Software Development Metrics

A Method of Improving Efficiency or an Impediment to the Process?

1 Measurement

In this section I will examine the nature of measurement, measurement in the context of software development and its purpose.

1.1 Defining Measurement

Measurement is the application of empirical methods to an object or process in order to extract quantifiable data on its nature or behaviour. Measurement is collected through a series of standardised steps in hopes of using this data individually or by aggregating it to test hypotheses, plan further action and to understand the object or process being measured better.

1.2 Defining Software Process Measurement

Applying standardised methods to the processes of software engineering is a means of extracting valuable (or invaluable) data. Various components of software development are easily measured such as lines of code, number of errors, hours spent programming or the length of program execution. Other figures are much harder to establish and often impossible to measure, such as the value of the contribution of one programmer, improvement of a program version to version or even productivity of a software developer. Measuring complex values like these is challenging as, unlike the number of lines of code, these data points are not readily available to us and require several steps, including deep analysis and inference, before they can be known.

1.3 Why, Who and How

Parties interested in the results of software metrics are generally organisations or individuals that want to quantify progress on a project and the productivity of the efforts in its completion. By looking at one of the earliest studies conducted on software development by Sackman, Erikson, and Grant (1968)[1], we can see the first attempts at gathering this data. Their intention was to measure the difference in programming performance between coders with a lot of experience and those with significantly less. Their results demonstrated a huge difference between these two groups, with those with many years of experience coming out miles ahead those in the beginning of their career. The results of the study come as no surprise, their significance however, lies in the tools the researchers used to measure performance and productivity. Their methods laid the ground for the future studies to come that attempted to measure software processes and development, namely to compare the productivity of various programmers on projects (Curtis 1981[2], Edberg & Bowman 1996[3,] , Hochstein 2005[4]; to name a few) as well as to understand the effectiveness and cost of the development process within an organisation (Faraj, proull 2000[5], Valett & McGarry 2000[6], Boehm et al 2000[7]). Questions like these arise more so now than ever before as multinational tech companies become leaders in a nation's GDP and take the world stage as leading employers. Their output now more than ever needs to measured, analysed and predicted with accuracy in order to manage their costs and investments to maximise revenue. Different methods have been used in the past and many more continue to be developed today.

1.4 Do We Need Metrics?

The issue with measuring software development lies in the very nature of the task being measured. Software development is difficult to quantify. A famous quote by Bill Gates surmises this issue "Measuring programming progress by lines of code is like measuring an airplane by how much it weighs." Though overused, the message behind the quote stands to be repeated again and again throughout history as we attempt to quantify one of the most complex processes in the world today. There is a reason programmers are paid a much higher salaries in the U.S.[8], U.K.[9] and Ireland [10]; the task of software development is renown for its difficulty and there are too few individuals signing up to do it[11]. Naturally, organisations jump to metrics as a means of finding faults and improving efficiency with their existing employees while investing vast sums into recruitment of top performers [12] in the area based on the data they collected to measure such performance. *But, are the valuable resources of these companies well spent on this task*? In this essay I will aim to answer the above by analysing the developments in algorithmic methods of performance measurement, what advantages (and disadvantages) these methods present to organisations and individuals and conclude by describing possible alternatives to existing software metrics.

2 An Algorithmic Approach

By analysing developments in algorithmic approaches we can see where organisations prioritised certain software development process components, why they did this and what this meant for the organisation.

2.1 When Metrics Didn't Matter

The first computer to be capable of storing a program in memory was delivered to the US government in the 1950's[13]. It marked the beginning of an era. Since then, the progress of technology accelerated at an astronomical rate, with the first microcomputer being released in 1960 and the first desktop, home-use device following in 1964. Like the invention of fire and the wheel, the computer changed the world. Then onwards we began to see advancement in technology propelled forward not only by organisations, but by computer hobbyists. The birth of programming as a pastime and an expression of creativity brought to this world the many things we take for granted such as the Windows Operating System, Apple computers, the Apache web server and many more. Did those brilliant individuals ever busy themselves with attempting to measure their productivity? Had they, would things have been different, been more efficient? Does quantifying progress on software development accelerate it or deviate from the task at hand? It is hard to say what difference between the performance of a programmer under scrutiny against and that of a programmer working under their own jurisdiction as the data for one cannot exist and therefore cannot be used to perform a valid comparison. Thus, this limits my analysis to the group of programmers that were analysed in scientific studies, as well as in organisations that adopted algorithmic methods to measure their efficiency.

