Measuring Engineering – A Report

Rory Murphy

Introduction

Measuring the productivity of workers has been a field of study since the early 20th century. Pieces such as Fredrick Taylor’s “*The Principles of Scientific Management”* and Frank and Lillian Gilbreth’s motion studies established the methods by which manual worker productivity has been measured for the past 100 years. However, these processes of optimising productivity by determining the optimal method of completing tasks and monitoring employees to ensure that these methods are strictly adhered to are no longer considered to be the definitive ways of measuring employee productivity. While the ethics of such meticulous probing of employees is discussed later, the key reason for the failure of scientific management today is that it cannot be easily applied to knowledge workers. As pointed out by Peter Drucker, knowledge workers, unlike manual workers, are not expected to complete the exact same task over and over. Instead, they are required to constantly ask themselves “What is the task?” and determine the best way to complete it (Drucker, 1999). With requirements that vary from job to job and constantly changing requirements in said jobs, the scientific management methodologies quickly become irrelevant for knowledge workers such as software engineers.

Due to this complication, many people argue that the measurement of software engineering productivity is a pointless endeavour. Articles such as *“The Myth of Developer Productivity”* argue that commonly used measurement methods such as lines of code and defect rates can simply be gamed by developers and restrict the innovative mindsets required to thrive in the software engineering environment (Barnes, 2015). Others, such as Fenton and Neil, point out that “Much industrial metrics activity is poorly executed” (Fenton and Neil, 1999) and if data is poorly collected and analysed, how useful an analysis does it produce?. Despite this, however, recent advances in measuring and analysing methodologies as well as developments in cloud computing technologies have led to the creation of more sophisticated data analytics techniques that, while not yet being perfected, can further advance the field of measuring software engineering, and other knowledge worker productivity. This report discusses the various types of data that can be measured, the tools and techniques used to measure this data and the algorithmic approaches used to convert this data into meaningful and useful analysis of the processes and practices used by software engineers. Furthermore, while the development of data collecting and processing methods greatly increases the amount of information now available to managers about employees, there is a pressing ethical concern with regards to the collection and usage of this data that should not be ignored. Thus, this report also dedicates time to the discussion of this issue in the hopes of ensuring that the ramifications of these levels of analysis are well known to the reader.

Measurable Data

Measuring data is the groundwork of performing any sort of analysis and forms the basis of productivity measurement. Without collecting relevant data, effective analysis simply cannot be performed. This section discusses some of the more general measurements made of software engineers to track their productivity as well as some newer methods adapted by agile environments. The successes and failures of these is also explored as is the effect that the Cloud and other technologies has had on the data being collected. Finally, the data collected to perform some of the more advanced performance analysis methods is discussed.

One of the most basic measures of a software developer’s productivity is the number of lines of code written. This method dates to the late 1960s (Fenton and Neil, 1999) and during the early years of software engineering was a commonly used measure of a program’s size. The flaws with using such a measure are clear to even a novice programmer, as it is easy to pad a program to make it appear longer. Worse still, pursuing longer code leads to a variety of negative results such as less optimised code and employees being afraid to delete lines even though doing so could make a program run more effectively or easier to understand (Barnes, 2015). Ultimately, large amounts of code can have a negative effect on a piece of software, and this is a prime example of a measurement that, while appearing to be relevant and useful, hurts the productivity of a team when measured.

Other simple metrics such as the number of bugs closed, the number of defects per 1000 lines of code and time estimations made by developers are similarly problematic measures because, like lines of code, they simply promote less effective work habits. Barnes discusses these, pointing out that measuring the number of bugs closed promotes creating more bugs in the first place, measuring defects will make programmers avoid attacking more complex problems for fear of how it will affect their perceived productivity and holding software engineers accountable for time estimations will create a situation where the dominant strategy is simply to estimate longer as well as cause difficulty when a problem’s definition and requirements inevitably change during the process (Barnes, 2015). The common element between these measures is that they are simple to find and understand and appear to be relevant at a surface level. However, it does not take much exploration to show that using these methods to measure productivity can reduce productivity, causing developers to sacrifice effective coding practices and avoid difficult work to improve these measures. Johnson refers to this type of analysis as “searching under the streetlight”, whereby easy to collect data is analysed however its result are mostly unhelpful (Johnson, 2013).

There has been a recent movement in the software engineering practice towards Agile development; a method that promotes faster response to changes and greater client communication to uncover “better ways of developing software by doing it and helping others do it.” (Manifesto for Agile Software Development, 2001). As a result, it has been noted that different productivity measurement methods are required to adjust for these changes (Javdani et al., 2012). Javdani et al.’s paper refers to a number of techniques centred around breaking down a project into very specific tasks required for the project’s completion and assigning them story points; measures of the amount of effort required to complete a specific task.

The following are the main measures created from story points. Effort estimation; a team’s estimation of how long each piece of work will take using story points to record the size of each relevant task with their sum being used as the measure of effort. Velocity measurement; the speed at which stories are being completed by individuals and the team which can be used to determine how productive members are relative to one another as well as be combined with effort estimation to determine an approximate time until completion of a project. Burndown charts which create an estimate of the amount of work to be done in a period, which can then be compared to the amount of work completed for said period. Cumulative flow charts which show the amount of backlogged work yet to be completed. Response to change measures which measure the hours spent reworking the project which is used to indicate the project’s ability to adapt to change.

These measures can be used to give a better picture of the size and speed of a project as well as the efficiency of each member of the project (Javdani et al., 2012). These measures are more focused than the previously mentioned methods, with clear links between what they measure and how it affects a project. As well as this they are lightweight and well suited to the agile environment in ways that manually collecting large amounts of data are not suited to (Javdani et al., 2012). However, their dependence on story point estimations, which are based on employee estimates, severely reduce their effectiveness, and as pointed out by Barnes, result in inconsistencies from team to team and allow developers to simply up the story point measure of the work that they do if their productivity is sited as being low (Barnes, 2015).

The methods mentioned up to this point have relied on basic measures and individual judgement to analyse productivity. However, the advent of cloud computing, which allow for low cost data storage that can easily be accessed (Pocatilu, Alecu and Vetrici, 2010), as well as the rise in accessibility of automatic data collection tools (which are discussed further in the next section) has led to an increase the availability of far more specific information about employees and their work habits. With this vast increase in measurable data there are now a variety of methods for which to assess worker efficiency.

Hassan and Xie propose that software repositories should be mined for the rich data that they contain about software systems and projects. Software repositories are generally used as a record keeping system, however with the volume of data contained by them, they have the potential to greatly enhance software development decisions, providing a wellspring of relevant information about the work being done by engineers (E. Hassan and Xie, 2010). Now that companies can easily store this level of data about their employees on cloud services, they have gained access to a huge amount of analysable data.

Another approach to measuring software engineers is to measure their workflow (Snipes et al., 2013). By using automated data collection systems (which are discussed in the next section) to collect “fine-grained data” about the specific steps taken by software developers when performing certain tasks to determine which methods are efficient and which are not. The information collected here can not only be used to assess which engineers are using more productive methods than others, but also to “provide information on how to improve and motivate the developer to adopt the new patterns in a positive way” (Snipes et al., 2013). Similar to repository mining, this approach would not have been feasible without recent technological innovations.