Measuring Engineering

PATRICK JENNINGS

Patrick Jennings
16322057

Table of Contents

Patrick Jennings
16322057

Introduction

In this report, I will be looking into how software engineering can be measured and assessed. In particular, I will be concentrating on four aspects:

1. The methods of measuring this data in terms of data and metrics
2. The various algorithmic approaches that accompany each metric
3. The various platforms available to perform this work
4. The potential ethical concerns surrounding these practices.

First however, I think it prudent to define a few terms which will be addressed many times throughout this report.

Software Engineering is the design, development and maintenance of software. It came about as a role as a result of the poor quality of software products that were being produced, and the vastly increasing and changing user requirements. Software engineering encompasses all of the programming part of the development of a product, but it also covers such things as requirements gathering, feasibility studying, and system analytics.

A **Software Engineer** is someone who performs the job mentioned above.

Before I get into analysing the various ways in which we measure software engineering, it is important to address the reason as to why we do it. To quote the software engineer Tom DeMarco:

You cannot control what you cannot measure

If we do not measure the performance of software engineers, then as a collective we cannot consciously improve. It is in the interests of both engineers themselves and companies therefore to measure and track performance in order to reliably increase said performance over time.

Patrick Jennings
16322057

Methods of Measuring the Software Engineering Process

Agile Metrics

Agile software engineering in an approach which targets the evolution of requirements and solutions through collaboration between engineering teams and the customer. The philosophy of agile development is to be adaptive rather than predictive, to iterate rather than use the waterfall model, and to prioritise code over documentation, with the idea that good code should be documentation unto itself. Its focus on adaptive planning, early delivery and continual improvement means that engineering teams themselves tend to have a lot more of a say in the decision making process, rather than just following orders from higher up. This means that ways to measure these teams are needed to ensure the best work possible is being done. The metrics used don't describe the software itself, but rather they give information on the teams themselves.

Security Metrics

In this day and age, security is a very important factor for consumers and businesses when evaluating the quality of a software product they buy and use. With all of the recent focus on data leaks and malpractices in technological giants such as Facebook or massive multinationals such as Equifax, producing and maintaining secure software is of upmost importance to every business with a software engineering team. Therefore, we need to measure how these teams are performing in their security obligations, by looking at how resilient their software is, and how well the team responds to security incidents.

Size-oriented Metrics

Size-oriented metrics are useful in their ease to calculate. Once we determine what constitutes a line of code, we can make measurements based on this. Due to the nature of different programming languages having different ways to compute the same thing, which might take more or less lines of code, this is not a useful metric when we want to compare software written in different languages.

Function-oriented Metrics

Function-oriented metrics measure how much functionality software provides. However, the functionality of software is quite an abstract concept, and there is no metric that directly corresponds to the functionality of a piece of software. Therefore, we must instead measure the number of function points software provides. Function points describe the business functionality provided. Importantly, it allows us to compare software written in different languages. Unfortunately, function points are not easy to calculate, something that I will touch on later, so it many software engineering teams choose to not use this metric.

Production Metrics

Production metrics try to quantify the work being done by a software engineering team. While the agile metrics go a way to measuring the software engineering of an agile team, they are more than anything comparing a team against itself. They don't give a good idea of how overall efficient a software engineering team is. Production metrics are all about figuring out if the work being done by software developers is the best work that they can be doing.

Algorithmic Approaches Available to Measure these Metrics

Agile Metrics

Velocity

Velocity is the measurement of the amount of work done by a team over a period of time. The work itself tends to be measured in story points, and time is split up into sprints. In software engineering, a story is a particular piece of work that needs to be done, and is assigned to a particular team. Story points then, are measurements of how difficult a story is to create. For instance a story with four story points is expected to take twice as long as a story with two points. Sprints are short periods of time, usually about two weeks, but can differ from team to team, during which items of work are ideally both started and completely finished. A velocity chart takes this data, and displays the work being done by a team as they move through sprints. This is useful for teams looking to plan their future sprints and how much work they can expect to complete. It also allows them to reflect on passed sprints to see what went well or badly depending on performance.

