CS3012 Measuring Engineering - A Report

Name : Cormac O'Dwyer **Student Number :** 15325054

Given Task : To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Trinity College Dublin
The University of Dublin

Table Of Contents

1.0 Measurable Data - Intro

In this topic I will discuss the practicality of measuring the performance of software engineers along with the many ways it can be implemented. For starters what do we mean by “Measurable Data” in relation to software engineers performance, well measurable data in this case is often called a ‘software metric’ and I will be using this term for the remainder of this report. A software metric is the measure of some software characteristics which are countable or quantifiable, e.g how many lines of code, the time taken, test coverage, ect.

It is generally considered the collection and use of software metrics are seen as a net positive for the overall progression of a project and daily running of a large software company. It helps in the focusing of projects and tasks and can also highlight any areas that could be improved in order to have a more efficient development. An issue arises when deciding what software metrics are considered important and worth measuring to base a developers performance off of. Should you determine their worth based off the number of lines of codes they have written or the churn rate of the code ? Relying to heavily on these metrics often leads to the disregard of the important human side of software development, like how well they interact and communicate with their team.

At first there was considerable push back from developers about the collection and analysis of their work performance, this was mainly due to the inefficient nature of the data capture when it was first implemented. The first iterations to capture software metrics often required the engineers to manually input their data, mainly in the form of some sort of spreadsheet. Analysis on this captured data would often have to be done manually also by some sort of supervisor or manager. These forms of data capture where seen as time consuming and detracted from the time that could be spent on the actual progress of the project, also due to the manual nature of the data input

it was heavily prone to human error and intentional miss information in order for the developer to seem more productive than they actually where.

However the industry in its current state has leaned in heavily to the use of software metrics in order to cut down on the development time of projects in order to reduce costs and streamline a company. Unlike previous iterations its not an issue of the data collection rather an issue of *what* data to collect. We are in an era with an unprecedented amount of data and it is often a more difficult task to decide what metrics are important and which are a hindrance. Often the heavy reliance on the wrong software metrics to evaluate an employee can lead to a loss of moral and annoyance. Below are some of the more prominent software metrics.

1.1 Lines Of Code : This is the measure of the amount of lines a specific developer has written. This is easy to measure and automate but has many issues. As it is only counting the amount of code written and not its performance it can be wildly inaccurate as it will count white spaces, comments, dead code, ect.

1.2 Code Churn : This is the measure of the amount of code that has been rewritten within a specific time period. This means if a developer has a high churn rate they are consistently going back and reworking parts of their project which is supposedly finished, this can be a good indicator that they are having issue with the given task and may need help.

1.3 Code Coverage : The measure of how much of the program is run when it is being tested. A poor code coverage usually means a lack of proper testing which can mean a higher chance of running into bugs or errors later on in the development.

1.4 Developer Time : Usually the measurement of how long a developer spends in their coding environment on the project. Useful to see if they

complete tasks periodically or in bursts, also if they started the given task early or waited until the end to get their work done.

1.5 Code Coupling : The measurement of the degree of which different classes are related to one another. Usually the higher degree of coupling means the harder it will be to implement changes later in the development cycle without affecting the relation it has with other parts of the code base which can lead to errors.

1.6 Cycle time : The measurement of the time take in order to make a change to your software and implement that change into production. Useful in order to see how quickly issues are being addressed during your development. Projects with a high cycle time will often lead to a high build up of bugs and errors in your project.

1.7 Team Velocity : The expected and measured time taken on a given task, usually on agile teams. An estimation on how long each task of the project will take is done before development starts and the actual time taken is then compared with this to gauge the development speed. This can be good in order to maintain momentum on a project so that it gets completed within an acceptable time frame.

1.8 Open/Close rates : the measurement of how many issues are reported and then closed within a specified amount of time during production. Less about the data and more about the trend it conveys. If there is a trend of many reports but few closes then there is a big issue in development.

2.0 Computational Platforms - Intro

In this section I will discuss what methods and software is used in order to collect these various software metrics. First I will look at how these platforms where first implemented and then compare and contrast them with more modern approaches.

