Measuring Software Engineering Report

Module: CSU33013 Software Engineering

Name: Rory O'Donnell

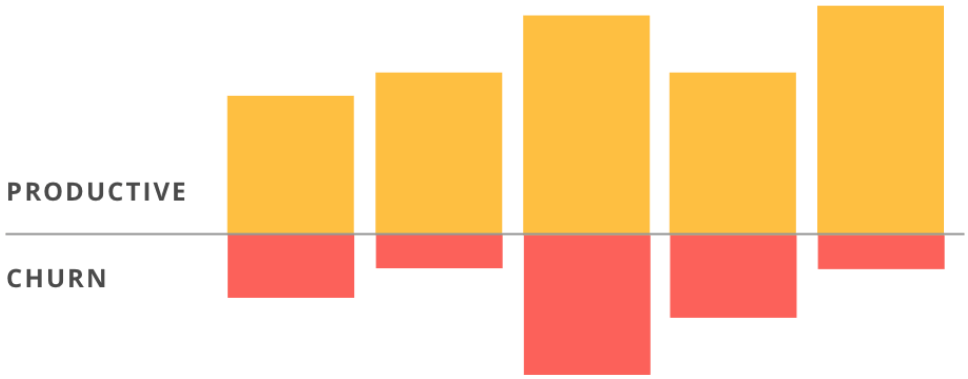
Student Number: 16321866

Measurable Data

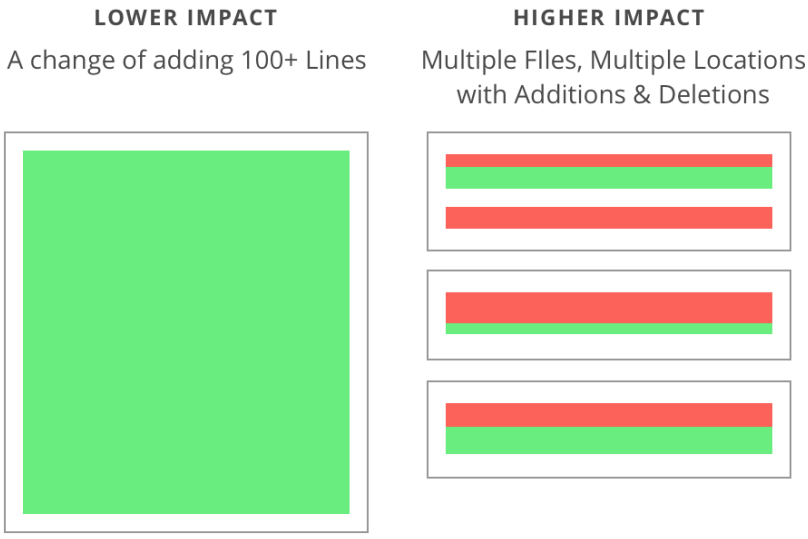
The process of measuring software engineering is a rapidly growing area of interest for many large corporations. An idea, that at first glance can seem rather difficult to execute accurately, is quickly becoming a high priority for most engineering companies. There are many reasons for this, some of which include the advancement of software in this area, its wide availability as well as the increasing popularity of the concept and its implications. The engineering process is an inherently difficult thing to measure for the simple reason that there is no one obvious metric to measure. Essentially, this means that the evaluating of this process involves recording a multitude of different aspects and combining them together. Different techniques exist, however, there are some common metrics measured in most methods.

Some of these evaluations are of obvious things such as the amount of time spent on projects, lines of code written and the number of commits to a repository. Of course, these metrics are flawed in their conclusion of whether an engineer is doing efficient work. For example, the amount of code written does not equate to good or important code to a program, and often complex problems can have solutions with less lines of code than those of simple problems. While each of these methods can be easily critiqued as not being accurate ways of measuring software engineering, they are still metrics which are easily recorded and used in combination with other measurements. Merging these results with other seemingly more abstract metrics can start giving some useful conclusions on the competence of an engineer or a team of engineers.

