Measuring Engineering - A Report by Conor Thorne

Conor Thorne - 17337490

Trinity College Dublin

Measuring Engineering

Table of Contents

Introduction

The Measurement Software Engineering is a highly debated topic and one of incredible fascination to anyone with any interest in computing. Tech giants like Microsoft, Apple, Google, Amazon and Facebook are investing millions every year into developing and purchasing software that measures engineering productivity. The pursuit of data and analytics to increase productivity is of extreme interest to any software developer and in my opinion it is a captivating new frontier of research that I thoroughly enjoyed investigating.

In this report I am to give a view into how the software process is being measured using different types of measurements, the companies offering services to do this work and the algorithmic approaches being developed. I also explore the ethics regarding measuring software engineers productivity.

Types of Measurements

The value of measuring software development productivity has long been debated. While researching I found many developers who were disgusted by the idea that one could possibly measure development quantitatively -

“Any kind of quantitative metrics for software developers tend to actually reduce overall productivity. They also have negative impact on motivation, and will eventually drive good people out.” — vartec (top 0.74% contributor on stack overflow)

And while the above view may hold some merit, it is very clear that measuring software productivity metrics is a massive science that tech giants are investing millions of dollars into every year. There are countless leaders and CEOs who are using high tech software to measure the performance of the engineering process.

The focus on this section is to outline the actual metrics that people are interested in and for the most part I object from indicating if they are useful or not, unless there is a way in which they have been improved upon eg. using code churn instead of lines of code.

It is possible to split the software engineering process measurements into two seperate method categories; quality based metrics and quantity-based metrics.

Quantity Based Metrics

- **Progress Against Estimates**

Measuring progress against estimates is basically comparing how many tasks are being completed within the agreed estimates. The necessity here is to;

A Have fair and reachable estimates and

B) Understand where estimates have been exceeded for good reason for which the developer can not / should not be blamed for.

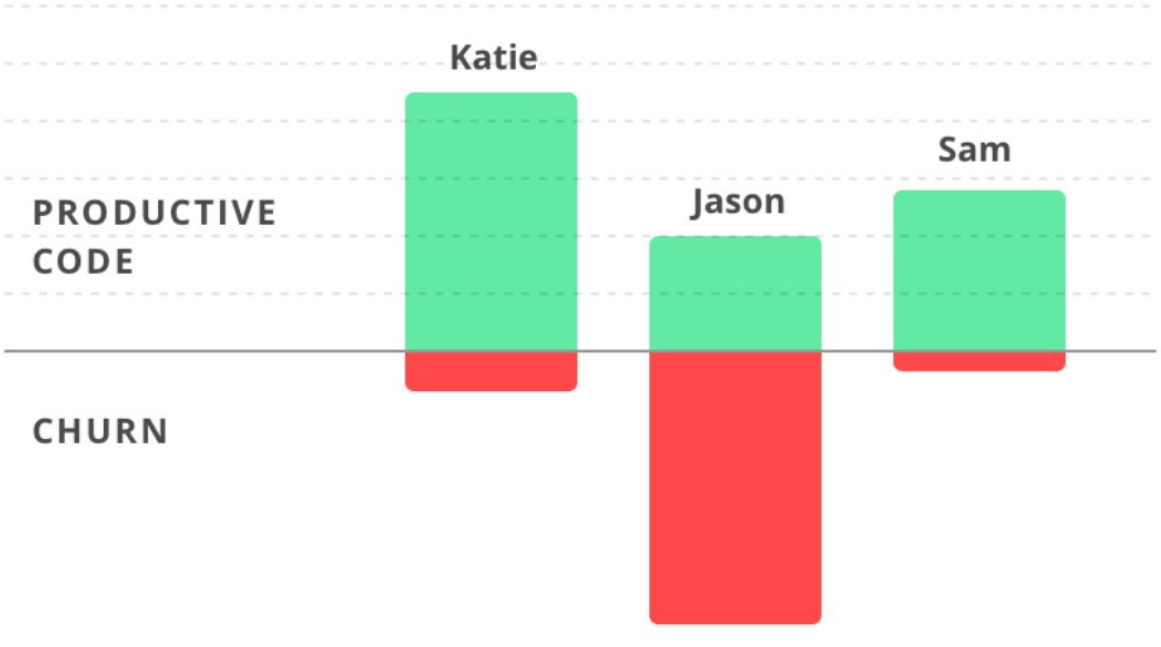
It is extremely necessary to not allow for padded estimations where developers are always reaching their goals easily.

