstation as well as work they made available on the cloud or on company-wide servers. His team also made a point of providing clients with instruments of their own (editors, build tools, test tools, etc)[14]. The team knew that workflow could be interrupted by having to constantly manually record the work process, and wanted to make data collection as unobtrusive as possible to avoid their observation changing the data itself. The example given is that if a developer works offline, Hackystat will save their progress locally before saving it to a server when the developer is reconnected to the internet. This is a much better method than insisting a developer is online all the time, as sometimes that is not preferred or feasible. Due to the constant data collection going on by Hackystat, the data can be analysed as minutely as desired, minute by minute or even down to the second. Johnson observes that this gives a more realistic insight into a developer's process than data collected at the end of a project or over longer intervals. Hackystat also collects data on group work, allowing developers to define ongoing work shared between multiple people to the server. This allows Hackystat to monitor the interaction between members of a team as they all work on the same project or file.

Hackystat is one of the recommended programs by Snipes, Augustine et al (2013) in "*Towards Recognizing and Rewarding Efficient Developer Work Patterns"*[15]. They write about the importance of recognising and promoting the most efficient and effective work practices in software engineering. It is the recognition, rather than the rewarding, that concerns this essay. They envision an automated system to recommend the best methods of performing certain tasks. This involves monitoring developers working on tasks and comparing their actions to the current best practise. The goals they outline for their monitoring is evaluating how the developer works with given code, what process they implement to debug the code, or see if they are following a best practise guideline, the example given being Test-Driven Development[16]. To gather their data, they collected data from the command line and from the click-stream of the monitored developers. It was the

[13] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 58. doi:10.1109/ms.2013.69.
[14] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 58. doi:10.1109/ms.2013.69.
[15] Snipes, Will, Vinay Augustine, Anil R. Nair, and Emerson Murphy-Hill. "Towards recognizing and rewarding efficient developer work patterns." *2013 35th International Conference on Software Engineering (ICSE)*, 2013. 1. doi:10.1109/icse.2013.6606697.
[16] Snipes, Will, Vinay Augustine, Anil R. Nair, and Emerson Murphy-Hill. "Towards recognizing and rewarding efficient developer work patterns." *2013 35th International Conference on Software Engineering (ICSE)*, 2013. 1. doi:10.1109/icse.2013.6606697.

working hypothesis of this group that from monitoring a developer's work patterns, one could divine whether the developer in question was using effective techniques for carrying out a given task. The developers were given a task, and monitored using programs like Mylyn Monitor, which logs commands issued when the developer is using Eclipse, Hackystat, and PROM, which uses plugins to log commands performed within the IDE. Using these tools, the team had access to the sort of wide-ranging and fine-grained data as described above. They also describe a video of the developers' activity being used to record their actions, with this video being translated into a text file describing the steps taken by the developer. This measured both the time taken by the developer to perform their tasks as well as their understanding of the task at hand. They could also analyse the effectiveness of the commands used by the developer by recording values such as the amount of times they repeated commands within the IDE.

The issue of Hackystat, and of this kind of constant monitoring in general, is where the question of ethics comes to the fore. Developers will have their own reasons for being uncomfortable with being constantly tracked. If it can be assumed that over the course of a project, a given developer will spend some amount of time not actively working on the project (be that socialising, going on breaks to suit their work method, etc), a program that constantly monitors activity could be used to justify unfair dismissal. The inability to add personal observations emphasises this, as work can be impeded for reasons as inane as roadworks providing a distraction to those as insidious as a colleague or manager constantly interrupting a worker (at this point the reader is invited to imagine the increased likelihood of this happening if the developer is a woman, racial minority, gay/bi/trans, etc). Privacy concerns will also make developers more self-conscious about the work they produce as they imagine someone else monitoring them, which could lead to a reluctance to innovate or try unusual methods for fear of embarrassment should they fail. This contributes to developers' reluctance to adopt such technologies, which inhibits both the ability of the company to engage with meaningful software engineering metrics and the ability of a given program to be adopted as an industry standard, putting programs like Hackystat in direct opposition to the Lines Of Code standard bearer of software engineering metrics.

What Johnson himself recommends is a hybrid approach to software engineering metrics, one which combines the most effective aspects of automated data collection and analysis with precise, impactful data entry carried out manually by developers. This allows for significantly more effective impact from the data analysis, while still keeping the associated

overheads within an acceptable range[17]. Having carefully chosen areas of analysis will also appeal to developers more as an active, effective method of analysing and improving data rather than appearing as an unnecessary, overly invasive system to interfere with their work style. To further the control that developers have over their data under such a system, Johnson invites the reader to imagine an online third-party repository for the data collected and analytics derived therefrom, where each developer has total control over whether colleagues and management can access their data[18]. This method is still corruptible in the absence of strong workers' rights within a company, as developers could feel pressured into divulging information they'd rather not share for the sake of getting onto a team or securing a project. Nevertheless, this idea is the strongest suggestion for maintaining some kind of data repository like this while maintaining developers' privacy.

In conclusion, there is no one version of software engineering metrics that will suit every project. When devising the methods relevant to a task, one must consider factors such as overheads, quality of data, relevance of data, and effectiveness of data analysis. Any developer involved in the process should also be invested in maintaining a sense of privacy and respect, and as such companies should be invested in looking for processes that suit their developers. While it is tempting to totally avoid getting bogged down in manual data entry by fully automating both data collection and analysis, it is in the integration of manual data collection that software engineering metrics can be made more effective, worthwhile, and personalised in a way that would incur massive overheads from an automated process, should one be developed that can match manual data collection. Nevertheless, automatic data analysis that applies functions such as regression are useful in making the most out of the data collected.

[17] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 62. doi:10.1109/ms.2013.69.

[18] Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): 62. doi:10.1109/ms.2013.69.

Bibliography

Johnson, Philip M. "Searching under the Streetlight for Useful Software Analytics." *IEEE Software* 30, no. 4 (2013): **57-63**. doi:10.1109/ms.2013.69.

Javdani, Taghi, Hazura Zulzalil, Abdul Azim Abd Ghani, and Abu Bakar Md Sultan. *On the Current Measurement Practices in Agile Software Development*. Report. Faculty of Computer Science and Information Technology, University Putra Malaysia. University Putra Malaysia. 1-7.

Fenton, Norman E., and Martin Neil. "Software metrics: successes, failures and new directions." *Journal of Systems and Software* 47, no. 2-3 (1999): 149-57. doi:10.1016/s0164-1212(99)00035-7.

Snipes, Will, Vinay Augustine, Anil R. Nair, and Emerson Murphy-Hill. "Towards recognizing and rewarding efficient developer work patterns." *2013 35th International Conference on Software Engineering (ICSE)*, 2013. doi:10.1109/icse.2013.6606697.