One of these combinations is the measurement of code churn. Code churn is the percentage of a developer's code which represents an edit to their own code. This measurement clearly must include the amount of code written and compares it to the amount of code which was added, modified or deleted to a program. Churn can be measured between each commit or over a certain period of time. If there is a spike in the amount of code churn of a developer, it may indicate inefficiency in the software engineering or a problem with the development process. Of course, this is not always a completely accurate measurement of efficiency. In cases where the problem being worked on is very complex or integral to the program, code churn is very common. Churn between multiple developers may also occur in agile projects where anyone can work on any section of the code base. These situations may result in high amounts of churn but not necessarily point to problems or inefficiency.



Another common metric in engineering measurement software is the impact of the code on the program. This is a measure of the effect that changes to code have on the overall project. The impact can be calculated using a variety of factors such as the amount of code in the changes of a commit, the number of different files it affects and the severity of the changes in relation to the program. The impact is not always proportional to the number of lines of code as, for example, many lines of code could be added but only affect one function on one file and so have little impact. Whereas a few lines of code added over many files in a program would have much greater impact. The complexity of the changes or additions made, and their difficulty can be factored into a higher impact score. The impacts at different points in time could also be compared and be used as a metric for the amount of progress being made on a project at any given point.



Efficiency is the measure of the productivity of an engineer's code and is evaluated by balancing coding output against the code's longevity. It is not related to the amount of code written, rather, a high efficiency rate is linked to the length of time the code is providing value in a program. Naturally, a higher churn rate would result in reduced efficiency ratings. Usually a high efficiency involves lots of commits, irrelevant of code length, and a medium to low churn rate. However, this can vary from engineer to engineer, as some people just naturally work better with less commits or higher churn rates without being less efficient as a result. Efficiency

can also vary massively depending on the problems being worked on at a given time. The fixing of bugs integral to the working of the main program would likely be less efficient than constant updating of a service by adding small new features.

The measurement of bug rates of an engineer or a team is another widely utilized metric. This is generally the average number of bugs generated for every commit or update of the code. This can be measured on an individual basis or per deployment of the program. The severity of the bugs can also be tracked and compared, e.g. an integrated bug which stops the whole program from working is far worse than an isolated bug which interrupts a small function of the overall project. The bug rate can help determine if code from a certain developer is of decent quality or is causing more problems than it is fixing for each deployment. Some other statistics often recorded alongside bug rates for similar purposes include memory utilization, changes in run time and effects on garbage collection.

Software companies are also enhancing other areas of their project development along with the introduction of engineering measurement to their eco-systems. As the art of engineering itself evolves, it is being realized that all sections of software development can be optimized for increased quality and even quantity in output. Individual software projects can be analysed in planning now to find the optimum number of developers for the project and the different numbers suitable for different sections of the projects' life cycle. Software development measurement systems can suggest these numbers before a project or by analysing data from an ongoing project, they can give feedback on the best strategy going forward.

Over time, these measures can save companies resources and financial costs which can be used on other development projects.

Computational Software Available

As already outlined, the increase in the amount of software engineering measurement systems available has been very noticeable. This supply does not come without demand, as more and more companies are looking to these systems to try and optimize their workers' outputs. As the financial benefits of the implementation of these systems is recognized by the wider engineering market, more and more corporations are seeking to get involved.

However, the cost of this software is not cheap. Many companies charge a monthly or annual rate or offer a purchase of the entire package for a greater price. On average, these systems cost €500- €600 per month or a few thousand a year at a better value rate. The few companies that do sell their entire packages price generally price them for a few thousand or one and a half to two times more than their annual rate. These rates can vary depending on the amount of people being monitored, with packages for 100, 1000 or unlimited number of employees. The prices of the software seem quite high at first but due to their complexity and intricacies it

is probably justified. Their prices are likely to only increase with greater sophistication of the software and growing demand.

