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 measures as well as data measurement in general in this industry is also explored. Finally, the data collected to perform some of the more advanced performance analysis methods is discussed as well as how it differs from the previously mentioned measurements.

One of the most basic measures of a software developer’s productivity in 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 and pursuing longer code leads to a variety of negative results such as less optimised code and employees being afraid to delete lines despite the fact that doing so could make a program run more effectively or easier to understand (Barnes, 2015). Ultimately, large amounts of code appear to actually have a negative effect on creating an effective piece of software, and as a measure it is a prime example of a measurement that, while appearing to be relevant, actually hurts the productivity of a team when measured.