Measuring Software Engineering

James Lunt

18323467

Introduction:

Throughout my research I have discovered the software engineering process is particularly difficult to measure appropriately however it has become increasingly apparent to me how important it is to somehow measure the work of a software engineer and a collective of software engineers similarly to how imperative it is in every other profession.

"If you can't measure it you can't improve it." -Peter Drucker

This report will consider the ways in which the software engineering process can be measured and assessed in terms of what measurable data can be used, the computational platforms available to perform this work, the algorithmic approaches available and the ethical concerns surrounding this kind of analytics.

Why do we measure?

1) Prompt Action

We need to measure so our decisions are informed and a team can figure out what is the next step to take and make a plan for success.

2) Goals and Alignment

All organisations and teams need some sort of way to create alignment to determine whether they are progressing towards their goals, to measure what they need to do and figure out whether they are doing it or not.

3) Advocation

Both employees and managers need measurements and numbers to point to as justification for change, for example if an engineer wants to change how a process is working it is extremely helpful and persuasive to have metrics that can back up his opinion.

4) Higher Purpose

For motivation it is important to see results shows. Measurements can give workers some self-actualization that there work is making a difference and progress is apparent.

Measurable data

Theres a range of metrics that can be used for measurement. Managers can assess a team based on their output such as new features and the quality and quantity of such features I.e project delivery. The success of a team could also be based off business metrics such as sales, customer feedback, profit margins and other measurements of how well the product does. This is a fair enough way to figure out how well a team is working; if the final results are good then the team or individual is doing well, if the results aren't so good then that is decent indication for change and re-configuration. However, if there is a problem with output then these statistics won't underline the source of the problem and even if the team is apparently succeeding the formula for success won't be obvious. Hence it is important for managers to seek visibility in other motions of the business such as refactoring, production issues, research, collaboration, maintenance, review and comments.

These micro-movements in software engineering come in several different forms but all are particular to the process of software engineering. They all are data points gathered from a team and individuals coding labour, as this is the work prominently carried out by the software engineer.

Before naming and describing these metrics it is important first outline what qualities are sought after in code. Bryan Helmkamp, creator of Code Climate outlines these well in his talk at GORUCCO 2016.

- Good code is **simple and clear.** Code should be elegant and not overengineered in order for readability and ease to change code in the future
- **Extensible** so the program can scale to more intense use and will last.
- **Well tested** as programs can break easily code must cover edge cases and protect the program from circumstances that won't allow the program to work
- Algorithmic studies would always suggest code must be **fast** as efficiency is imperative for a well working program.
- Sometimes we mightn't no why code was changed or what the initial problem with the program was. Therefore, code must be well **documented** with comments or extra documentation. A programmer might not even remember what his own piece of work does so documentation works well as a reminder and an enlightener to co-workers. Repositories such as GitHub and Bitbucket are essentially used as they provide great space for documentations and a record of actions in a repository such as the git commit history.

So how do we measure these qualities? Well the following metrics are currently used by software engineering teams however there level of effectiveness is certainly varying in fact Abi Noda, senior product manager at GitHub elaborates at his GitHub Universe 2019 talk that misused metrics are horribly destructive and metrics that seem rational aren't so.

"Not everything that can be counted … counts" -Albert Einstein

He goes on to point out the five metrics that are currently heavily used by modern tech companies to monitor their employees but considers them all flawed.

Commits

- These are basically the number of changes made to a code base in a repository. You can measure commit frequency and also size, this might allow you to figure out how active a software engineer is and it however it is only effective to spot the workers who are doing very little or those who are ineffectively overworking. Someone with a small frequency of small commits is doing little in comparison to someone with either a small frequency of large commits or large frequency of small commits although someone with a large frequency of large commits could be taking too long to create a solution. Commits are the measurement of change in a program however they have no indication to how well the change contributes but are decent for showing workers activity patterns.

