Software Engineering
Measuring Engineering Report

Adam Warde
Student Number: 13328034

December 4, 2017

Introduction

The purpose of this report is to show some of the methods as to how software engineering and engineering projects in general can be measured. We will ask what data we can measure, whether performance can be measured at all, as well as some platforms that can be used to measure performance, algorithmic approaches as some ethical concerns also.

A good place to start would probably be to define what we mean by software engineering. A satisfactory definition I found on wikipedia is "Software engineering is the application of engineering to the development of software in a systematic method". We should also note that in almost any software project a team or number of teams will be involved. One thing we should note from this definition is the "systematic method" part. If something has a method then that method must be followed and therefore performance of individuals can be measured based on this method. However do all projects have a method, does each team member even use the same method. A team member may not be performing well based on analysis of a set of data but is that data correct and are you measuring the correct things and weighing their contributions correctly. The first section of this report will deal with measurable data. Mostly what types of measurable data there are and how useful they can be when associated with a method and how they can then be used to compare performance of individuals and entire projects.

Measurable Data

Metrics

What are the different types of metrics that can be used to measure the productivity of a developer?

An easy one to start with is time spent working on a project. Whether this is time spent physically in work or time spent in your IDE this is an easy one to measure. If you want to measure when an employee is at work you could simply implement a clock in clock out system, similarly you could use a program to log how much time the developer has there IDE open for. But are these metrics really useful. For example a person may be at there desk with their IDE open but are they actually doing anything useful. Two developers may spend the same amount of time with their IDE open while one developer does twice as much work. So although this data is easy to obtain it is arguably not very useful to measuring performance. At least not on its own.

Another easily obtainable piece of data we could find is the amount of lines of code written by a given developer. But again does this really give us an accurate account of their performance. You could be given two programs that each do the exact same thing, one being twice as long as the other. Is the longer program better simply beacause it has more lines. No, arguably it is worse being more verbose and less efficient. This method is also open to exploitation with developers simply writing longer code so as to artificially up their productivity.

A final simple metric that could be used is number of commits assuming that a revision control system such as git is in use. However again this has similar pitfalls to the above method as more commits does not necessarily mean more work done and it can also be gamed by simply committing more frequently.

A more complex metric that can be used is tests. Tests are a good way of showing if a program does what it is supposed to do in all cases.. However who writes the tests. If an individual developer writes their own tests this can lead to a bias where the developer simply writes the code and then writes test that the program will pass. This can be solved by having another developer write the tests but this also has pitfalls. The developer writing the tests may have a different way to go about the problem than the original developer a so the tests require the developer to solve the problem in a certain way that they might not be comfortable. Tests also are not a great way of comparing performance. They don't take into account how long it took to solve a problem or how well they are done.

Estimates are also a better way to gage performance. This is when before a task is done a time frame is given for when a problem is solved. However who should calculate this estimate. The developer doing the task, if so many times people will give simply come up with the estimate after the task is done and give a similar estimate to how long it actually took. Conversely letting a manager or someone else make it dosnt take into account for how good a developer may be at a given task. Estimates often don't take into account for issues encountered during a project.

Although individually many of these methods seem week they can be used in conjunction for better results. When using many of these methods at the same time each factor becomes less important individually and hence any error within a metric is reduced. It also gives us a bigger picture of how the project is coming along and many metrics to look at. We can possible combine them together into an algorithm assigning each metric a significance from high to low for how important we think it is and adding or multiplying them together to get some statistic for someones perfromance.

Gathering

Now that we have outlined some of the measurable data we want let's look at how this data can be gathered.

The data can either be gathered by individual developers themselves or by automated systems. Lets first have a look at a manual system where the developers gather the data themselves.

PSP is a platform where the developer gathers there own data and then uses PSP as a framework to analyse this data. The upsides of a PSP style system are that they are easily extendable and there are no privacy issues as it is the developer who is gathering and analysing the data individually. If a developer believes that some issue is affecting their performance it is easy for them to use the framework to analyse that issue and see if it is actually affecting them.

However there are some downsides to PSP style systems. They require a lot of work by the developer to gather and analyse the data. While mid workflow you may have to stop and log data and then later analyse it. It also has a steep learning curve with developers having to do a lot of the heavy lifting themselves. It is also prone to bias with individual developers not putting in the correct data or manipulating data and requires the developer to buy into the system.

The other type of gathering system is something like Hackystat where all data gathering is automated.This has the advantage of a low input cost for the developer as they don't have to spend time inputting data. It also provides its own analysis of the data so the developer doesn't have to.

However these systems do have some drawbacks. As the data gathering is automated only what is automated can be analysed and so if a developer feels that some other piece of data is affecting their performance it is not easy to add to these systems. Another issue with these automated collectors is a privacy issue. Many developers feel that they are being spied on by management with all their data being used to access them often times without their knowledge.