Most large software multinationals are involved in some sort of development of these kinds of systems or they are at least linked to companies designing them. An example of a company like this is Mavenlink. The 'project management software' as they label it, is a very popular choice for a lot of companies. They provide services for managing employees as well as systems for tracking time management, project budgeting and resource allocation within a development process. It is also clear to see their popularity in their integration with other services such as Google Drive and Google Sheets. Like many alternate services, they provide a demo service for companies trialling different systems in this area.

Communication software for developers is another type system used for gathering information and statistics on employees. Slack is an example of this. These types of software are essentially a messaging service for software developers to communicate with one another and share ideas and solutions. Slack, like other services of this type, have many popular applications integrated in its service such as Github, Dropbox etc. These services can easily track what its users are sharing between each other about their specific development projects. The information gathered, such as the use of the integrated apps, can be useful in the process of measuring software engineering development for employers. Slack and many others like it are free services and nowadays the majority of software companies implement the use of these services in their own development projects. Packages for larger software projects are also available for purchase, with more features that are more suited for big development teams.

Other services available include data analysis of development projects. Some companies and even universities offer a service of detailed examination of the data obtained from the use of software development measurement systems. For projects of a larger scale, there would likely be vast amounts of data generated purely due to the amount of time spent or developers involved. Volumes of data this big can often be difficult to draw conclusions from and need critical analysis to gain useful feedback from it. The services offered either take data already generated from a project or ask project teams to use their preferred engineering measurement software to create the data. After the analysis of the data, feedback is given to the companies on areas to improve and even areas of success. These services can be useful for those unfamiliar with the use of software engineering measurement and provide an unbiased summary of the efficiency of development and some key data analytics.

The reasons for the notable increase in the adoption of engineering measurement systems and software are quite interesting. A core reason for this is simply the availability of the software. More and more companies are releasing packages for software companies to monitor their engineers work on many different levels. The trust and acceptance of these systems has also massively increased in recent times as before, the concept of measuring the software engineering process was quickly dismissed. While some still hold this view, with greater exposure to these systems, enough managers of software engineers believe enough in the

concept to implement these systems for their own employees and so, the business for this type of software has grown. Of course, for many companies, profits are the main goal and the adoption of these systems is seen as a valuable and worthwhile move.

Other than the doubts in the legitimacy of the concept, the benefits of the systems if they are to work as intended are clear to see for employers. Managers can easily monitor the output of their engineers and intervene if they deem their employees work to not be up to standard. Advice could be offered to employees on how they could improve or on areas they could work on in their job. As well as this, it is human nature to try look better when under the microscope and so this could lead to higher employee outputs. Management can also pinpoint standout employees to possibly reward or promote them and keep them in their companies. Unfortunately, the opposite may be true for this as managers may choose to let employees with less efficient outputs go. This of course brings in to question the morality and ethics of these software systems, from the constant monitoring of employees to the possibility of costing people employment.

Tracking the growth of the use and development of this software will be an interesting prospect. At this point in time, it is likely greater detail and sophistication can still be achieved in software development measurement systems. The greater uptake of these systems will only encourage the further development as well. However, an interesting proposition is whether companies will eventually integrate these systems internally or will they continue to outsource the process to other companies. As the systems become more widespread and ingrained in software development, one would assume that at some point, all software development projects will undergo measurement and monitoring. If this is the case, it is likely companies will have structures in place to constantly measure their developers work and won't need purchase packages from outside sources. This could be obtained from linking or joining with software measurement companies, developing their own systems, or the software simply becoming mainstream and open source for all to use.

Algorithmic Approaches Available

The evolution of development measurement software has led to many highly detailed systems which dissect and analyse almost every known aspect of engineering. As already mentioned previously, the number of different metrics being recorded for evaluation is as large as it's ever been, meaning the algorithms needed to handle all this extra data have become more and more sophisticated and complex. With the substantial quantity of data being accumulated, various algorithms must be used to measure the many metrics and to merge the resultant data points into useful, simple output.

