Monday, 20 November 2017

Measuring Engineering

Conor Wallace
Student Number - 15329699

Objective

Deliver a report that considers the process in which software engineering progress can be measured and assessed in terms of measurable data, an overview of computational programs available to perform this work, the algorithm approaches available, and the ethical concerns surrounding this kind of analytics.

Report

Recent decades have seen a rapid growth in data from increased communication, information and technology systems. The increase in data has not been sought after but has, in many cases, been a by-product of the exponential increase in the use of technology. Professionals are becoming a scarce resource in society as technology becomes our source of intelligence for an ever-increasing number of problems. Software engineering and the development of these platforms are no exception. Everyday interactions have had to change over the years to adapt to the dynamic demands of software and the increased dependence on professionals for tasks that cannot be mechanized. In the 1970's the planning and control of software was introduced with studies such as the 'Mythical Man-month' stating that throwing personnel at a delayed project could be more of a hindrance than a help. The results of this study highlighted that increasing personnel resulted in the additional division of the problem and increased the need for communication, consequently, productivity fell. The time delays were due to getting others up to speed and then the time-consuming task of combining the now numerous and fragmented submissions of code. This was the first real conclusive outcome from software engineer monitoring.

From the 1980s onwards, it was not just the speed at which software was produced that was an important recordable factor, it was the reliability of the software that was of upmost importance. The speed and reliability of software development became imperative as society became ever more reliant on its output, telephone exchanges could not make errors and moon landings had to be sure to be successful. Software became more of an industry than an accessory and the production process had to follow. By the 1990s another variable inevitably came to the forefront as the efficiency of software became a key aspect of development. Software had to be compact and precise as well as uniform as growing developments and resources saw programs interacting as well as working alongside each other to solve ever more complicated problems. Monitoring code for these three variables was, and still is, a necessity.

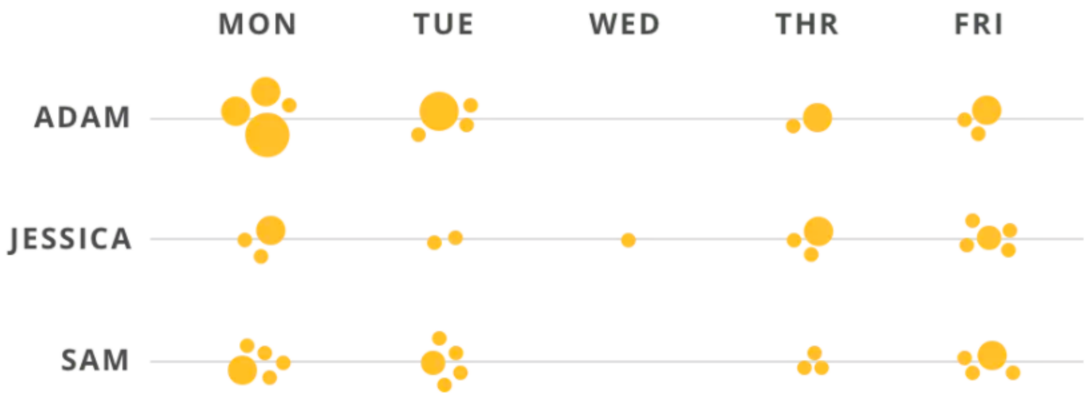
Measuring software is not an exact science as there is no single variable which can rank or rate production levels. The rate and type of production can also vary depending on which stage of development the software is in. Productivity is one of the toughest challenges software engineers face, many use a range of complex metrics to assess productivity while others use none. The following are a list of five developer metrics essential to all software managers.

Lead Time
Prediction of development lead time is necessary for all managers. The constant flow of work and tracking of task completion dates is essential to not only maximise the use of the skilled developers employed, but also to keep morale high. Unrealistic completion dates can result in stressful working environments and reduce productivity. Interestingly, the software team who can guarantee a quick delivery time is rarely the most desirable development team. When being monitored on production time it is easy to skip corners, leave out testing and collaborate code, limiting the usage, just to make deadlines. Early completion can in fact be more of a concern than late completion for many developers. Based on this, I submit that the relevance of lead time is limited. Quick production is rarely a top priority for many clients although accurate prediction using previous lead times for similar developments is a key requirement.