As is often with data, the harder it is to obtain the more useful the data is and often automated systems don't look at the behaviour of the developers but often at the product itself which doesn't help engineers improve themselves.

Computational Platforms For Doing This Work

In this section we will take a look at some platforms that can be used to measure the performance of developers.

PSP

PSP or the personal software process is a manual input platform for measuring software development and trying to improve developers performance. Its main objectives are to improve estimating skills, reduce defects in work and manage the quality of projects.

As PSP mostly only provides simple spreadsheets it is easy for developers to track and add analysis for specific problems that they think are affecting their work and is very in depth in its work.

However there are some downsides to PSP. One of the main problems is that it requires lot of developers to track their own work and log their own data which can interrupt workflow. It also has a steep learning curve and requires developers to buy into it to work properly. It is also open to personal bias as it is the developer themselves that inputs the data.

Hackystat

Hackystat is a platform for measuring software development. It is automated with plug ins for developers that records data on a minute to minute basis and even second to second in some cases. It seeks to remove much of the overhead of a manual data collect method such as PSP. It also has server side data collection as most most software development also includes cloud based services. One of its other interesting features is its group based features which allow project leaders to define projects and shared artifacts to represent group work.

However there have been some complaints about Hackystat, mainly it's automated data collection which developers were apprehensive about. Many developers were uncomfortable about management having such in depth knowledge of what they were doing.

Ohloh

Ohloh now known as open hub is a open source software measurement web service that provides software metrics and statistics about projects on revision control repositories.

Although originally mainly used for getting global statistics on languages on projects it has since opened up its API and can now be used on any project to get metrics and statistics. They have also released a new metric called a project activity indicator (PAI) which uses an algorithm to combine the number of commits and contributors along with the history of these commits to give an indicator of how active a project is.

Algorithmic Approaches

COCOMO

COCOMO the Constructive cost model is an algorithmic approach to estimating the time taken for a software project to take, these estimates can then me compared to the actual time taken to complete a project and gage the performance of the team.

COCOMO has 3 different levels, basic, intermediate and detailed. The basic model can be used for quick and easy approximates of how long a project will take, however its accuracy may not be great due to the lack of data required.

Intermediate takes in more data and takes longer but is more accurate due to using factors to calculate the estimate. Detailed takes the most data and gives the best approximation of how long a project takes but requires more data than both basic and intermediate.

COCOMO as has three different modes, organic mode which is used for small and stable projects, embedded mode which is useful for larger projects that have a complex software interface and tight constraints. And finally semi-detached mode which is for projects larger than basic but smaller than embedded.