Lines of Code

- Lines of code has been a classic although extremely archaic way of measuring software engineering. This metric is actually counter-productive if allowed, as mentioned before quality code is simple and efficient however if an employee is rewarded by the amount of code he writes, then he may tend to inflate his code which will make it messy, less elegant and heavy. Code works well in the artistic sense that little changes can make all the difference, similarly to how a few paintbrush strokes can make a painting rather than as many strokes as possible, a few lines of code can make a program rather than as many lines as possible. Lines of code is only useful to measure the size of a software system and how your codebase is changing but not for effort.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight" - Bill Gates

Pull Request Count

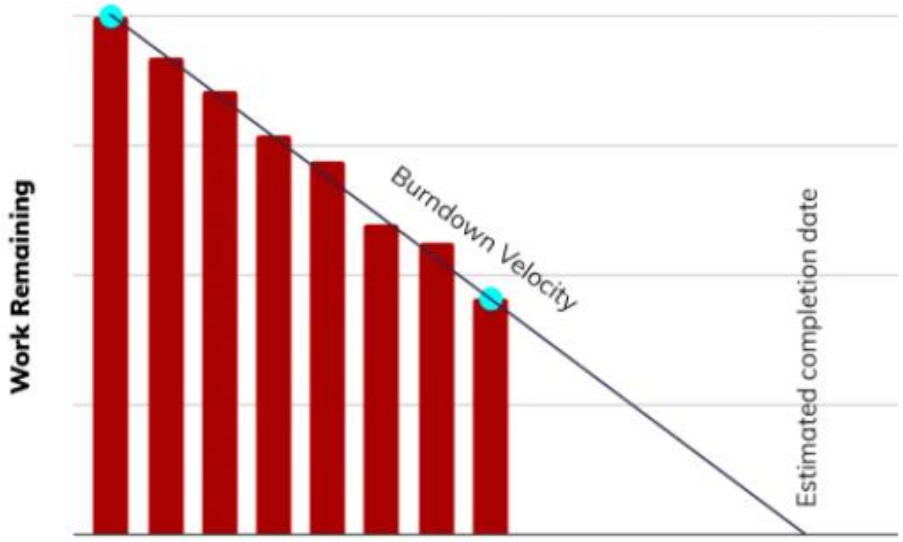
- Pull requests let you tell others about changes you've pushed to a branch in a repository. Once a pull request is opened, you can discuss and review the potential changes with collaborators. A useful feature however when used as a metric to measure productivity it doesn't factor size or effort required of work and can encourage negative behaviours like creating unnecessarily small and chunked pull requests. However, Bryan Helmkamp discovered that there is a sweet spot pull request size of around 400 lines where reviewers will be able to find most defects within this amount of code. If you were to order your team to create pull requests of about this size this would allow for maximum productivity while allowing your reviewer not to miss anything and could then be considered a good metric.

Agile Velocity and Story Points

- Agile velocity measures the amount of work a single team completes during a software development iteration or sprint. It represents the amount of story points

completed over time and can be visualized as the slope in a classic burndown chart (see fig.1). Story point are a unit of measure for expressing an estimate of the overall effort that will be required to full implement a product backlog item or any other piece of work. Story points are a subjective measure and decided and agreed upon by an individual software development team or organization. Agile velocity is a powerful metric when used for individual team sprint capacity planning but when used for anything other than this, bad things can occur. For example, if software engineers are evaluated by a point system per sprint then talented engineers working on hard problems that could take longer are devalued by the equation.

Fig.1 taken from (source3)



Impact

- Impact is a more recent metric that analytic products have incorporated, it takes lines of code and other factors such as files changed and new code versus legacy code. Legacy code is code written by someone or yourself over two weeks ago. These factors are all combined to create what is called an “impact score”. According to Abi Noda this metric is almost unanimously unliked in the industry. It suffers from the same problems that the lines of code metric does , it is also hard to understand the different factors and how they are all combined to create a score, i.e. it’s too abstract to be actionable.

Algorithmic Approaches

Before moving on to more complicated algorithmic approaches I will explain the calculation of two more metrics that can be used as measurable data.

ABC Metric

- The ABC metric is calculated by counting your programs assignments, branches and conditionals, adding each counts’ square power and then finding the square root of

that answer. The following rules give the counts of assignments, branches and conditionals in the programming language C. Taken from source 4:

1. Add one to the assignment count when:
    a. Occurrence of an assignment operator (**=, *=, /=, %=, +=, <<=, >>=, &=, !=, ^=**).
    b. Occurrence of an increment or a decrement operator (**++, --**).
2. Add one to branch count when:
    a. Occurrence of a function call.
    b. Occurrence of any goto statement which has a target at a deeper level of nesting than the level to the goto.
3. Add one to condition count when:
    a. Occurrence of a conditional operator (**<, >, <=, >=, ==, !=**).
    b. Occurrence of the following keywords (‘**else**’, ‘**case**’, ‘**default**’, ‘**?**’).
    c. Occurrence of a unary conditional operator.