Sprint Burn-Down

Sprint burn-down is a method for measuring the performance of an engineering team over the course of a sprint. Typically it is represented as a line graph with an ideal line going straight from the total number of story points that were predicted to be completed in the sprint at day one, to zero story points at the last day. Whereas velocity is used to measure a team's performance over many sprints, burn-down is used to measure performance of a singular sprint.

Release Burn-Up

Release burn-up is another method for measuring the performance of a software engineering team over many sprints, but constrained to a single release. A release is a large number of story points which are expected to be finished by a certain day, at which they will be released as a finished product. This method of measurement tracks the story points delivered of a release period. Similar to the sprint burn-down, the release burn-up is typical represented as a line graph with the ideal line going from zero story points delivered at sprint zero, to all story points completed at the final sprint in the release. Teams can use this to assess how they are performing over a release, and judge whether they are going to meet the release deadline. It can also be used as a method tracking feature creep. Feature creep is when extra work is tacked on to a release, and can sometimes result in work originally planned to be cast aside and not be completed in time for release. The release burn-up chart can assist teams in managing this.

With all of these methods of measurement for agile teams, it must be noted that they are intended to be used to measure a team against itself, and not against others. Each team may have different ways of splitting up their time, and judging how many story points each piece of work is worth. Standardising these measurements goes against the philosophy of agile development.

Production Metrics

Lead Time

Lead time, also known as cycle time, is used to measure the time taken for an idea to propagate from the requirements planning phase all the way through to deployment. In contrast to the use of sprints as a method of measuring time, lead time deals with calendar time. For dealing with customers and people not familiar with the software development process and terminology, this allows for data-driven estimates to be given in weeks and months, universally understood metrics of time. By analysing the data presented by

measuring lead time, teams can identify points of the process slowing them down, and hopefully decrease overall time taken to deliver a project.

Active Days

Active days is a very simple measurement of how long a software engineer spends actively coding. If too many people on a team are spending their time doing administration, planning or other tasks, I could be indicative of a need for more programmers, or a re-evaluation of how the work is being done. It can also highlight unexpected interruptions to a project and the costs thereby associated with them.

Assignment Scope

Assignment scope is a measure of the amount of code a software engineer can actively create, maintain and support. For team planning purposes, it is a good signifier of when teams need more pairs of hands, or when more support projects can be assigned to a team.

Code Churn

Code churn is the number of lines of code added, deleted or modified over time. We can learn a surprisingly high number of things from this simple measurement. For instance, if code churn is increasing as a project nears release, there is a heightened chance of errors and bugs. Instead, the aim should be for code churn to stabilize and level out as release approaches. It also points to the efficiency of a software developer. Someone with a low churn is likely to have very efficient code, and someone with high churn probably has very inefficient code. Thirdly, code churn can indicate the development process of an engineer or a team. A steady code churn indicates an agile process, where the whole development process gets iterated on over time, whereas regular spikes indicate a waterfall model.

Mean Time between Failures

Mean time between failures (MTBF) is measured as the average of the times between each failure. A high average means that software is performing well, whereas conversely a low average means that the software is failing to perform and requires work. The calculation of this is as follows:

$$MTBF = \frac{\sum(\text{start of downtime} - \text{start of uptime})}{\text{number of failures}}$$

Application Crash Rate

Application Crash Rate (ACR) is the number of times an application fails divided by the number of times it was used. Obviously, the more something is used the more likely it is to break. Just looking at the frequency of failures doesn't give us a good idea of how likely

software is to break for someone. An application used by millions has a much higher chance than one used by a few hundred.