2.2 Measuring Productivity

One way of measuring software processes is by analysing through a systematic method the end results of the software processes and comparing them to the input, or simply put; productivity. Productivity is not objective and can be measured in many ways. Productivity can be measured as the efficiency with which resources are deployed to develop software based on requirements (KS White 1999 [14]) In another study, productivity is said to be the fraction expressed as the production output divided by the cost and efforts to its creation development (Nwelih & Amadin 2008 [15]). There are more definitions but their distinctions are slight, generally the consensus is that productivity is a measure of the output divided by input and these two variables are determined individually. So what is output; lines of code, number of functions, decrease in time to execute a program or the speed of bug resolution? How can we measure productivity fairly across a number of organisations when so many different factors affect the software development process. To determine which is a measure of results for an organisation or researcher, they must first identify what is considered important in the context of the software. Due to the lack of a uniform measure of software effectiveness, this decision will be entirely subjective. As a consequence, two studies cannot be compared fully accurately as they each measure productivity in a slightly different manner. This issue was what spawned the growing volume of research on the topic of measuring software development. This research not only focuses on attempting to measure productivity but to also use the data as a method of comparison between different teams and individuals in attempt to identify factors that affect it, including the characteristics of a team/individual, choice of in-house development or offshore and products used in development, to name a few.

In one study; 'Measuring Productivity in Software Industry' (Anselmo & Ledgard 2002), a model of productivity is presented that measures the productivity as the input to the production of a piece of software and the revenue it generated. A curve illustrates the input of money into the project across time; a characteristic of this curve is that more time is spent on error correction and support of the software than its initial product development. A second curve is used to illustrate the revenues from the software. From the initial release, revenues enjoy a steady rise until they reach a plateau. The plateau remains unless a new product is developed or the system becomes obsolete. The productivity is thus the ratio that is found in the cross-section of these two curves where the return on investment is the total revenue less investment. The ratio is described as $C*T = K*M*T$, where C is the cost incurred, T is time to develop the project, K is the constant describing overheads and M is the number of man hours worked as a total.

While the theory behind it is simple, the practical applications are flawed. Several factors contribute to the number of M and C that cannot be accounted for through this simple equation. These factors are:

- Language or platform used. Is it proprietary? Low-level hardware? Completely new technology? Many of these could contribute to the increased man hours to complete the project and thus it cannot be controlled for across different projects.
- Quality of developers available to the organisation. If a team with mostly experienced developers is compared against a team of starting programmers the results will be heavily skewed in favour of the team that is more advanced without it being accounted for in the equation.
- Environment in the workplace. One of the most overlooked factors is the culture curated in the workplace by the organisation. Does it stimulate creativity? Does it punish mistakes? Is it a safe space for workers to challenge themselves or is it bounded by rules with no room for self-development? When the place of work for a developer is more draining than inspiring, then the time taken to complete the project will inevitably increase.

These are not the only issues of concern when attempting to measure the input to a project and to use the result as a means of comparison between other projects, but there is enough there to illustrate that the simple metric comparing cost and man hours is not adequate to capture the whole picture when it comes to productivity.

If an organisation wishes to analyse their productivity and use it for predictions of resource requirements and expenditure, then they must account for all the factors that influence the input, as described above. Furthermore, a company must use existing research that controls for these factors and leverage it in their own development processes so as to maximise their efficiency and decrease costs. For example, it has been shown that the capital advances and the use of the latest technology has been beneficial to the increase in worker output (Hitt & Brynjolfsson 1996, Francalanci & Galal 1998) and that reusing existing software can be beneficial to time savings (Antovski & Imeri 2013).

A revised version of this exists whereby the productivity data takes in to account reuse (of pre-existing structures and programs), functionality and length, dividing it by the effort taken to produce the end result. While this adds further complexity to the figure of productivity, painting a more realistic measure of the output, it uses somewhat dated understanding of size i.e. the length is measured in LOC (lines of code). However, its inclusion of code reuse describes a more realistic scenario of development within organisations but, it does not describe how this can be measured if the code is taken from sources outside of an organisation or previous projects e.g. internet resources with suggested code solutions. Often programmers will use code that already exists on the internet and there is no way to measure exactly how much of the code used in a piece of software was sourced from examples or solutions elsewhere on the internet. Of course, this issue would not arise in an organisation that relies on solely proprietary software.