- This metric is similar to impact score in the way it is hard to interpret. It is more useful than lines of code as these assignments, branches and conditions have more meaning but it can still be manipulated by engineers looking to inflate their metrics and discourages precise code.

Cyclomatic Complexity

- A quantitative measure of the linearly independent paths in source code that can help you understand the complexity of your program and improve code average.
- Microsoft documentation states: “cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches. A low cyclomatic complexity generally indicates a method that is easy to understand, test and maintain.”
- It is calculated like so:

$$CC = E-N+1$$

- Where CC denotes cyclomatic complexity, E the number of edges in the graph and N the number of nodes in the graph.
- Cyclomatic complexity is a useful metric because it rewards elegant and simple code. However less complexity isn’t necessarily correlated to efficiency or extensibility.

Next, I want to outline a couple of theoretical frameworks that attempt to equate the effectiveness of networks and co-worker behaviours on business productivity. These are taken from a work paper called “Network Effects on Worker Productivity” by Matthew J. Lindquist, Jan Sauermann and Yves Zenou. Ther study isn’t specific to software engineering but is certainly relevant and consists of algorithmic approaches to measure worker

productivity which can be applied to the software engineering office. The following is an example of these algorithmic approaches.

Identifying Key players.

- The key player is the member of the team who should be targeted by the planner so that, once removed, they will generate the highest level of reduction in total activity. Here, the key player will be the worker that the firm would most like to retain because, if removed, total productivity will be reduced the most.

Formally, a *key player* is the agent whose removal from the network leads to the largest reduction in the aggregate effort level in a network. Let $\mathbf{M}(g, \phi_1, \phi_2) = (\mathbf{I} - \phi_1 \mathbf{G} - \phi_2 \mathbf{G}^*)^{-1}$, with its (i, j)-th entry denoted by $m_{ij}(g, \phi_1, \phi_2)$. Let

$$\mathbf{b}(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = \mathbf{M}(g, \phi_1, \phi_2)\boldsymbol{\alpha}$$

with its i-th entry denoted by $b_i(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = \sum_{j=1}^{n} m_{ij}(g, \phi_1, \phi_2)\alpha_j$. Let $B(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = \sum_{i=1}^{n} b_i(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = \mathbf{1}_n' \mathbf{M}(g, \phi_1, \phi_2)\boldsymbol{\alpha}$ denote the aggregate effort level in network g, where $\mathbf{1}_n$ is an $n \times 1$ vector of ones. Let $g^{[-i]}$ denote the network with agent i removed. Let $\mathbf{G}^{[-i]}$ and $\boldsymbol{\alpha}^{[-i]}$ denote the adjacency matrix and vector of covariates corresponding to the remaining agents in network $g^{[-i]}$. Then, the key player i^* in network g is given by $i^* = \arg\max_i d_i(g, \phi_1, \phi_2, \boldsymbol{\alpha})$, where

$$d_i(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = B(g, \phi_1, \phi_2, \boldsymbol{\alpha}) - B(g^{[-i]}, \phi_1, \phi_2, \boldsymbol{\alpha}^{[-i]}). \quad (4)$$

- This method of identify a key player or key players is particularly useful for software engineering teams as often it is theorised that there is a few members of a team who are truly strong innovators who the rest of the team should centre around. The 10x theory, for example, suggest that 10-20% of a team are ten times more efficient than the rest and these people are important to spot.

Computational platforms available to measure software engineering

There are a variety of platforms that are adopted by tech companies to measure their software engineers and the software engineering process. Three that I researched, Pluralsight flow, Waydev and code Code Climate, draw parallels as git analytic platforms. They provide features that intake metrics as before mentioned in this report and display them to provide visibility for managers and employees.

