Róisín Ní Bhriain - 18326577

Measuring Software Engineering

Abstract

This essay will examine the process of measuring software engineering. The first section will examine the methods used to measure engineering activity. Then the platforms that can be used to gather and perform calculations on the data gathered will be discussed. The third section will explain some computational methods used to profile the performance of software engineers. Finally, I will give my opinion on whether this type of profiling is ethical.

Introduction

Measuring Software engineering is something which is done in every software producing company worth it's salt.

Measuring Engineering Activity

A piece of software is at its essence, a computer program that consists of lines of code. However this is irrelevant when it comes to the end product. The software could also include documents of any requirements or specifications. Any consumer, however, will use software based on its functionality rather than any documentation provided or how the code was implemented (Sidler, 2002). Essentially to measure software engineering you will need a ratio of the software value / the labour and cost involved in creating it.

In the past Lines of Code (LOC) was the measure of a software engineer. This would be divided by Programmer Month (PM). This calculation gave an indicator of the programmer's productivity. There was also another measure used for program quality which was the number of defects per thousand LOC (KLOC). The LOC were used as a *size variable.* In the mid-1970s it became clear that this was a crude measure. There were many different types of programming languages and comparing the lines of code of a lower-level language vs a higher level language was not possible using the LOC measurement. Thus there was an increasing interest in measuring the complexity of code (Fenton & Neil, 1999).

Data from commits could also be used as a metric. The number of commits a user makes may not be a good metric in itself as there may only be a few line changes or new comments. Similarly, the size of a commit doesn't give too much information on the progress of a project. It is possible that an engineer may be solving a problem over several commits. Commit messages will tie all the data together and indicate how much the user has worked on the project and what has been achieved in the commit. The commits will also tell us how much progress a user considers to be reasonable enough to commit.

A software defect is essentially a flaw that impairs the software or the software process. These defects are caused by human error but not all mistakes cause a defect. These defects will be detected during formal reviews and testing activities as well as design and code inspections. Problems can also be detected during the development of code. It is important that problem reports are generated following these discoveries. Customers may also report problems in the user experience (Florac, 1992). Keeping tabs on defects and problems in code will give a good indication of how the process is working in a company. Ideally, most of the defects will be caught early. So it would be better for a company to catch defects during reviews, inspections and testing. If the customer is reporting more problems than your software engineers are reporting during the development of a piece of software then something is going wrong during the process. Another question can be asked: can we analyse the data from these problem reports? You could of course count the number of defects per project and call it a day. As we know, things are rarely this simple. Thus we may need to indicate how urgent a defect is or how critical the issue is, maybe even the type of problem (hardware, OS). By assigning traits to a problem we can then begin to analyse the data.