Code Churn
Code Churn is arguably the most relevant measurement tool for software engineering monitoring. It examines the lines of code added modified or deleted over a period. Large churn rates can be a sign of an issue in development or a programmer developing in an exploratory nature, investigating and discovering what small changes in the code will do. Monitoring of churn code will encourage developers to produce structured, well developed and compatible code in order to avoid returning to make changes later. Churn rates will uncover those with high production rates but low quality outputs, who may be causing more hindrance than good. This measuring metric's ability to encourage well developed code while uncovering numerous issues that would otherwise go unnoticed to the naked eye is, I believe, what makes it such a valuable resource.

Impact
Impact is a measure of the effect a code has on a project. When adding code are critical methods being implemented to link key components of the software? These methods are instinctively harder to implement so the impact will therefore be greater. Adding one hundred lines of code to one method which is not called throughout the package will have much less impact on software than a smaller change that makes multiple insertions and deletions throughout multiple files. Impact can be a useful measurement to compare developers, determining the capability of an individual. Within a team the developer with a larger impact is likely to be the stronger programmer. Impact is a good way to consider the cognitive load put on the developer.

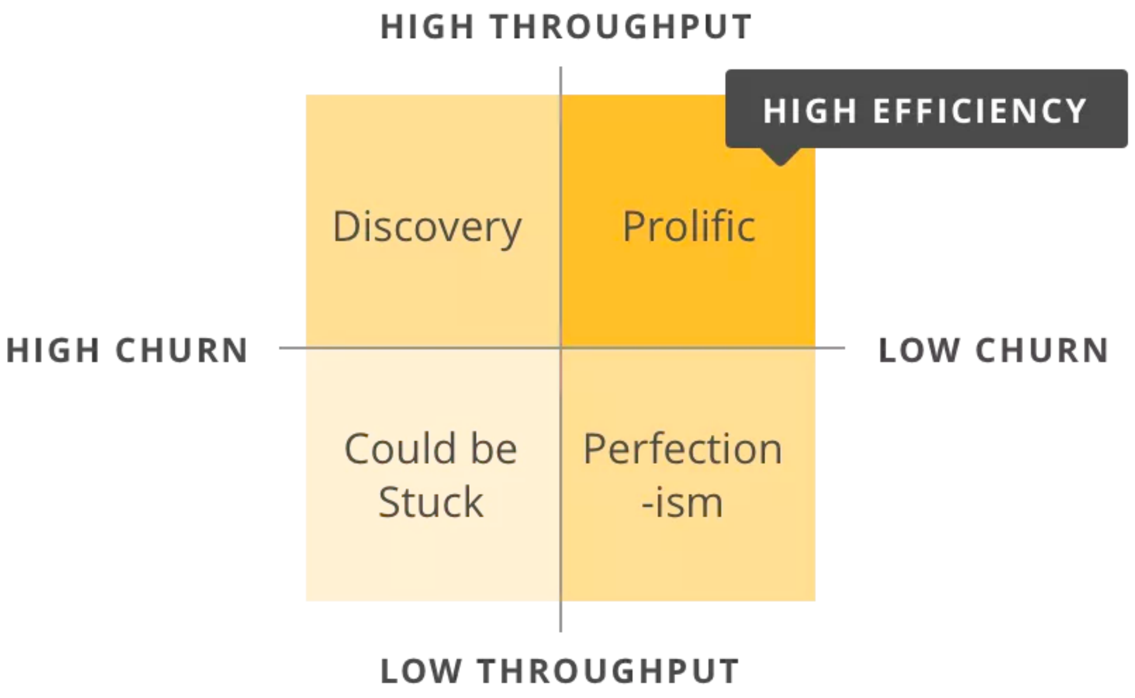
Active Days
Active days is the record of the days and times when developers contribute and commit the most code. Active days can show slump times during the year, month or day which is helpful for managers when planning rest periods and deciding on deadlines. Active days is a very simple concept although I personally consider it one of the most interesting to monitor. It is essential to see what motivates and influences a team of developers. Whether it be used to determine if team meetings and collaboration of ideas aids progress, or to define the average time it takes for progress on a new project to decline, it is valuable knowledge. Active days can be mapped, as seen below, to show projects which developers are most involved in as well as mapping daily productivity.



