Measuring the software engineering process

Cathal O’Brien

17337323

Trinity College Dublin.

Introduction

This report focuses on measuring software engineering. That includes common approaches to measuring software engineering and the ethical ramifications associated. Software engineering is the systematic application of engineering approaches to the development of software. Software engineering is often a long and expensive process. Thus the industry has strived to produce ways to quantify and measure the software engineering performance. This assists with keeping the process on track, on time and under budget.



This report describes the background and history of measuring engineering and how we arrived at where we are today. It also covers the methods of quantifying and measuring software engineering before looking at the ethical impacts of this kind of surveillance.

Background

Software engineering measurement has been researched since the dawn of time, Unix time that is. For over thirty years we have been trying to accurately measure the impact the software engineering process. The urgency of this has only increased as computers have become a more and more important part of modern life.

Despite (or perhaps because of) the increased programming activity today, most programming projects don’t survive to reach the public. According to an article in Medium less then a third of software engineering projects were on time and on budget in 2016. Also in that article, medium stated that over two thirds of business leaders in IT expect their projects will fail. These examples demonstrate the need to measure software engineering. It also shows that there’s plenty of room to improve in the field of software engineering itself.

Measurable Data

There’s lots of different types of data you can collect when trying to measure a software engineering projects success. Caution is advised here as depending on what metric you choose, you will receive a different interpretation of the projects success. For example, as total lines of code in a project increases, the code coverage of that same project may not increase at the same rate. Ideally the metric you choose will be easy to calculate and relevant to the overall project.

Metrics that are commonly used to measure the software engineering include:

* Escaped Bugs
* Code coverage
* Engineer satisfaction
* Total line count

Escaped Bugs

Escaped bugs are defined as every bug that is detected and reported by customers. A pro of measuring this kind of data is that its very easy to collect. However an obvious flaw of this approach is that it only applies for projects that reach the market. As we defined earlier, most projects don’t reach this stage.

Code Coverage

Code coverage is a very popular metric to measure. Code coverage involves using an external program to count how many lines of code in your project have ever been run at least once. The logic goes that the more lines of your code that get run, the more thorough your tests are. In my opinion the reason for code coverages enduring popularity as a testing metric is that it is intuitive and it’s widely supported in a variety of languages and programming environments.

Some argue that code coverage as a performance metric is overrated. A high code coverage, they argue, will only tell you that the engineers have written a lot of tests. This does not necessarily mean that these tests are any good however (Cooney, 2018). Others argue that as code coverage approaches 100%, the value of this metric decreases. Code coverage is a powerful metric but one that should be used in conjunction with others.

Engineer satisfaction

This metric has been on the up and up in recent years. As the biggest tech companies compete to attract the top talent, they’ve poured more and more into benefits-in-kind. These perks of the job, like an in-house barber and free gym membership, all contribute to increasing engineer satisfaction. The idea goes that a happy programmer is a good programmer. Is there any proof to this though?

Psychologists at the university of Illinois found that while worker satisfaction didn’t have any positive correlation to creativity, the participants that self-reported as happy scored higher in the problem-solving assessment. This would seem to indicate that there is a link between worker satisfaction and overall quality of the finished product.

Microsoft japan recently performed a study where employees were given a four-day work week for five weeks. This led to a 40 percent rise in performance, with a notable fall in the amount of meetings being held every day. The results of this experiment seems to suggest that a happier workforce leads to better results.

Lines of code

Lines of code is a simple metric that is used to approximate the complexity of the project. A project with more lines of code will usually be more complex. While this sounds simple, in practice it is of limited use. Using this metric alone, you can only compare your project to projects in the same language and with a similar coding standard. One must also grapple with the question of what counts as a line. Should one include comments? How about blank lines?

Third-party platforms

Companies often turn to outside programs to measure the software engineering process. These third-party programs, such as Hackystat and LEAP, are custom built to measure the quality of the product. Using these products can free up a team’s time and allow them to focus more on the product itself. They can also provide valuable real-time feedback that can improve the product. However, some of these services blur the ethical line between data-collection and outright surveillance. Also, since these are third party products, one should read the terms of service carefully to ensure your rights are being respected.

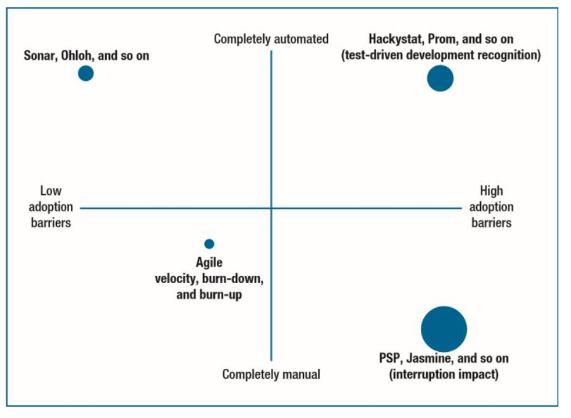
Personal software process

The personal software process is a structured software development process that is designed to help software engineers better understand and improve their performance by bringing discipline to the software development process and tracking their predicted development versus the actual. In one version, PSP asks that developers fill out 12 forms including: time recording log, a project plan summary, size estimation template, time estimation template and a code checklist (Johnson, 2013).

Information collected is analysed and feedback is made readily available to the developers. Real time feedback like this can help locate areas of weakness. However, the drawbacks of this approach soon become apparent. Manually filling out forms for p2p increases project overheads. Also, the results are only as good as the data inputted. Frequently the manual nature of p2p can lead to incorrect conclusions as engineers make mistakes filling out these forms (Johnson, 2013). Garbage in is garbage out.

Hackystat

Hackystat is a kind of PSP data collection system developed by the university of Hawaii. This was developed in order to automate the data collection phase and reduce overhead for programmers. The program achieves this by attaching sensors to development environments and continuously monitoring developer activity. This collected data is then sent to a server for analysis. Hackystat is open source and has been in continuous development since 2001.

Hackystat boasts an impressively unobtrusive data collection system and minute by minute data collection. The opportunities granted by this kind of system didn’t go unnoticed for long and Hackystat soon became a software measurement tool of choice. However, some developers have raised concerns about the types of data collection employed by Hackystat and the ways in which their data was being collected and stored. (Johnson, 2013)

Bibliography

10 reasons why software development projects fail, medium.com (<https://medium.com/specstimate/10-reasons-why-software-development-projects-fail-7200e7c9ae2e> )

Is test coverage a good metric for test or code quality, Chris Cooney 2018 (<https://hackernoon.com/is-test-coverage-a-good-metric-for-test-or-code-quality-92fef332c871> )

Study proves that happy programmers are better engineers, Phil Johnston 2014 (<https://www.itworld.com/article/2693429/study-proves-that-happy-programmers-are-better-programmers.html> )

[Scale of Positive and Negative Experience Affect Balance](http://internal.psychology.illinois.edu/~ediener/Documents/Scale%20of%20Positive%20and%20Negative%20Experience.pdf)

Searching under streetlights for useful software analytics, johnson 2013

Microsoft Japans 4-day workweek experiment sees productivity jump 40% (<https://www.cnbc.com/2019/11/04/microsoft-japan-4-day-work-week-experiment-sees-productivity-jump-40percent.html> )