2.3 Measuring Goals

Moving away from simple calculations, we look to a metric known as Goal Question Metric which gives software developing teams an opportunity to characterise their metrics in a way that is unique to the project in question (Basilii, Caldiera, Rombach 1999) [18]. It adopts a hierarchical structure in which the first step is the determination of a goal i.e. the purpose of taking the measurement. This goal is identified by selecting an area of improvement in existing software or alternatively, specifying a standalone piece of software that needs to developed for some purpose. The process then requires the creation of several questions that will define the steps needed to completely fulfill the task of goal completion. This end goal could can be realised by creating a quality model that pertains to the issues; such a model can be a list of desired functions in the end product software. Finally, measures must be identified that will aid in the process of development by answering some initial questions in relation to the issue and by giving direction for the efforts towards a quantifiable goal (i.e. speed of execution of a certain component like updating a database). After this has all been completed, the project executioners must update their data collection methods along with implementing a systematic method for validating and analysing the metrics.

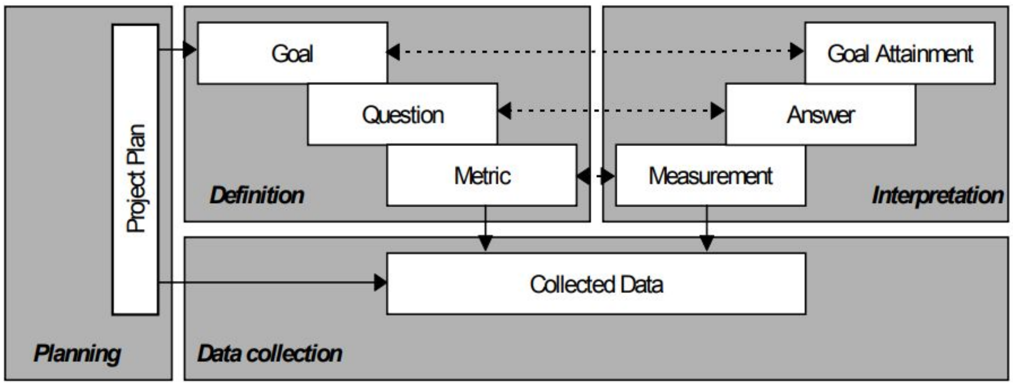

Figure 1: Four phases of Goal Question Metric

The cost to an organisation to implement this method is two-fold; in actual physical costs to train staff and develop systems to record data regarding the ongoing process of GQM, and in cost to the organisation in terms of effort and man hours spent on maintaining GQM across the duration of the project (Solingen, Berghout 1999)[19]. Currently there are no industry standard tools for the GQM method and all data is collected initially on paper and, later, on in-house developed tools. This can prove somewhat arduous and, according to the writers of the ‘Goal/Question/Metric’ method handbook, is the most time consuming in the initial phases of development where the goals and plans are set out. This is done through multiple one-on-one sessions with team members and through further feedback sessions. In an organisation that wishes to prioritise efficiency, this method can be seen as a deviation from the task and can even act as a defuser on creativity as the initial energy found in the beginning of a project is dampened with tedious back-and-forths between management and developers.

2.4 CVS and Later Developments

With the dawn of the internet and open-source programming, came the tools that are used by developers across the world to control all processes in the development of software. Concurrent Versions System is a tool that allows programmers to track all changes to a piece of software with details regarding time of change, editing history and the authors of the changes [20]. These tools have been extremely beneficial to programmers in the open-source community where all changes are publically moderated. However, there is a pessimistic view that developing software through CSV can give rise to issues in software integrity. This is seen especially in cases where the artifact being worked on is still in early stages and the core product must be modified regularly, unlike in open source software where a core product exists and is modified in the occurrence of issues and suggested performance improvements.
CSV also does not provide any measurable evidence of performance improvement and is merely a method of counting contribution and non-complex productivity metrics such as lines of code and functions.

3 Ethics & Issues

This final section will be a discussion of the ethical issues presented in measuring software processes and their impact on the organisation and individual.

3.1 Measurement as a Dysfunction

A popular text on the topic of measuring performance and processes is Robert D. Austin's 'Measuring and Managing Performance in Organizations'. The book describes the methods and ethics of measuring various performance aspects within an organisation, finally coming to the conclusion that if a process cannot be measured in its entirety, with 100% of components accounted for, then it yields what the author describes as 'dysfunction'. This presents the idea that an activity within an organisation cannot be measured completely, the very act of measuring it deviates from its performance and thus yields a lower performance as a result of the time spent attempting to measure it.
This can be easily applied to software, where the task of development is so overly complex and multifaceted that it is impossible to measure every single aspect. where the time spent on a project is made more expensive through the capital it requires, both in technology and man-hours, and is

thus very valuable. So this begs the question, are we wasting valuable resources on efforts to measure performance when in fact we could be stunting the end figure by this action alone?