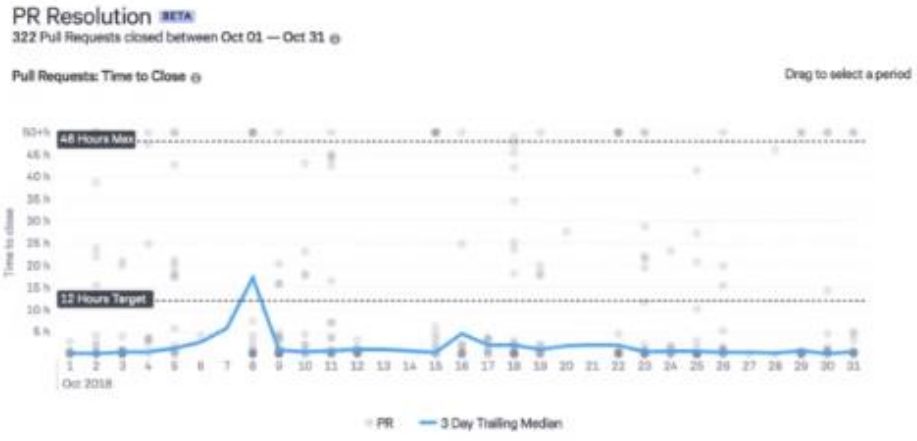
These highly regarded systems are fairly similar to one another so I will go into details on Pluralsight flow and what its features look like as it gives a good gist of what these platforms are about.

1) Coding and Review Metrics

- **Total engineering throughput over previous 90 days**. This uses the impact metric as mentioned before and looks how the impact score rises and falls over the last while.

- **Segmentation of work delivered**. This measures new work versus legacy refactor versus code churn. Code churn is a measure that tells you the rate at which your code evolves.

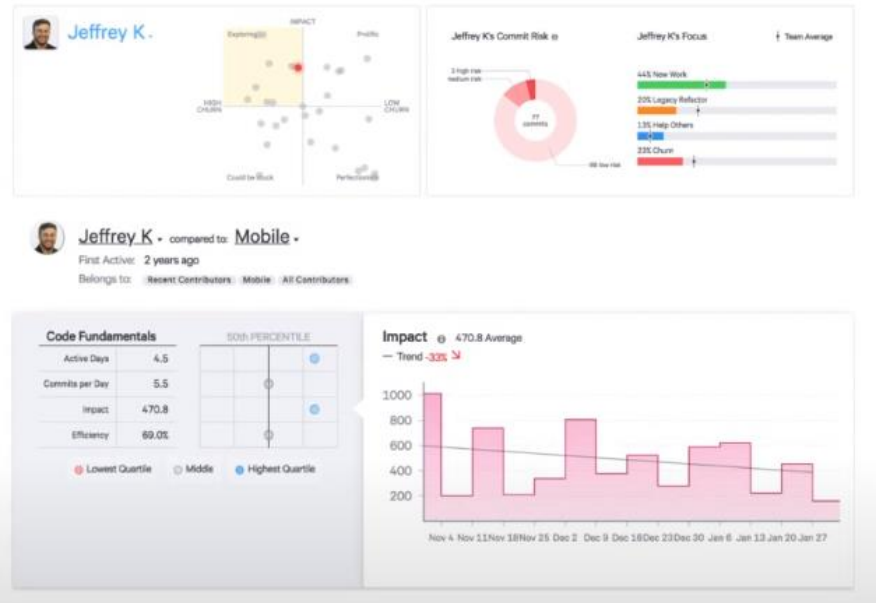
- **Code review resolution time**. This counts pull request and how quickly they are resolved.



2) Operational data for team managers

- **Cross-repository activity dashboard**. Shows size and frequency of commits, merge commits, pull requests submitted, pull request comments and ticket comments.