Appendix 1- GitPrime Active days graph

<u>Efficiency</u>

Efficiency is the measure of the amount of a developer's contributed code that is productive, generally balancing coding output against the code's longevity. Efficient code has a lower churn rate and therefore provides a greater business value. Different types of engineers will have varying rates of efficiency depending on their goal. The most prolific programmers contribute lots of small amounts of code with low churn rates resulting in high efficiency which compares to those with the opposite, indicating that they may be having difficulty with the task at hand.



Appendix 2- Diagram showing metric efficiency

Computing Analysis

With the quality, efficiency and impact of code becoming such a significant factor in the software development industry, a gap in the market has opened for an analytic service to process this data. The above analytical measuring metrics are examples of some of the measures used to monitor the code being committed. Developers have noted a demand for software that can scan commits for bugs, security leaks and even test the code for variables that developers may have missed. Every code commit has the ability to improve or reduce the quality of the code, with software to monitor those commits businesses can ensure only clean, healthy and well tested code is merged with their software systems.

The majority of these software platforms run on top of other code storage platforms such as github, sourceforge and gitKracken, allowing code to be stored externally away from physical on site servers subject to potential damage. This online commit service also facilitates constant monitoring and the collection of large data pools measuring the time, size and even regularity of commits. Data which can be very useful to managers and even personally for each developer.

Examples of some of the leading software development analytics companies are-

Code Climate

It is said that 'you cannot improve on what you do not measure' which is where code climate is essential for improving productivity, efficiency and impact of code commits. Code Climate supports many languages, technologies and frameworks and is used by many of the big players in the field such as Pivotal, jQuerry and New Relic. It prides itself on being very stable, well maintained and displaying an easy to use UI. Working off GitHub, a platform which is already used and well known to many software developers, it is an easy transition for many developers to make and could serve simply as a feedback service to many already committing to GitHub repositories. Code Climate provides a report on churn, cognitive complexity, cyclomatic complexity, duplication and maintainability. All very useful factors to review when trying to ensure a team leaves code better than when they found it and that progress is sought by all developers.

Some downfalls reported by Code Climate users include that there is no support for objective C and compared to its competitors Code Climate appears to be the most expensive tool, although it is free for open source projects. Currently Code Climate is only available as a chrome plug-in but they are working on safari and Firefox support. Taking all of this into account it is clear that Code Climate is a very useful tool when programming, this is supported by its extensive use in the industry. I believe it is Code Climate's clear feedback system and auto GitHub synchronisation that make it one of the leading software development analytic packages available.

Codacy

Codacy has many features similar to those found in Code Climate. However, Codacy's ability to define issue based goals to improve your code base causes it to stand out. This enables users to select code analyst patterns that matter to them as well as changing parameters to customize the code analyst experience. Codacy is a small company that is expanding quickly often releasing fresh features to its customers, some of which are big players in the field, with the likes of PayPal and Adobe using their services. Codacy provide lots of security measures when analysing commits, assigning strange values to private API's removing the possibility of omission errors which can lead to unexpected app behaviour. Code duplication review is also a very useful feature of Codacy.

Areas to be improved in Codacy include some incomplete documentation with reports that images are hard to read and interpret, as well as the amount of information presented being insufficient at times. Codacy uses visual representation of data very regularly and graphs and images presented can at times be hard to draw exact figures from. That being said, Codacy do support C++ as well as many other languages. Additionally, the huge flexibility thanks to disabling/enabling patterns on whole packages or even selected files makes it a unique and highly sought after automated code review tool.

Scrutinizer

Unfortunately, Scrutinizer doesn't stack up well against other review tools available in the market. However, it does show signs of an impressive API and is one of the cheapest solutions on the market, the API is only available to users who subscribe to the most expensive package. Unfortunately, even when subscribed to this premium package the software still suffers from some very avoidable errors such as issues reading certain file names along with the ability to change the configurations of checks but no way to restore the default settings. These small issues when using the software combine to reduce its value and effectiveness. Developers require a system that is compatible and fluid to use when providing feedback, users should not have to develop and name files in a certain way for the review software to work. Automated reviewing software should aid production rather than hinder it, which is where these small errors can cause problems. All considered, the documentation is good along with a site dedicated to the current status of the services and a feature of filtering issues by users are services new to this software only provided by Scrutinizer.