- **Lines of Code / Code Churn**

The productivity of an engineer or team of engineers could potentially be measured by the amount of lines of code written. This is a highly controversial measurement and for good reason. In my opinion it is extremely apparent that the quantity of code is by no means an assurance of quality

*“Measuring programming progress by **lines of code** is like measuring aircraft building progress by weight.” - Bill Gates*

Although measuring lines of code alone has been observed as unreliable an alternative to this would be measuring **code churn.** Code churn is when an engineer rewrites their own code and it seems that it is a better way of measuring the productivity of written code. This is because you are enabled to examine what code is actually being shipped.



- **Speed of Developer**

The speed in which a software developer or an entire team can complete their given tasks is a potential area of measuring the engineering process.

- **Number of Commits**

This is a metric based around the number of times an engineer submits his code to the team. It is especially easy to calculate if the team is using a software development version control like GitHub.

Quality Based Metrics

- **Code Quality**

Some Engineers argue that the quality of code can be measured by examining it's test coverage and efficiency (services like "CodeClimate", which I will examine in further detail later, give grades on repo efficiency). In Agile, a method of programming, programmers write tests before they write their code so technically speaking the quality of code could be measured by the tests it covers. Code quality could also be measured by using the simple metric of;

the number of bugs discovered pre-launch **-** number of bugs discovered post launch.

- **Testing**

Number of tests written and the test coverage of a program is thought by some to give an idea of how efficient the code is and it's overall quality.

- **Technical Debt**

Technical debt is the additional work needed to finish the software development, it is caused by many factors; poor conception, poor scheduling, bad development practices, code coverage, code complexity etc. This is a measure used by some to investigate how inefficient a process is.

- **Peer Evaluations / Self Assessment**

Although I felt it would be a difficult metric to include I found that peer feedback and self assessment were two quality metrics that continued to re-surface throughout my research. Tech giants like Microsoft, Google and Facebook all use self-assessment and peer reviews to measure how their engineers are performing. Google's "Promotion Program" works by having engineers create a promotion application which includes details like detailed self assessments of performance and contribution. In Facebook and Uber employees find peers that will submit a review of their performance over the previous cycle.

Overview of Computational Platforms

In this section I Intend on examining the different platforms that are being offered to companies today that offer insights into engineering team productivity. I will do this by choosing four popular analytical companies and doing a case-by-case profile on how they approach the following;

- Data these companies observe
- How they perform analytics / computations on the data
- Exploring the Infrastructure used by these companies

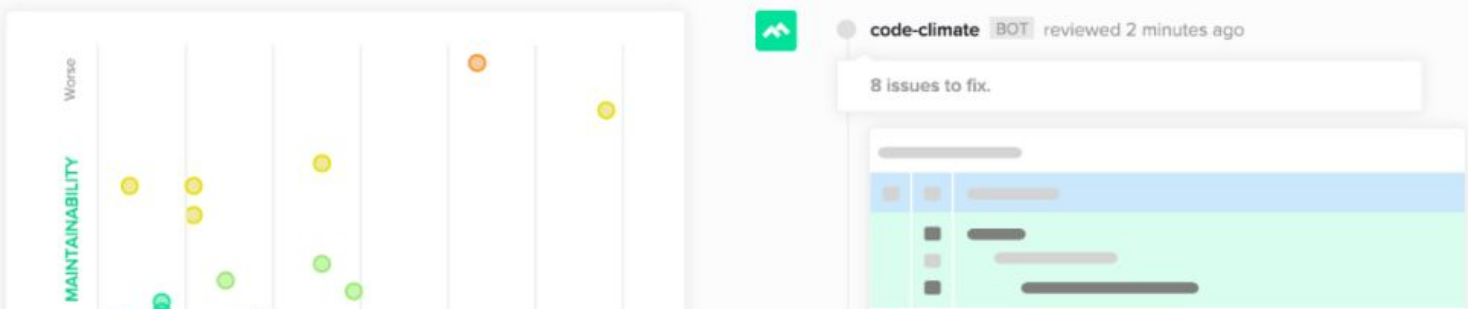
Although I found it difficult to pinpoint the exact methods companies were using to analyze data, as for obvious reasons this would be kept private, I did my best to examine the types of measures they were focusing on and what data they created from these metrics.