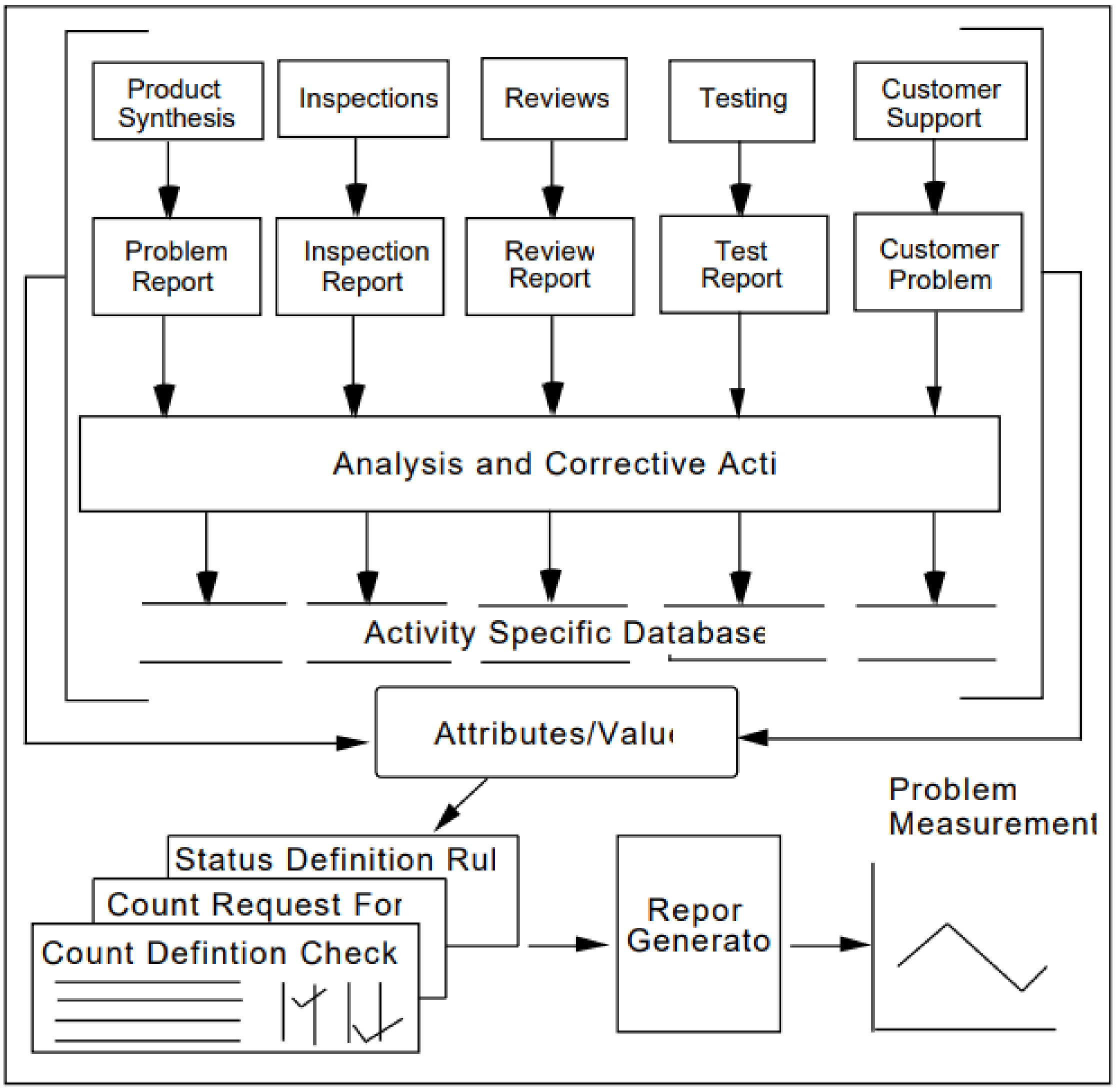

Figure 1: An example of methods for locating problems in software from (Florac, 1992, pg 10).

A pull request is a part of the code synchronisation required by version control servers. It is a request to change code in a central codebase. Pull requests are often used by managers as part of the engineering process. They require collaboration from a team and thus cannot be used to measure engineers on an individual level. When analysing the data from pull requests, comments can be analysed to see which comments engineers respond to more productively i.e. meaning issues will be fixed quickly (Ortu et al., 2017). Other metrics that can be pulled from pull requests include time to merge, pull request lead time, pull request comments etc. Pull request lead time is how long a pull request takes to be either merged or closed. This is an important measure of how long it takes before engineers are getting to solve pull requests after they are opened. Time to merge is how long it takes for the original branch to reach the master branch. Taking the difference between these two gives an indicator of how long an engineer works before opening a pull request. You can also pull the information from comments to see how much discussion there is before a pull request is closed. Measuring the size may also be a useful metric (Guimarães, 2019). Of course, all this data is best used together to gather an accurate representation of how the team is merging and closing the pull requests. Ideally, you would want a team to be opening and closing pull requests at the same rate to keep the work flowing.

Platforms

Software companies generally tend to use version control systems. These systems will gather information on each commit and this data can then be gathered and analysed. The repositories will have insight on pull requests, productivity, work style etc. This can give analysis into a team as to who is solving issues in the project, who is making the most significant changes in the code and so on. His data can be accessed using an *application programming interface* (API) such as:

- GitHub API
- BitBucket API
- GitLab API

These APIs are free and they offer raw data on repositories and users.