When evaluating automated code review software packages, it has become clear that it is those packages which can go unnoticed to the developer that are the most desirable. A leading factor in a good review package is compatibility and ease of use. It is this reason that I consider Code Climate an industry leader. The ability to be used simply as a GitHub add on (which many developers already use to commit code) means it is not an inconvenience to the developer. Detailed reporting and monitoring of code structure as simply a bi-product of committing is what makes Code Climate stand out to me. The addition of C objective compatibility and the ability to use the software on any internet browser will ensure Code Climate is the software of choice for many developers regardless of the cost.

Algorithmic Approaches

The algorithmic approaches to this type of work is a deep and complicated topic with many opinions. Previously many would have assumed that work of this nature, analysing professionals, was a job that only a highly skilled and intelligent professional themselves could do. Jobs of this nature with so many variables would be reviewed on a judgement basis with code being manually reviewed and read through, a judgement which no software program could recreate. Previous views on AI software completing tasks like this was that they had to replicate and make decisions like humans, judgemental and opinionated decisions that computers just could not replicate

It was in the 1990s when AI systems were coming to the forefront that software engineers were trying to model systems to make decisions like experts in their field. It was thought that software could be made to model a world champion chess player at the time, but never to beat him, in 1996 this changed. Software was developed to use its brute force processing power to beat the world champion. The software could compare hundreds of moves in a matter of seconds for each scenario on the board to estimate which one was best suited, something a human brain could not do. It was this technique that beat the world champion who did not have the ability to compare all of the moves on the board but instead relied on his expert judgement to do what he thought was best. Patrick Winston has subsequently stated that 'there are lots of ways of being smart that aren't like us' this is the direction in which AI systems have now been lead, not to be smart and opinionated like humans but instead to use brute force and substantial processing power to be smart in different ways.

The advancements and change in the perception of AI has lead people to consider the fact technology can make decisions and provide feedback that was once limited to experts. As algorithms get more and more progressive it seems that machines can take on increasingly cognitive tasks. Traditional professions such as teaching, law and medicine are now moving to online algorithmic platforms such as Khanacademy, Legalzoom and WebMD which have thousands of visits each day. Research has shown that some of the best known legal brands are not traditional law firms, but in fact Legalzoom, a document drafting service offering legal advice online. On EBay sixty million disputes a year are resolved online without lawyers, using an E-mediation platform. Notably, this figure amounts to forty times as many civil claims as are filed in the English and Welsh justice system.

It is the decomposition of these systems that makes this work possible, for example, the breakdown of software analysts' results in the divided measurable variables noted previously. Code Churn, efficiency and impact are all factors that could previously only be judged or ranked by a professional with no numerical measurement. For this to change developments in technology had to enable programs such as Codacy and Code Climate to divide the problem into smaller logical chunks that a computer could compute. This confirms that computers may not be able to recreate our way of thinking but they are smart in their own way and we can adapt problems to make use of that. The decomposition of problems, along with the use of binary comparisons and algorithms, enables technology to take over and assist in jobs that were previously strictly professional. This movement in professional work changes the commonly known one-to-one professional feedback system to now a one-to-many relationship.

Some of the algorithms used to solve the divisions of these problems depends on the data that is collected. Programs can use predetermined known data to develop an understanding of the field they are providing analysis for, not understanding in a theoretical way, but in a logical and calculated way. Developers regularly use two types of system learning techniques, supervised and unsupervised. For pre-determined data sets supervised learning is required, the system will run algorithms such as linear regression, logistical regression and deep learning algorithms. With this data and complex analysis, the software will be able to determine a set of constraints in which it can measure an unknown variable to predict which class or grouping it belongs to. This can help with patient diagnosis, legal advice and business decisions all based on predetermined data, similar to the experience accumulated by a professional over time.

Unsupervised learning, the method of distinguishing distinct groups from undefined data sets, is tool used more to help professionals than it is to replace them. Unsupervised learning uses algorithms like k-means, priority Algorithms and Hierarchical Clustering to determine set clusters in the data. These complex advancements show that the answer is not to replicate the human mind's way of processing, rather it is to change the perception that only professional judgement can be used to advise and review.

The question then stands, what limits should be placed on the extent of computation carried out?