Code Climate

Code Climate is an open, extensible platform for ensuring *"code health"* by analyzes trillions of lines of code per week. Code Climate was founded in 2011 and offers two main tools; "Quality" which is an automated code review tool and "Velocity" which is a product that analyzes data from GitHub repos and provides analytics and custom reports to give a perspective on how an engineering team is working.

Quality

The main services of "Quality" is automated test coverage results and scoring the maintainability of code. The data that "Quality" collects is mainly diff test coverage, frequently changed code and duplication. Code Climate uses this data to analyze the code's technical debt and code quality. It seems they calculate technical debt by focusing on files that are being changed consistently and pose maintainability threats due to low test coverage. Code quality is measured by correlating areas of high code churn with negative scores and returning analytics that indicate areas that should be focused on.



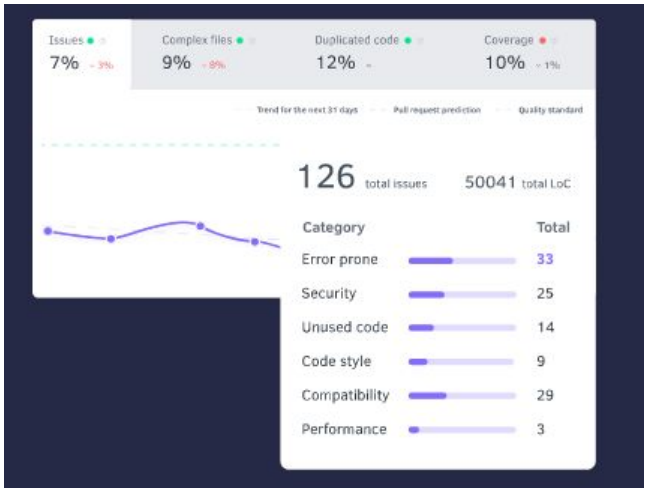
Velocity

Velocity claims to provide analytics on where engineers are stuck on certain problems and information on how quickly a team is moving. Velocity gathers data on commits, pull requests and tickets to pinpoint areas of hidden bottlenecks and where work is stuck and which engineers are having trouble and need help.

Codacy

Codacy focuses on analyzing the quality of code. It is mapped to your repository and files on GitHub. The services Codacy advertises are returning report analytics on the Code Quality, Technical Debt and Security Risks of code in a repository.

In order to create analytical reports on code quality it seems that Codacy ask the user to select a preferred “Code Standardization” and Configuration Rules. Codacy then take all code in the repository and use these two parameters set “Quality standards” and return reports which indicate if these standardizations are being applied to all the code being submitted. They also use parameters like duplicate code, code coverage, unused code and code performance to gather analytics on code quality.



Codacy also give analytics on security risks of code and potential problematic areas of code.

Gtmhub - OKRs (Object & Key Results)
This company caught my eye by the list of companies that employ its services; Google, Facebook, Amazon, Salesforce, Intel, Microsoft, Netflix, Twitter. Nearly every Blue-Chip I had ever heard of. OKR is a goal setting and management platform for businesses and it claims to integrate data from over 150+ sources to provide analytics on feedback on productivity.

OKR measures data from business systems (over 150+) from Slack to Jira to provide individual reports to members of engineering teams. OKR also provides an API called REST that can be integrated into the team's GitHub repositories to provide analytics.

BetterWorks
BetterWorks creates software that acts like a video game to try boost worker productivity and boost worker-to-worker relationships. The big difference between BetterWorks and companies like Codacy and CodeClimate is that it is not based solely around software engineering but more so worker productivity. It works by having workers and their bosses set short term and long term goals and log their progress on a digital dashboard that all workers in the company can see. If progress towards one's goal is being achieved their "tree" will grow and flourish. If they are failing their goal the tree withers and dies.