Security Metrics

Mean Time to Repair

Mean Time to Repair (MTTR) measures how long a piece of software remains down before the error is fixed and it can be used again. This allows software engineering teams to assess their response time to errors along with the severity and complexity of errors. The calculation for this is as follows:

$$MTTR = \frac{\sum(\text{start of uptime} - \text{start of downtime})}{\text{number of failures}}$$

The longer an application is broken or vulnerable, the more likely it is that a serious security breach can occur, and sensitive data leaked.

Flaw Creation Rate

The frequency at which flaws are created is an important piece of data that indicates if the current development process is viable long term. When we compare this with our MTTR, if flaws are being created at a higher rate than they are being fixed, then over time the project will accumulate more and more bugs, which greatly decreases customer satisfaction and increases the chance of there being a serious security incident.

Defect Removal Efficiency

Defect removal efficiency (DRE) measures the ratio between defects and errors found during production, and after product delivery. It is given by the formula:

$$DRE = E / (E + D)$$ where DRE = defect removal efficiency, E = errors after product delivery, D = errors before product delivery

The closer DRE is to one, the more efficient the error finding is in production. If a team can successfully detect and eliminate defects before shipping them to customers, they reduce the chance of a security breach. Teams with a high DRE can be trusted more with handling software engineering that deals with sensitive information.

Size-Oriented Metrics

Lines of Code

Possibly the most basic method of measuring software engineering is measuring lines of code written. This can be extrapolated out to measure the frequency of errors, and defects per thousand lines of code (KLOC for kilo lines of code), or the cost per KLOC.

As I mentioned earlier however, this falls down when we try to compare projects written in different languages, as there are different ways to do different things in different languages, which will not use the same number of lines. It is also generally accepted that using lines of code as a metric for measuring software engineering is a bad idea. Depending on how the lines are counted, it can encourage copy-pasting of code rather than the use of classes and functions, can encourage the adding of superfluous comments to inflate numbers, and can discourage refactoring.

We also have to consider the extra costs of having more lines of code. The more code there is, the harder it is to understand, which means more time is wasted on this rather than actual development work.

Function-Oriented Metrics

Function Points

Function points generally come from two categories - business transactions that can be performed by the user, such as inputs and outputs, and data that the software can access and store, such as internal files. These are then weighted based on their complexity and by their classification type to give them a point score. Function points can be used then to measure the productivity and quality of software in a similar way to how LOC are used, by looking at errors, defects, and cost per function point, among other things.

The advantage of using function points over LOC is that it is much harder to fudge the numbers to increase performance. There are sets of criteria for working out how many function points a project has.

Computational Platforms Available

There exists a plethora of tools to measure software engineering. The desire for companies and managers to quantify and improve productivity has created a lucrative market for

companies that can accurately and efficiently calculate insightful and relevant metrics. Here is an overview on a few of these tools:

GitHub

GitHub is version control web-based tool owned by Microsoft. It offers all the features and functionalities of Git, along with many of its own services, including bug tracking, feature requests and wikis. There are reportedly 28 million GitHub users, and 57 million repositories, 28 million of which are public. It was written in Ruby on Rails and Erlang and launched in April 2008 by Tom Preston-Werner, Chris Wanstrath, P. J. Hyett and Scott Chacon.

GitHub offers a publically available API that can be used to request a whole host of data about publically available repositories, including but in no means limited to commits, connections, pull requests and repository contents. Anyone that has access to a repository can pull down all its data and then run evaluations on that data. There even exists projects publically available on GitHub for the extraction and analysis of this data, so one does not necessarily need to have a great knowledge on computer programming to begin measuring software engineering for no cost.

GitPrime

The more lucrative possibility for using data from GitHub to generate insights is in private businesses. GitPrime is one of a number of tools that make use of APIs on version control websites like GitHub, GitLab and BitBucket in this way. It processes the repository data for the company, creates meaningful reports on key metrics, and provides insights into the software engineering of the company.