Ethics

In modern times, it is routine for our every movement and action to be tracked, data is produced from almost everything we do. With increased technology comes increased data but where is the limit? The time software developers are typing, browsing or idle can all be monitored along with now the standard and quality of the code they produce. Is this progressive for development or can this in fact frustrate developers and result in over working staff? Personally, I feel there should be restrictions on how strenuously this data is monitored. Averages recorded over lengthy periods of time are of course acceptable and when used to improve an individual's technique can only be a positive thing. However, when short term goals become daily requirements and performance is solely based on figures, this is where issues can arise. Figures are the output of logical parameters, so just as logical parameters cannot replicate human intelligence, it is impossible for human intelligence to be judged solely on logical measurements.

Measuring software developers is in no way dissimilar to managing anyone else, managers are there to help resolve issues, avoid non-development work, and help employees achieve their personal career goals. The difference with software developers is, now that large volumes of data are readily available for all to see, all work is tracked, stored and analysed in detail. Software development is one of the first industries for this to occur in. The question is, if this data will remove the manager's objectives noted above, can managers still provide the service expected of them? With analytical analysis, some might even argue that the manager's job is redundant, I strongly disagree. Managers must respond and listen to developers as well as just exert control. Data has an amazing ability to explain and prove things, but its black and white responses are uncompromising and it is not right to hold developers accountable to these outputs. There are issues in the field with regards to the freedom to explain data, developer reports can show an array of results for developers who will inevitably produce the same output. Some may start fast and finish slow, others may put together a rough solution before nailing down the requirements and occasionally people may specialize in pulling commits and software together not producing much of their own code at all. Are any of these strategies wrong? They will inevitably each provide the software that the client requires while individually flagging different deficiencies within the code. Monitoring software using computational platforms can encourage a rigid approach to programming, reducing the freedom and creativity of the developer in an effort to avoid bad practice being flagged, something, I believe, is wrong.

When faced with the issue of what data is acceptable to track many opinions and variables must be considered. When considering the data required to track software developers, questions arise around privacy of the developers. If software developers wish not to be tracked but then submit a final functioning program is this an issue? Developers may not like having managers looking over their shoulder constantly. Does this constant monitoring hinder the creativity of developers? Measuring the likes of code churn and lead time will, I submit, reduce a programmer's freedom. Developers will not feel they are allowed to trial new non-conventional solutions to problems as doing so would flag errors in their development process. This could be regarded as a hindrance for the industry and only applicable to development as we currently know it. Developers also propose the question who owns this data? Writing and developing code will obviously help and assist a programmer's knowledge base and just like in any other field, the more you do something the better you get. With programming, if a developer was to use these skills and regurgitate a program at home which was used at work who then owns this code? It is copyright practice that while employed any code you develop, that falls within the scope of a task you have been given in the work place, belongs to the employer. If you are to develop a program in your free time unrelated to any assignment at work, then of course this is your own code. In contrast, any work that falls within the "work- made-for-hire" scope belongs to the employer or client paying for its development. A fair process in my opinion.

In conclusion, the addition and involvement of monitoring software engineers progress in the above ways is productive for the industry. Using software to measure efficiency, churn rate and impact will ensure code is produced to the highest standard while minimalizing errors in the code. It will also help to identify people's strengths in coding, whether that be efficient method combining, exploration or high security programming, this is what I believe the progress reports should outline. Developers' effort and work ethic should not be denoted by these progress reports and all developers should have the opportunity to explain results. The software's understanding of the situation outside of code input is absent and so it cannot be regarded as a replacement for a manager. The areas in which there is a possibility for future concern include the ownership of these development reports, is it the employer or the employee who is allowed to distribute or claim ownership of this data? Will it become common practice for employers to ask for data on previous commits as an employment process? Software engineering is the first industry to come across this method of monitoring due to the abundance of data available, with increased technology and online submission platforms I feel many sectors will follow suit. It will be interesting to see the advances and issues this may cause.

Index

http://asp-software.org/www/misv_resources/business-articles/who-owns-the-code/

https://www.netguru.co/blog/comparison-automated-code-review-tools-codebeat-codacy-codeclimate-scrutinizer

http://www.informit.com/articles/article.aspx?p=30306&seqNum=5

https://blog.gitprime.com/6-causes-of-code-churn-and-what-you-should-do-about-them/

https://www.itproportal.com/features/comparison-of-automated-code-review-tools-codebeat-codacy-code-climate-and-scrutinizer/

http://www.dandemeyere.com/blog/code-climate

https://blog.gitprime.com/why-code-churn-matters/

https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/

https://www.youtube.com/watch?v=Dp5_1QPLps0