It was a relief to see that the companies offering services to help measure the engineering process all used types of measurements that I had focused on before. I thought the four companies all contrasted well together, with Codacy and CodeClimate being quite similar, Gtmhub being similar but distinct in its approach and finally BetterWorks being the overall representation of how these models work.

Algorithmic Approaches

Functional Point Analysis (FPA)
FPA is a method of *Functional Size Measurement,* Which is where it measures the functionality delivered to users of the software based on the user's view of functional requirements. Instead of measuring the internal view of the software eg. code quality FPA focuses on the user's external view. FPA was originally developed by Allan Albrecht in the late 1970's at IBM and further developed by IFPUG.

FPA measures functional requirements in terms of the:

- Processes ie. processes that user can perform using the software
- Data Groups ie. data groups the software can store/access

Calculation of Function Point (FP)
Step 1.
F = 14 * Scale

- Scale varies from 0 to 5 according to the *Complexity Adjustment Factor (CAF)*

Scale	**Complexity Adjustment Factor**
0	No Influence
1	Incidental
2	Moderate
3	Average
4	Significant
5	Essential

Step 2. Calculate Complexity Adjustment Factor (CAF)

CAF = 0.65 + (0.01 * F)

Step 3. Calculate Unadjusted Function Point (UFP)

Function Units	LOW	AVG	HIGH
EL	**3**	**4**	**6**
EO	**4**	**5**	**7**
EQ	**3**	**4**	**6**
ILF	**7**	**10**	**15**
EIF	**5**	**7**	**10**

- Multiply each individual function point to corresponding values in TABLE.

Step 4: Calculate Function Point

FP = UFP * CAF

Once Function Point Count is measured you can use the measure to develop the following project performance indicators:

- Scope of software development product
- Quality indicators
- Productivity
- Performance

Machine Learning Approach

Some have decided to apply machine learning to measure the software engineering process. One great example of this is the work done by Semmle Inc.

Semmle apply machine learning to version control to measure software engineering development productivity. Semmle's LGTM ("Looks Good To Me") analyses over ten million commits by over three hundered thousand developers to approximately thirty thousand open. LGTM'S QL is a query language that treats code as data. Each commit is treated as a new snapshot that QL uses hundreds of queries to identify problems with the code quality and code output (ie. effort put into code).

Productivity metrics are developed from these quieres.

Code Output

To develop code output productivity metrics are developed from these queries by using code coding time models to create datasets that associate code changes with estimated coding times. The datasets are then used to train a model that given any code change can predict the coding time required to produce it. This model is then used to represent the coding time profile of the standard coder.

Code Quality

LGTM quieres detect a wide range of potential problems (called alerts) in code including problems like run-time bugs, technical debt. LGTM then uses these kind of alerts as a proxy for code quality, the more alerts triggered by a piece of source code the worse its quality.

Code Quality Comparison Chart from LGTM.com

In this graph the x-axis represent the lines of code and the y-axis is quality. Each bubbl is an open source Java Project. The highlighted project, apache/flume, has 107 alerts and a quality score of 0.22 (Grade D)

The ‘quality score’ for each category is si = 1 − Fl,i(ni) where l is the project’s LOC and ni is the actual number of alerts in category i. The interpretation of si is straightforward: it measures how well the project performs (0 = worst, 0.5 = as expected, 1 = best).

Ethics of Measuring the Software Process

Abstract
The topic of measuring the engineering process is, as mentioned before, highly controversial. Ethical concerns lie at the heart of this controversy. Is it right to try measure engineering productivity? I feel the question requires a deeper answer than yes or no, I think it would be best to outline the problems engineers have with measuring productivity and possible solutions to these matters.

Data Protection
In some cases it is quite possible that the data being collected by teams may infringe upon the data privacy rights of software engineers. With the implementation of GDPR (General Data Protection Regulation Law in the EU) in 2016/2017 the data privacy of individuals carries great weight. The idea that a lot of worker's data is being constantly analysed and used for reports may make some developers uncomfortable.