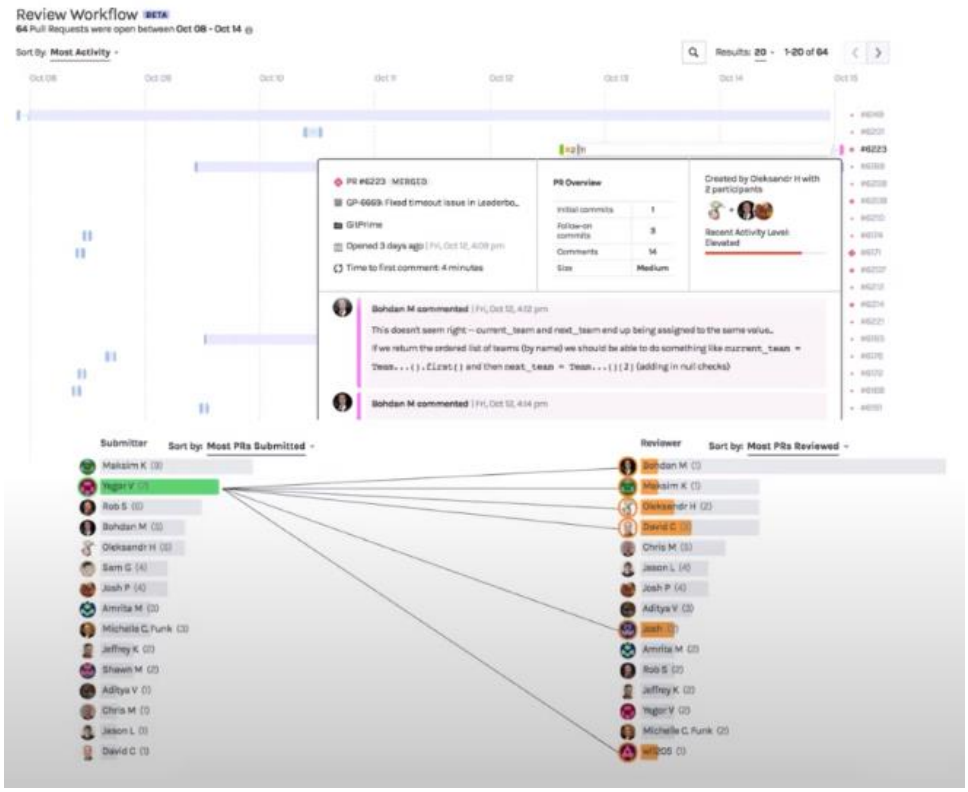
- Insights on each engineers work patterns and comparison graphs to averages, expectations and other ranges.



3) Code review and pull request analysis

- Team and organization-level metrics around pull request duration, responsiveness and collaboration levels.

- Details of pull request activity history, plus visual mapping of team collaboration patterns.



Apologies for photo quality, all screenshots taken from source6.

Code Climate and Waydev have similar features as outlined but with their own flare. How effective these platforms are seems to be ambiguous but Pluralsight, for example, is used by

Google Cloud, Outlook, Amazon and GitHub so it is clearly a helpful tool. I think it is extremely worthwhile to use one of these platforms for your tech company as it gives great visibility on the activity of your software engineering process. However, there is no direct correlation between owning one of these services and progress, it depends on how you use them. Pluralsight offers an adoption plan to help their owners use it the correct way to help their business succeed.

Conclusion and ethical concerns

Conclusively on measurable data, it is quite clear there are numerous metrics that can be used to help evaluate the work of software engineers however these metrics when used in the wrong regard can be extremely harmful for the individual and the business.

I remember, in second year, I was working with my team on our college software engineering project, my teammates informed to consistently commit as we would be partially graded on our activity on the groups repository. In retrospect this is an example of a harsh and somewhat ineffective system as the weight in terms of code for me was the same as my teammates however I was given an easier workload in comparison, I did indeed contribute less to the project but was graded similarly because my commit frequency was similar. Furthermore, this encouragement to over-commit enhanced my already substantial confusion with git.

Metrics are flawed but to make them work it is important to measure process not output, measure against targets not absolute numbers and avoid individual metrics.

Managers should employ systems like git analytics platforms but use them as guidelines and proof of a working formula or to seek indication for change. It is dangerous to reward employees solely off what they consider “good” metrics.

Bibliography

Source1: The elusive quest to measure developer productivity - GitHub Universe 2019

https://www.youtube.com/watch?v=cRJZldsHS3c

Source2: GORUCO 2016 - Keynote: Code Quality Lessons Learned Bryan Helmkamp

https://www.youtube.com/watch?v=vcH0RBe4Eew&t=1s

Source3: Why Agile Velocity Is The Most Dangerous Metric For Software Dev Teams

https://linearb.io/blog/why-agile-velocity-is-the-most-dangerous-metric-for-software-development-teams/#:~:text=Agile%20Velocity%20measures%20the%20amount,in%20a%20classic%20burndown%20chart.

Source4: ABC Software Metric (Wikipedia)

https://en.wikipedia.org/wiki/ABC_Software_Metric

Source5: Network Effects on Worker Productivity

https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

Source6: Debug Your Software Development Process With Pluralsight Flow | Webinar

https://www.youtube.com/watch?v=zks8f-WTIps

Source7: Pluralsight website

https://www.pluralsight.com/product/flow