Measuring Software Engineering: A Report

Methods of assessing the software engineering process:

The software engineering process is nuanced and complex. As such, it can be difficult to accurately assess in terms of measurable data. There exists many different approaches to tackle this, each with their own advantages and disadvantages. Before attempting to make judgements on a team or individual's performance it is important to ask exactly what factors are essential to successfully carry out all the tasks they are responsible for. Each method can elucidate some of this information but more often than not several methods will be used in order to gain a more informed understanding of the process.

Measuring Productivity

Time Active

Time active is perhaps the most easy to understand metric. As the name suggests it is a measure of how much time a team spends actually contributing to a codebase as opposed to planning or completing other tasks. On it's own it is a rather vague metric as the type and scale of tasks can vary within projects. It is also important to take into account any time spent bug fixing code that has been rushed as this can cost more time in the long run than a more thought out approach.

Lines of Code

The most rudimental means of measuring the software engineering process is assessing the number of lines of code a developer or development team produces in a given time frame. The idea is that a more productive software engineer can produce a greater amount of code at a faster rate. This method is fundamentally flawed and is rarely used in proper analysis. A poor programmer may commit inefficient code which

uses many lines but provides very little useful functionality. Where lines of code of code or KLOC(Kilo Lines of Code) can become more useful is when it is part of another metric such as errors or defects per KLOC.

Code Churn

A more sophisticated take on the analysis of committed lines of code is looking at code churn. This involves measuring the lines of code that are thrown away or replaced during the software development process. High levels of code churn can be a sign of a team or individual who is uncertain about the requirements of their project or how to implement a working solution. Of course most projects will have a certain amount of redundant code so some code churn is to be expected. However, when code is rewritten at a rate that deviates far from the average it can be a warning sign that something is not right in the software engineering process such as a vague specification or poorly trained engineers.

Open/Close Rate and Pull Requests

Another way to measure productivity is by seeing how often production issues are reported and how quickly they are addressed. If production issues start to pile up then it can indicate that the team is unable to maintain their codebase and handle the increasing number of bugs and errors that can occur as a project grows and expands. Similarly, if the number of open pull requests begins to grow then the team may be unable to deal with the volume of code that is being written and more time may need to be invested in deploying existing code.

Velocity

Velocity is a metric used in agile development models. It is the measurement of work done over a given time frame. A unit of work is decided upon by the team or management along with an interval known

as a sprint. The velocity is simply the number of units the team can complete in a single sprint. The measurement is almost entirely useless on its own or in comparison with the work of another team or individual as it is completely relative. What it is used for is estimating how much time a piece of work should take based on how much time it took to complete similar work in the past. It is a widely criticised metric as it is dependent on the accurate assessment of how many units a specific task is worth. This means if a simple task is assigned too many units the velocity will appear to be high even when the team's throughput is low.

Cycle Time

Cycle time is the time it takes for a team to implement and deliver a change to the project. This is arguably one of the most important metrics in measuring a team's performance as a fast cycle time indicates a highly optimised and effective development process. When cycle time is high code can be shipped to market much more quickly which is an incredibly valuable asset in all of software development.

Lead Time

Lead time is a similar metric to cycle time. Instead of being a measure of the time it takes for changes to existing code to be implemented, it is the measure of how long it takes for an idea to go from conception to a deliverable product. In other words, it indicates how long it takes for a product to go from nothing to being put to market. This is just as valuable a metric as cycle time as the time spent getting a project off the ground can be the most expensive time especially for younger companies who need to get products to market in order to start generating income.

Measuring Reliability

Mean Time Between Failures

MTBF gives an indication of how reliable a piece of software is. It is the predicted time between failures of the delivered system. Software failures happen in almost all deployments so it is useful to have a prediction of how often this occurs. Obviously being able to minimise this figure is hugely advantageous as it means less time and resources are spent on repair and recovery. This metric should be factored in to all analysis of time/resources involved in developing and maintaining a piece of software.

Mean Time To Recover/Repair

MTTR goes hand in hand with MTBF. Knowing how often a system goes down is not useful information on its own as it gives little to no indication of the ramifications of said system failure. If a system failure is catastrophic and requires a huge deal of work to repair or is difficult/impossible to recover data from then it becomes far more costly to maintain. On the other hand, if a system failure is quick and easy to recover from then it is often regarded as more of an inconvenience than a deal breaker.

Crash Rate

Crash rate is another simple performance metric that can be used to track how reliable software is. It is the frequency in which a program fails with regards to how often it is run. It is not as insightful as the other two metrics as it does not take into account the real life use of the software however it can be used as a quick and easy way to analyse system performance.

Measuring Performance

Cyclomatic Complexity

Cyclomatic complexity is a measure of the number of linearly-independent paths through a program. Having lower cyclomatic complexity can be hugely advantageous as it means less tests are required to validate code as well as reducing the risks involved in making changes to the code. Code with low cyclomatic complexity is often more easy to read which improves the long term usability of the code making it more valuable.

Measuring Security

Endpoint Incidents

Security is a hugely important factor in all areas of software engineering. The easiest and perhaps most insightful analysis of security is the measurement of endpoint incidents. This is simply the number of times a device involved in the running of the software has suffered a malicious attack such as a virus. Despite being a very easy to analyse metric, it generally provides a pretty accurate view of the security qualities of a system.