3.2 Company Priorities

In software project metrics, several factors are considered to contribute to the end product. One such factor is the experience level of the developer. Companies seek to grow their productivity by hiring developers with X-plus years of experience in the industry in order to deter themselves from experiencing a decline in output through the acquisition of under-experienced employees. This leaves a gaping hole in the industry whereby developers are not taken on and enter a vicious cycle whereby they need a job to gain experience and cannot get a job without the experience.
Rigid job requirements prevent companies from taking on ambitious, young talent that can benefit their company in the long run by way of in-house training and self-development. Companies that adopt a motto of personal growth and creativity see a much lower turnover than companies that do not prioritise employee training (Brum 2007) [23]. Thus, the ethics of measuring a companies success solely by its end of year revenues and production costs can act as a detriment to the employee welfare and as a result, the end product.

Conclusion

The report above detailed several methods of acquiring metrics on software development. The first method was focused on the company output as measured against the company input; or simply put, productivity. Using productivity as a metric of software process has been shown to forego many factors that contribute to the end result, rendering it a poor estimation of the process. It focuses on simply the output of the developers, ignoring the added value such as personal growth or contribution to the future projects in the form of frameworks or software tools. As well as this, it acts as a dampener to creativity as it only rewards rigid measures such as lines of code (or functionality, complexity, etc.) and does not reward non-functional results for example, the UI or user experience.
Second, we looked at project metrics in GQM where the measurable data was identified through development of goals and quality models. While this method is thorough, it is time-consuming and requires effort on behalf of the whole organisation to maintain. There is no existing software that helps calculate the ongoing collection of data that accompanies the method and thus everything must be recorded by hand or by means of proprietary software which must then be developed in house, further using up valuable resources. It can act as a demotivator and can stunt the process.
Third, we touched on process metrics which involve comparing version-to-version development of software through existing CSV tools. This method is applicable in situations where the process of development needs to be recorded and can be controlled on a small to medium scale; this is why it is commonly seen in open-source software. To apply it to the development of software within a large scale organisations would be difficult as it threatens integrity control and would require the

development of a robust platform that would oversee this process. As a result, many organisations implement their own version control tools unique to their company needs.
Methods for measuring software development are numerous and their focus depends on the needs of the organisation taking them. Companies in the software industry prioritise different aspects of the development process and it is for this reason that no uniform method of software metrics exist. Software engineering is complex and hugely varying, thus it is impossible to measure it in its entirety. The attempts described above are simply that, attempts. The organisation that endeavours to measure their processes should understand that no one figure will be an indicator of their success, rather it is a light that may lead the way in the right direction.

Bibliography

1. http://www.dtic.mil/dtic/tr/fulltext/u2/645438.pdf
2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1456356
3. https://www.tandfonline.com/doi/abs/10.1080/07421222.1996.11518117
4. https://dl.acm.org/citation.cfm?id=1105800
5. https://pubsonline.informs.org/doi/abs/10.1287/mnsc.46.12.1554.12072
6. https://www.computer.org/csdl/proceedings/hicss/1988/0842/02/00011818.pdf
7. http://csse.usc.edu/TECHRPTS/2000/usccse2000-505/usccse2000-505.pdf
8. https://money.usnews.com/careers/best-jobs/software-developer/salary
9. https://jobs.telegraph.co.uk/article/software-engineer-salary/
10. https://www.irishjobs.ie/careeradvice/it-salaries-2018/
11. https://www.cnbc.com/2018/09/06/companies-worry-more-about-access-to-software-developers-than-capital.html
12. https://devskiller.com/true-cost-of-recruiting-a-developer-infographic/
13. https://www.computerhope.com/jargon/u/univac.htm
14. https://accelconf.web.cern.ch/accelconf/ica99/papers/mb1o01.pdf
15. http://docsdrive.com/pdfs/medwelljournals/ajit/2008/484-488.pdf
16. https://www.researchgate.net/publication/220427489_Measuring_productivity_in_the_software_industry
17. https://www.researchgate.net/publication/262377645_Review_of_Software_Reuse_Processes
18. http://www.cs.umd.edu/~mvz/handouts/gqm.pdf
19. https://courses.cs.ut.ee/MTAT.03.243/2015_spring/uploads/Main/GQM_book.pdf
20. https://ir.canterbury.ac.nz/bitstream/handle/10092/9677/tr_0405.pdf?sequence=1
21. https://savannah.nongnu.org/projects/cvs/
22. http://www.it.bton.ac.uk/staff/rng/teaching/notes/CSCWgroupware.html
23. https://digitalcommons.uri.edu/cgi/viewcontent.cgi?article=1022&context=lrc_paper_series