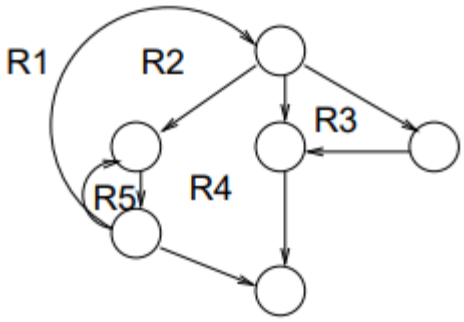
One of these methods is the Halstead Complexity Measures for software, created by Maurice Howard Halstead in 1977. Halstead's goal in designing his algorithms was to identify measurable properties of software and the relations between them. For his algorithms, he has four base variables of the given software: the number of distinct operators, the number of distinct operands, the total number of operators and the total number of operands. By adding the distinct number of operators and operands, we get the program vocabulary and by adding the total number of operators and operands, we get a base value for the program length. Using these results, Halstead suggests measures such as the Estimated Actual Program Length, the Volume, the Difficulty and the Effort involved in writing a program can be measured. His algorithms also offer estimates for the approximate number of bugs in a program and the approximate time to actually write the program.

n1 = no. of distinct operators in program
n2 = no. of distinct operands in program
N1 = total number of operator occurrences
N2 = total number of operand occurrences

Program Length: $N = N1 + N2$

Program volume: $V = N \log_2 (n1 + n2)$

Another method for evaluating software is McCabe's Cyclomatic Complexity of Programs created by Thomas J. McCabe, Sr. in 1976. He claimed it to be a measure of testing difficulty and reliability of modules. Mccabe's hypothesis was that the difficulty of the understanding of a program was largely determined by the complexity of the control flow graph for it. The complexity of a graph (M) is defined as: M = E – N + 2P, where E is the number of edges in the graph, N is number of nodes in the graph and P is the number of connected components. Mccabe also includes an algorithm for calculating the cyclomatic number (V) of a connected graph (G) which is the number of linearly independent paths in the graph or number of regions in a planar graph. For this measure, he recommends a maximum V(G) of 10.



The algorithms mentioned primarily focus on the measurement of the software itself and not so much the process. However, measuring these metrics is a completely viable and commonly used strategy to assess software engineering. If the software can be critically evaluated, one can then infer that the development process in making the software was satisfactory, substandard or somewhere in between. For example, a program with low difficulty ratings but a corresponding high time spent rating can easily be evaluated as poor without any expertise needed. Of course, like all of these methods, they cannot be used in isolation to determine whether a program is of good quality.

The key question in relation to engineering measurement software has to be can these systems accurately evaluate software engineering performance and whether they can do so to at least the same standard that a qualified person could. Otherwise, these systems are essentially useless as a human being could simply do a better job. In my opinion, the answer involves both the systems and human interaction. These systems can already generate endless statistics for many different metrics of an individual or a development team's work, but what can actually be done with this information? Personally, I believe the optimal use of these systems involves a data analyst monitoring their application and filtering the generated data into useful feedback to give to management of development teams. The use of the systems alone would likely lead to over saturation of information and no real direction as to what to do from knowing this information.

Another issue in relation to software engineering measurement is the scalability of the systems in use. While the algorithms available are undoubtedly useful and viable as a method of evaluating engineering, it is hard to discern the quality of the development process on very large-scale projects. For example, a system might highlight that the work of a team in a development project was behind schedule and not up to standard. However, this team could have been working on a much more difficult section of the project than any other. Even with the algorithms already highlighted, there is no definitive method to determine the quality of their development process without more investigation, which would likely come from an employee and not a software system.

These systems may also struggle to accurately depict the contributions of individuals in a team. It is a common occurrence that the few outweigh the many in terms of contribution to software engineering projects. This can be in terms of sheer code written or difficulty of the problems solved within the project. In the case of the latter, a programmer may only have achieved a few tasks compared to the many of their peers, but their tasks, of course, are of greater difficulty. Systems may struggle to highlight this happening in a development project and its feedback may unfairly reflect the work done by each member.

Despite these concerns, the software is incredibly detailed, of very high quality and with proper monitoring and management, can be extremely useful for companies in accurately tracking their developers' engineering.