Analysing the gathered data is another story. Most people in charge of software engineering teams will not have a deep understanding of statistics and wrangling meaning from data. So to properly understand the productivity of their team they will most likely need some software to perform analysis on this data. Luckily there are many platforms that do this kind of analysis and often will connect with GitHub or other such sites to perform analysis on data directly from the source. Some examples of this kind of software:

- Jira
- GitClear
- Pluralsight
- Code Climate Velocity
- LinearB
- Hackystat
- SourceLevel

The second section should be concerned with the platforms on which one can gather and perform calculations over these data sets. The important infrastructure here is the emergence of utility computering which supports the gathering of large volumes of data, and the processing of this data via algorithms. This infrastructure enables complex and computationally expensive calculations to be performed. Your text in this section should explore what is possible here, and also what specialist infrastructure has emerged to perform various kinds of data analysis.

Data and Analysis

- Function Point Analysis (FPA)

Function Point Analysis was developed in 1979 by Allan Albrecht (IBM). There were problems with other system size measures such as LOC as discussed previously. A system was needed to allow different systems/languages to be compared regardless of the technology used. It measures the size of the system in two areas: *specific user functionality* and *system characteristics*. Specific user functionality is essentially a measurement of 5 types of functions that the application delivers for user request. These consist of external input, external output, external enquiries, internal logical files and external interface files. Each function will be placed in one of these categories and classified from low, average or high, after this it will be given a weight. The weights will tell us the size of any information processing and the sum is referred to as the Unadjusted Function Points.

Function Point = (User Functionality) X (System Characteristics)

In terms of System Characteristics, there are fourteen that are identified to evaluate the functionality. They each have a Degree of Influence (DI). These will range from having no influence to strong influence (0-5). The sum of these will then tell us the Value Adjustment Factor for each project (The Government of the Hong Kong Special Administrative Region of the People's Republic of China, 2009).

The product of each of these gives us the Adjusted Function Point:

Adjusted Function Point = (Unadjusted Function Point) X (Value Adjustment Factor)

- Constructive Cost Modeling (COCOMO)

This model was first introduced by Barry W. Boehm in 1981.

- Cyclomatic Complexity

The general concept of cyclomatic complexity is concerned with conditions and control statements. It is a quantitative measure of the complexity of a program.

Function Point Analysis, Constructive Cost Modeling, and Cyclomatic Complexity

The third section of your report should be concerned with various kinds of computation that could be done over software engineering data, in order to profile the performance of software engineers. And there are various techniques for data computation, including but not limited to defeasible argumentation of the kind used in expert systems, simple counting, various software engineering algorithms measuring concepts such as coal and complexity, and various machine learning approaches for

plastering and analysing data sets. Your section should attempt to set out the landscape here and consider the strengths and weaknesses of various techniques for the problem at hand.

The Ethical Conversation

Ethics must be considered when doing any kind of analysis of personal data. This is especially clear to people after the introduction of the GDPR (General Data Protection Regulation) in 2018. People are a lot more aware of how their data is being collected and processed. In light of this, the question must be asked. Does each engineer consent to their data being processed and analysed in order to assess their productivity. I can imagine that not everyone would be happy with this.

There have been plenty of dystopian films and media regarding the faults of measuring the productivity of workers. Most likely the measuring of software engineering would not extend to the measuring of heart rate and eye contact and the likes as is done in these films. That is not to say that it won't happen in the future and in my view that is certainly unethical. The introduction of GDPR was intended to slow down the exploitation of personal data but companies will find a way around this without a doubt.

I think it is important that people do have control over their own personal data. While it may be beneficial for management to have access to data from workers in order to promote productivity, I don't think that from an ethical point of view that they should. What's important to remember is that each engineer in a company is also a person. In general, each engineer specifically in such a specialist job like software engineering will be a fairly motivated person. Giving each engineer access to their own data may be a good solution to this problem. Each worker can then assess their own performance and work on improving what they regard as important in their work. Something that is worth remembering though, is that if workers are rewarded for achieving certain things in their work then it is certain that they will focus on these things. So is it worth doing this kind of analysis on your workers to encourage them to work harder or would it be easier for management to change the work environment to keep the workers happy?

Doing data analysis on worker productivity is not the only way to improve worker productivity either. That is why companies do not focus solely on these metrics.

Before section should be concerned with the ethics and legal or moral issues surrounding the processing of this kind of personal data. The core question is whether it is reasonable to perform various kinds of analysis on the performance of software engineers as they go about their work, or whether some of this crosses a

line. And your goal here should be to form an opinion that is informed by your prior research and assessment. You may take any view that you like.

Conclusion

### this is by far the worst essay I’ve ever had to do (and I did higher-level English in school)