2.1 PSP : One of the first platforms that was actively used was PSP, or Personal Software Process. As mentioned briefly in the previous section this platform offered one of the first methods to collect software metrics, but it all had to be done manually. PSP was very much like a simple spreadsheet, it required each individual developer to manually input their data. Due to it being a spreadsheet it allowed for a wide range of metrics to be tracked , and it was often left to the individual company to choose which data they want to track. A developer would be required to keep a record of the various software metrics their company needed, be that time estimations, defects, size estimations or even interruptions. Then said developer would have to manually input that data to PSP at a regular interval. The biggest advantage of PSP was that it was flexible allowing any inputted data to be tracked, but that flexibility also left it very fragile and vulnerable to human error. Data could very easily be altered or simply entered incorrectly meaning a large overhead was required if you were to maintain any sort of reasonable quality data set.

Another drawback for this method of data collection was that it offered no analysis on that data. Much like the data collection the actual analysis all had to be done manually by reviewing the spreadsheet. This was usually done by a supervisor or manager and this brings along the same issue of human error into the equation. Overall PSP was good in theory, the flexibility of the platform allowed for options on what data you wanted to track but that also brought its biggest weakness and that is human error.

2.2 Leap Toolkit : The leap toolkit, or Lightweight, Empirical, Anti-measurement dysfunction and Portable software process, took the PSP concept one step further it addressed the data quality problem highlighted by automating the data analysis. Leap would automatically perform various forms of analysis on the provided data set and could be used to calculate the size of the product , defects various users found , estimations on time along with checklists and various other options. It also provided a few quality of life features such as personal repositories for each individual developer , allowing them to easily bring their personal data from project to project. While this was an improvement on the analysis aspect of things the developers still had to manually enter in the required data which was still prone to human error. The manually entry method was increasingly becoming less viable as it was felt the manual entry of data was a hugely time consuming task and was detracting from actual work getting done.

2.3 Hackystat : The next platform I will discuss tried to address this manual entry problem and it was hackystat. Hackystat changed its approach to the collection of software metrics. Originally with platforms such as PSP and Leap the method was to choose which metrics were important to the running of your project and then require your developers to enter in specific data pertaining to said metrics. Hackystat took the approach of gathering the data first and then performing a specific analysis on said data set to infer what metric you wanted. For this to happen hackystat had to automatically collect the data without any need for manual entry by the developer. This was done by the use of collecting large amounts of data in the background and storing it in a cache. If the user was offline it would wait until they have reconnected before it is uploaded to the web server. Hackystat then could perform various analysis on the data set and graph it in a manner that was easy to understand and accessible by individuals or teams.

The largest issue hackystat ran into was the push back from the developers themselves. The automated background collection of data about your own employees raised a lot of ethical questions that I feel we are still grappling with to this day. The broad amount of data that could be collected and easily viewed by your managers brought a sense of unease. Even if the process is automated at the end of the day it is your peers who will review and pass judgements based on that sensitive information.

2.3 Jira : For our modern example of a platform I will be looking at Jira. Like most modern implementations, Jira has moved away from the pure data collection and analysis and instead has integrated many features that are useful in modern software development, be that bug tracking, issue tracking or project management into one location. So instead of the data being collected in the background and then the analysis highlighting certain areas that need improvement it has now been directly integrated into our workflow.

An individual creates their own account in which they can work on various projects which are ran by their team leaders. Every person has a clear outlined task with goals that they are expected to reach within a specific time frame. Anyone on a team can view their peers work, being able to see when they committed, if they are behind on work, if there are bugs that need to be fix ect. Many of the various software metrics mentioned in the first section of this report can now be tacked and displayed automatically and can be easily viewed. As stated previously the ethics for this practice are still debated to this day but it is clear from the shift from background collection to integration into our daily workflow that this practice is here to stay for the foreseeable future.