Ethics

The topic of software engineering measurement cannot be thoroughly discussed without addressing the morality and ethics involved in the practice. This is the same with almost any sort of increased monitoring of people in any area. Should it be done?

The natural, initial reaction for almost everybody is to reject the idea. This is in part due to legitimate doubts in the viability and accuracy of these systems and partially due to the increased monitoring and constant evaluation of performance. However, with the evident popularity of the systems with management of development teams, software engineers have had to adapt to their integration within their companies. Surprisingly, many engineers are welcoming towards the systems as long as they are accurate. They are, for the most part, confident in their own abilities and do not mind the detailed evaluation of their work on the condition that they are fairly awarded for good engineering. This is likely a relief for employers who may have feared some sort of backlash on introduction of these systems.

However, just because some may be tolerant of these systems, it doesn’t eliminate the genuine concerns of the morality of the systems. It is also not only about the doubts of the concept, but also with what employers can use the information for. As of yet, there is no sanctions in place to prohibit management of developers using this data to punish those with results below whatever their own personal standards. In theory, employers could punish or let employees go for not generating adequate results from the systems evaluations. While it seems unlikely, that they could terminate employment based solely on the feedback of their system, it could easily happen or at the very least be the point which tips scales in a decision. With the ambiguity of what the exact output of the systems actually means, this could be very dangerous for employers to base any serious decisions on the results. In short, the performance of engineers cannot be measured solely on the results of these systems and the prevalence of them in the

software industry nowadays could set a dangerous precedent of over reliance on their feedback.

Another element to consider with software development measurement is its compliance with the recently introduced General Data Protection Regulation (GDPR) laws. What party owns the data that is being generated in these systems, the employee or the company? According to GDPR, it is likely that the employee owns it, meaning the employers implementing the systems must ask for permission to use and store the data. This only leads to more questions, are employees aware of their rights regarding their data? Are employees asking for permission for use of the data? Even assuming the answer to both these is yes, are employers asking for permission when hiring or before projects? Theoretically, permission could be asked during interviews, possibly pressuring employees into agreeing to secure employment. Whatever the answers are to all these questions, it cannot be denied that the guidelines, restrictions and rights in this area need to be clearly defined for all parties.

An obvious counter argument to people rejecting these systems, is the frequent evaluation, monitoring and even surveillance that exists in other employment fields. Very few professions in the modern world escape consistent evaluation, from periodical inspections to evaluation forms from those higher up in companies. Any retail workers, police and others are continually being filmed through CCTV or performance cameras. Similar types of monitoring employees exist in almost all other fields of work, so why not software engineering? While it's difficult to credibly dispute some form of evaluation being introduced, caution in this area must be encouraged. Software Engineering is a uniquely troublesome when it comes to measurement of its process due to its inherent complexities. The software cannot be allowed to be used as the only form of assessment for engineers as they are still unreliable. The introduction of these systems should not be dismissed entirely however, they should come with strong regulations protecting those who are subject to their consequences.

Personally, I would be against the use of software development measurement in my own work environment. Having investigated the possibilities and limitations of these systems in depth, I believe that, in their current state, they are unable to accurately gauge the quality of software development due to its delicate and implicit intricacies. Perhaps with further development in the future the systems could achieve a level of consistent and fair evaluation, but even then, I would still have concerns.

Firstly, I would have to be informed of all of my personal data which would be in use in these systems. As well as this, I can imagine being very wary and paranoid of almost any piece of coding I submit to repositories, or even any messages I send in company messaging services. The lingering thought of whether any data point I am creating would be analysed, evaluated and possibly used against me is not comfortable position to imagine myself in. Some may argue this is a good thing, as employees would take extra care and caution in their work and what they submit. However, I think there needs to be a certain level of trust and confidence between both an employer and employee. If faced with this situation in the future, which I expect to be, I

will try to be very aware of what information I am generating for a company and how they use it.

Overall, I think engineering development measurement is a complex but truly fascinating area to explore into. It will be interesting to see how it evolves as software, its future prevalence and popularity and the reaction of most software engineers to their constant evaluation.