Semantic Designs

There also exists tools outside of git environments for measuring software engineering. Semantic Designs produces one of these. It instead uses language specific metrics for a number of supported languages that analyse the code itself and come up with insights and measurements from the data gathered. It also can analyse the files and directories of large projects and gather data from there. Thirdly, they support the creation of custom metrics for different models and for unusual programming languages.

Ethical Concerns

There are some very obvious ethical concerns when it comes to measuring the work of software engineers. Due to the nature of the work of a software engineer, a large part of their day is spent at their computer writing code. It is not unreasonable to assume that companies would be able to track what their employees are doing at pretty much any point during the day. One of the first things that comes to mind when writing about this is the various controversies Amazon has faced from tracking the productivity of its warehouse employees. I recall not too long ago there was uproar over the way Amazon tracked how long their employees spent in the bathroom, or spent talking to their co-workers. People were horrified at the notion of having such fine details of their day known to their employer. And just recently it has come out that Amazon has patented the technology for a wristband which will track the movement of their hand, and sends vibrations if they are doing something wrong. Once again, this has caused outrage over how closely Amazon are tracking their workers.

It is probably safe to assume that if Amazon is comfortable with being so blatantly obvious about how it tracks its floor staff, then many if not most or all tech companies track their software engineers in a similar way. The thing is, they are not doing it to spy on their workers. Amazon doesn't care that you went to the toilet, and Google doesn't care about your morning coffee break. What they do care about is productivity, and the more time you spend doing these things, the less time you spend programming and making money for the company. However it is the fact that this data does exist, and the fact that our privacy is being invaded, even just a little, that is disconcerting and uncomfortable for people. It is certainly something that needs to be examined; if it is right for people to lose out on privacy for the gain of the company.

We must also consider the fact that the conclusions drawn from the statistics and data gathered might not always match reality. Like I mentioned earlier, using LOC as a metric to measure software engineering is ill advised, as it is possible for a developer to make themselves appear to be better than they are by playing the system. If extensive research and testing is not done before using a metric or datum to evaluate the performance of an employee, it is possible that good, efficient workers will not get rewarded as they should for good work. It is the responsibility of managers and companies to ensure that something like this doesn't happen.

Conclusion

The options today for measuring software engineering are vast. The metrics, algorithms and tools I covered in this report only touch on a small part of what is available for both companies and individuals. The desire for companies to measure, quantify and improve the performance of their employees is understandable. The value of a software engineer today is extremely high, and one can't expect companies to try and get anything less that the best value for their money. As long as people stay aware of the human factors involved in collecting this data, and the media and employees keep a scrutinising eye on those unscrupulous few who try to abuse the data they collect, I believe we do not have to worry too much. The measurement of software engineering in part and parcel of the job now, and rather than try to fight it, I feel efforts would be better spent trying to improve it in an ethical manner as much as possible.

Bibliography

- https://en.wikipedia.org/wiki/Agile_software_development
- https://resources.sei.cmu.edu/asset_files/TechnicalNote/2014_004_001_77799.pdf
- https://whatis.techtarget.com/definition/story-point
- https://screenful.com/blog/software-development-metrics-cycle-time
- http://ptgmedia.pearsoncmg.com/images/0131424602/samplechapter/0131424602_ch05.pdf
- https://codescene.io/docs/guides/technical/code-churn.html
- https://en.wikipedia.org/wiki/Mean_time_between_failures
- http://www.ques10.com/p/8370/explain-size-oriented-software-engineering-metri-1/
- https://stackoverflow.com/questions/3769716/how-bad-is-sloc-source-lines-of-code-as-a-metric
- https://www.totalmetrics.com/function-point-resources/what-are-function-points
- http://www.1000sourcecodes.com/2012/05/software-engineering-function-oriented.html
- https://en.wikipedia.org/wiki/GitHub
- https://www.gitprime.com/
- https://www.semanticdesigns.com/Products/Metrics/index.html
- https://www.nytimes.com/2018/02/01/technology/amazon-wristband-tracking-privacy.html