3.0 Algorithmic Approaches - Intro

As I mentioned a few approaches in the first section I thought it best to see what algorithmic approaches are likely to be implemented in the future. There is a huge amount of financial resources tied up in the development of software and a large portion of that development cost is down to the errors cause be us, humans. Because of this, huge amounts of time and effort go into the testing and maintenance of your product and nearly every stage of production. This is why a machine learning has become a promising avenue for the future of development efficiency.

With machine learning we can use the historical input and data to be able to predict results. With machine learning it will be given various learning algorithms of fault prone and none fault prone to be able to develop predictive models. While this can be hugely beneficial it is also in the early stages of development and will probably not be implemented for some time, but so far machine learning can be categorized into three categories. That being Reinforcement learning , Supervised learning and unsupervised learning.

3.1 Reinforcement learning : Much like how we learn through trial and error, reinforcement learning is the process in which the machine is given a long term task to complete and then proceeds in a step by step fashion to try complete said goal. If a step is made that leads to a negative response it will retrace its steps to the option it had before triggering the negative response and tries another option. If it works correctly it is given a positive response and proceeds to the next step. It will continue this process of trial and error until its long term goal is achieved and now knows what steps to take in order for that goal to be fulfilled.

3.2 Supervised learning : For this process of machine learning we ‘train’ the algorithm to predict some well-defined output based on the input data. In this

process we give the algorithm a test data set and is asked to map it to some desired output. E.g given photos of two different types of objects and is asked

to separate them. It is then going to create the appropriate rules that maps the input to said output. Once this is done we then go and select the most precise model that has been created. Once this 'training' is done the system should be able to assign an output to which it has not yet seen.

3.3 Unsupervised learning : very much like supervised learning in the way it is tasked to create the most precise model to complete the given task but instead it is not given any desired output to be mapped to. This is used to uncover any hidden structures but is rather difficult to implement and as such is not used nearly as much as supervised learning. It may be used in the initial phase of supervised learning as the internal structure of the data can provide additional information on how to better reproduce outputs.

Its of my belief that machine learning will play a huge role in automation in the coming years and will likely see them have a large influence in the developers work flow in the future.

4.0 Ethical Concerns

Data collection and analysis is not just a concern for software developers but for most individuals across all forms of industry. The increasing norm this day and age is the never ending pursuit of increasing workplace efficiency in order to maximise productivity and profits, and the most widely used practice for this is to collect data on exactly how every step of your work is done in order to highlight errors or bad practices that can be improved upon using a similar platform to the ones mentioned previously. This has lead to nearly every workplace being monitored by some sort of automated data collection, and behind every one of these monitoring softwares is a company or team that has made a conscious decision on what data to track and what data to not. This has inadvertently pushed software developers and tech companies into a position where they could negatively affect a large amount of people. It raises questions about what data should you be allowed to store and analysis and what crosses that privacy boundary for that individual.

It is not just the collection of data that is troublesome , some of this data can be sensitive personal information that can have a detrimental effect if it were to be known, be that your personal information , bank details etc. This also raises the concern even if all of it was ethically viable , should we collect such information if we can not say with absolute certainty that the information will be handled in a confidential and safe manner. It is all too common to hear about data breaches, where huge amounts of personal information is accessed leading to untold consequences.

With the current rapid pace of technological advancement it has led to the new workplace territory being defined by dramatic and swift technological change. I feel this rapid development has led to a culture of trying to get a product out quickly to be the new ‘innovation’ within the industry. I feel this has lead to products being released where they have not fully realised what sort of human impact it will have. And as the technology improves so does the

knowledge gap between developer and consumer. This leads to a situation where the vast majority of the users using your product do not know the full

ramifications that come along with it. We have seen this recently with the facebook data breaches, along with the breaches we have seen a huge surgance of consumers becoming aware of what exactly they are presenting on such platforms and how much of that information is tracked. As this gap increases it is of my belief that the ramifications of these practices will only become more and more frequent.

In conclusion, yes the collection of data has been immensely beneficial overall but with the rapid development and integration of new technologies has lead to a culture of implement it first and worry about it later and the ethical concerns will only become more significant in the years to come.