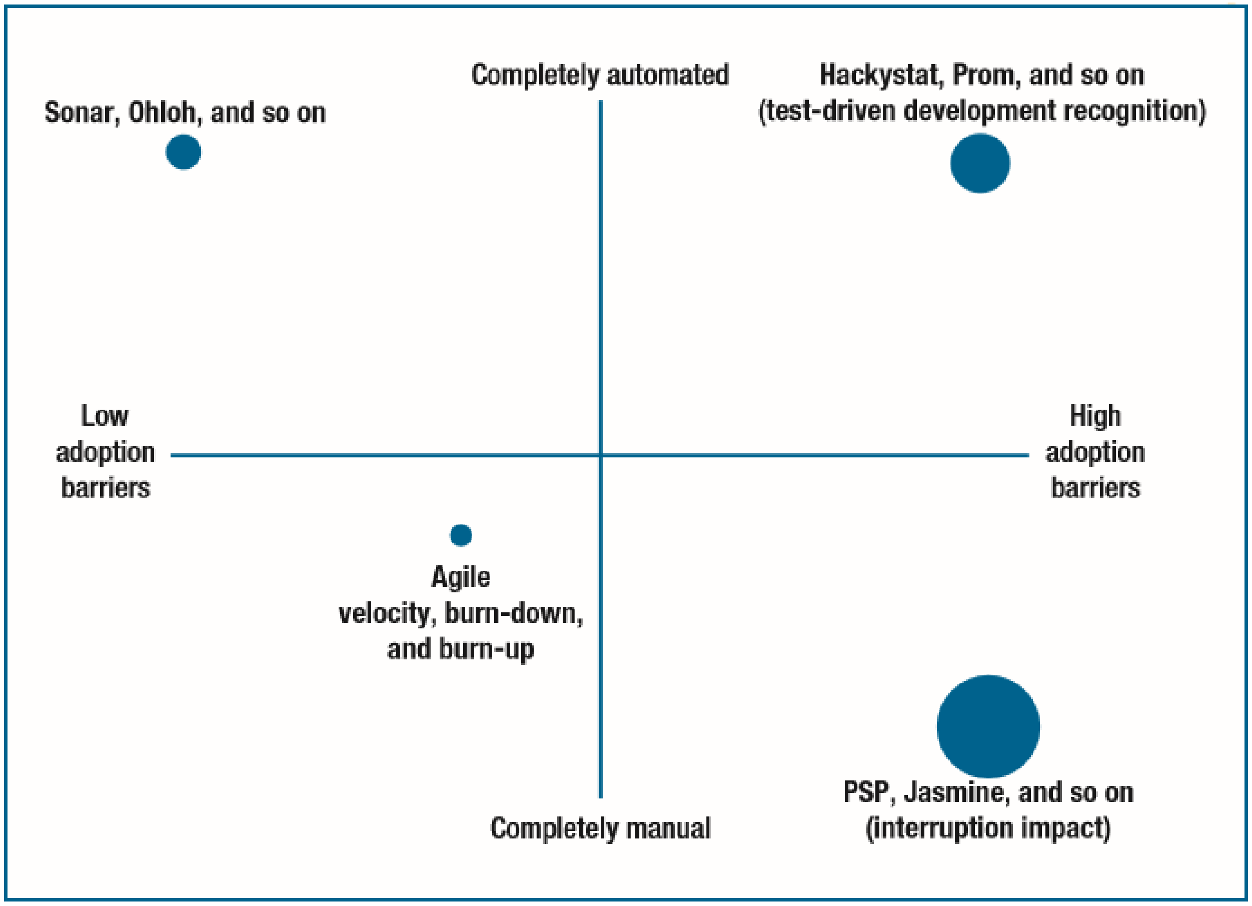

Figure 1: A graph of various software measurement platforms based on how automated they are and how hard they are to use

COCOMO uses a number of different formulas to calculate the effert taken to complete a project which is then used to calculate the development time. It also takes into account a number of effort multipliers and cost drivers whose importance is determined by a manager who assigns them a significance rating from very low to high. These are then all combined to give the final estimate.

Ethical Concerns

Although these methods and metrics can be used to improve productivity and compare the performance of employees there are some ethical concerns.

The most obvious ethical concern is one of privacy. Many of the software platforms can be very intrusive logging everything a developer does on their workstation. This often doesn't sit well with developers who don't want their data being seen by management to the degree that they are. Some of these platforms and ideas for measuring performance not only take into account what an employee does at work but also what they do outside of the workplace with employers monitoring what employees are doing on their social media accounts. Should someone be judged based on what they do outside of the workplace? With many software jobs offering the ability to work flexible hours and from outside the work place, increasingly employers are arguing yes to this question. There is no longer company time so they want to know what you are doing at all times.

Due to the nature of these measurement platforms they can have huge affects on a developers life. Many companies use these tools to decide on pay, promotions and even whether an employee should be let go. Therefore companies and individuals who use these tools must be absolutely sure that these tools are used correctly and at the right times. Sometimes these tools can be used to misinterpret how a person is doing and it may be their circumstances thatare causing a performance drop or the tools are simply looking at the wrong things, after all correlation does not imply causality.

Conclusion

In conclusion we have taken a look at some of the metrics that can be used to measure the performance of software engineers involved in a project. We have seen how some of these metrics can be misleading in how much information they can provide us based on how they are collected and who is collecting them and that a number of these metrics should be used in conjunction to get the best use out of them.

We have also looked at a number of computational platforms that can be used to measure the performance of software engineers. There are the mostly manual platforms such as PSP that require the user to input most of the data and do their own analytics but are more extendable. Then there are the mostly automated platforms such as Hackystat and Ohloh which gather most of the information by themselves and do the analysis themselves but are narrow in their spectrum of analysis.

We also had a look at some algorithmic approaches to estimating the time taken on a project so it can be compared to the performance of a software developer, mostly with a view to the COCOMO model.

Finally we had a look at some of the ethical concerns of these platforms and of measuring performance in general. Mostly the privacy angle of management having so much access to developers data and sometimes even monitoring them outside of work. But also the fact that these tools can have large influences on people's careers and hence should be used with care.

Bibliography

[1] Philip M. Johnson, University of Hawaii at Manoa *Searching under the Streetlight for Useful Software Analytics.*
`http://www.citeulike.org/group/3370/article/12458067`

[2] *Software Measurment Slides*
`http://www.cs.umd.edu/ mvz/cmsc435-s09/pdf/slides16.pdf`

[3] Brian York *The Best Developer Performance Metrics*
`https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0`

[4] *Evolution of the networked enterprise*
`http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf`

[5] *Wikipedia for software engineering*
`https://en.wikipedia.org/wiki/Software-engineering`