It is in the hands of the developers of these programs and the managers to ensure that the data being collected is totally transparent and passed all guidelines recognized by policy laws like GDPR.

Productivity Boost via. Public Embarrassment
One ethical concern I would have is the shame that workers could feel under by report systems like CodeClimate and BetterWorks. Imagine a scenario where you just had an awful month, some terrible things happened to you in your personal life or you were having some mental health issues and as a result your productivity in work began to slide. Years ago it would have been a simple case of talking to your superior and explaining your situation but for me it seems in some cases we are sprinting into a potential distopian future of enforced productivity by public embarrassment. Business models like BetterWorks displaying your failures to the entire company is a way to force workers to hit their achievements via the threat of a public display of failure, for all your co-workers to see, in the case that you do not manage to achieve these goals. For me this is boosting worker productivity at the cost of worker happiness and it is the wrong way to motivate. I feel companies like codeClimate which limit the report results to your engineering team or superior handle it much better. I don't believe that worker efficiency is an entire workforces concern but perhaps your immediate peers and superiors.

Pressure on Workers

I definitely think that workers should feel some amount of pressure to deliver. You should take pride in the work you do and be proud of your work. Although one moral concern I would have with measuring worker process is the pressure that could be put on to workers to reach goals no matter the cost. While I believe workers should feel some pressure to deliver I do not think that workers should be working through lunch, staying late every night or staying up late at night dreading their productivity report the next day, knowing they haven't achieved their goals. Something to exemplify this is highlighted in a New York Times article
Where in a trucking company instead of using GPS tracking to judge driver efficiency, as they had done previously, they posted scoreboards in break-rooms and mailed bonus cheques to

spouse at home to increase competitive pressure from work and from home. This led to drivers take lunches to continue their work in order to increase productivity. While this example is not exactly a software engineering team the idea and the result could easily be applied to one. This is a situation where motivating and attempting to increase worker productivity using malicious methods results in an overall worse situation for workers. This is a major ethical concern with measuring productivity, if it is done in a tenacious manner it could actually violate worker rights.

Authenticity of Data Analytics - Listen to the God

In one of my favourite poems "Mirror" by Syliva Plath the author makes the comparison between a mirror and God, suggesting that the way we worship mirrors makes it a little God. In many ways I feel the same about the computer. We live in a society where people consume the information from their phones or computers without any questioning of its authenticity, information from an email for example could be photoshopped and posted online to an audience who would immediately believe it is authentic.

The reason I explain this theory is because I strongly believe a similarity can be drawn between the readiness we have to accept information from computers and the danger of accepting the authenticity of measuring worker productivity. It is the duty of managers to ensure that the analytics from measurements are valid because if they are not the consequences could be costly and irreversible.

Conclusion

The aim of this report was to clearly outline the measurements being used to analyse software productivity and the ethics surrounding this objective.

As I mentioned before, I truly believe that this activity is extremely important to the future of software development and I hope that my opinion and research has been clearly defined within this report.

References

- **https://engineering.kapost.com/2015/08/you-can-and-should-measure-software-engineering-performance/**
- **https://www.jmoses.co/2019/07/08/software-engineering-manager-guide-measuring-performance.html**
- **https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/**
- **https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/**
- **https://blog.gitprime.com/why-code-churn-matters/**
- **https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology**
- **http://www.folklore.org/StoryView.py?story=Negative_2000_Lines_Of_Code.txt**
- **http://thinkapps.com/blog/development/waterfall-vs-agile-software-development/**
- **https://codeclimate.com/**
- **https://www.quora.com/What-metrics-do-big-tech-companies-Google-Facebook-Netflix-etc-use-to-track-the-performance-of-their-software-engineers**
- https://dzone.com/articles/unit-testing-youre
- https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0
- https://codeclimate.com/about/
- https://www.codacy.com/?utm_source=GoogleAds&utm_medium=ppc&utm_campaign=GoogleAdsBrand-Phrase&utm_term=codacy&utm_device=c&gclid=EAIaIQobChMI4rS5jIvT5QIVmK3tCh2Glgb5EAAYASAAEgIIOPD_BwE