Computational platforms to measure the software engineering process:

For all the methods of assessing software engineering, there are as many ways to compute and process the data involved. This enables us to analyse large scale projects with many contributors and big code bases.

Version Control Tools

Almost all software engineering tasks are completed with the help of version control software. Tools such as git not only provide the tools to keep track of a codebase but often also include tools to analyse the activity on that codebase. When multiple users are making multiple commits to multiple branches all on a single repository it can become very messy very quickly. In order to extract data for specific users or to track something specific like commit frequency, simply looking to the statistics provided by the git repository can be incredibly helpful.

Code Review Tools

There exists a host of code review tools that work with all sorts of languages and frameworks. They are used to analyse code and extract data which compliments some of the methods mentioned previously.

Codacy

Codacy is perhaps the most widely used computational platform in software engineering analysis and can be incorporated into popular software version control platforms such as Github. It provides automated analysis of uploaded code including useful metrics such as cyclomatic complexity, code coverage and duplication. These can in turn be used to make important judgements on the performance of a team and the quality of the code they produce.

Algorithmic approaches to measuring the software engineering process:

For all the data we can collect, there exists numerous algorithms which are used to calculate specific attributes of the software engineering process using this data.

COCOMO Model

The Constructive Cost Model is a procedural software cost estimation model which computes software development effort as a function of program size in terms of KLOC. It is based primarily on two key parameters. They are Effort which is the amount of work required to complete a task and Schedule which is the amount of time required to complete the task. There are three variants of the COCOMO model which provide different levels of detail and accuracy. The basic model is very limited as it only takes into consideration the two primary parameters. The intermediate model is more detailed and accurate as it takes into account cost drivers and the advanced model is even more detailed and accurate again as it takes into account the influence of individual project phases. The COCOMO model is widely used in industry and as such there is a lot of information and discussion surrounding it. This makes it a useful way of assessing a team’s performance as well as to predict the cost of a project. However due to the nature of the calculations it is unsuited to small scale projects.

Putnam Model

The Putnam model is an alternative effort estimation model. It works by collecting data from existing projects, working out the Effort and Size of those projects and fitting them to a curve. From here, any future effort estimates can be calculated by matching with the size of the project in question on the graph.

The ethics of analysing the software engineering process:

As with any analysis of people, there are a host of ethical questions which must be answered before collecting, processing and interpreting their data. Despite there being many clear benefits for a company to have access to the data and tools outlined above, it is not always in their worker's best interest. In a time where data privacy is becoming increasingly scrutinised it is important that companies are weary of the implications of performing these analyses.

Software engineers, like anyone else, are concerned with their privacy. It could be argued that by gathering data like time an employee logs coding for a specific problem provides a useful insight into how problems are tackled and the breakdown of work within a team or department. However, many would deem it a step too far to be tracking an employees actions to such a high degree and thus an unethical method of analysing the software engineering process. Though less obvious, similar issues can arise in the use of almost all methods and computational platforms used in this kind of analysis. This can be seen when using platforms such as hackystat which provides similar statistics to codacy but must be embedded into an IDE. Instead of monitoring an employees output upon commit, it continuously assesses their work. This approach is clearly more invasive and though more abstract, should still be called into consideration when looking at the ethical implications of an approach to software engineering analysis.

It is important for a company to have happy and satisfied workers. Happy software engineers produce better quality end results, are less likely to suffer from burnout and are more likely to stay with the company for extended periods. Having unethical practices such as invasive analysis are a sure fire way of reducing happiness in the workplace. For this reason, companies should take the ethical considerations of all their actions, including analysis of the software engineering process, very seriously.

Conclusion:

In conclusion, there is a wide range of methods and tools used in the measurement of the software engineering process and each have their own advantages, disadvantages and ethical concerns. It is essential for any large tech company to perform analysis on the software engineering process however it is just as important that this analysis is carried out in a way that is both accurate and impactful. The use of computational platforms, algorithms and well defined methods should combine to provide insight into every aspect of the process and throughput of an individual, team or department. However it is not a black and white issue and achieving this is often more complicated than it seems.

For one, there exists no perfect metric for analysing software engineering. Though lines of code or cycle time may seem like totally valid metrics, the fact that the world of software engineering, even within a single company or department, is so changeable and the quality of the output can vary so much all of these metrics will be inherently flawed to some degree. Advanced algorithms can give the illusion of accuracy and dependability, however as they are fundamentally based on these same imperfect metrics they too will be flawed. Ultimately, an understanding of the specific workload a team or individual is under used in tandem with the methods outlined above is what is required to perform meaningful and accurate analysis.

There can exist many roadblocks to achieving this insight, not least of which is the ethical concerns in performing this kind of analysis. A company must always weigh up the advantages and disadvantages to analysing its software engineering process as well how deep they need their analysis to go. A surface level analysis will not be as impactful as a more in depth, hands on approach. However, it is not always the right decision to invest resources in monitoring and measuring every little detail as you will be sure to encounter issues along the way. Striking the right balance is the key to effective software engineering analysis.

Bibliography:

Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

http://www.citeulike.org/group/3370/article/12458067

https://stackify.com/track-software-metrics/

https://www.geeksforgeeks.org/software-engineering-cocomo-model/

https://www.codacy.com/

https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

https://codescene.io/docs/guides/technical/code-churn.html