Measuring Software Engineering

Table of Content

Introduction

Seeing that the software market is a multi-billion industry and still growing, research on how to measure the software engineering process must be conducted to minimize the loss and maximize the efficiency of said process.

Due to the nature of software engineering, the software engineering process has been difficult in measuring, seeing that changes will be made and a flaw in design would set back the whole process. The measurement of effort in completing a task or fixing a bug is also hard to define, even by an experienced software engineer, not to mention that the amount of error each need in completing the same task would differ.

As a result of the effort of different parties, we have come to a situation where we have identified several measurable metrics that would let us have an outline of the progress of software engineering. While not being precise, it does give us a point of reference to seek out points of inefficiencies and substance that would improve the efficiency of the software engineering process.

It is also important to know that some metrics do not provide enough context when being used by itself, such as source line of codes (SLOC). Different approaches of the same functions that provide the same functionalities using the same algorithms with the same efficiency timewise and space-wise could be the differences between 3 lines of code and 100 lines of code. Just because the approach with 100 lines of code has more line of code, it does not mean that it contributed more to the software engineering process, nor can we say otherwise.

Measurable Data

Engineering Data

- **SLOC**
  Source line of code (SLOC) is the measurement of the size of a product by the number of text lines. This is a very crude way of measuring the progress of a software engineering process with the upside of the metric being able to be acquired by automation and provide a rough overview on the ranges of engineering work and maintainability. It is also one of the earliest metrics that was used in measuring the software engineering process, and thus having more data from using such a metric.
  The downside is where it is very much unreliable when developing a product when the numbers would vary from a language to another and the logical method used by different developers, providing faulty estimations.
  That is without mentioning the increasing popularity of the agile development process, where a relatively fresh code will be discarded later, throwing a bolt into the already unreliable SLOC measurement.
- **LOC per commit**
  Lines of code per commit means lines of code per commit.
  LOC per commit matters a lot, especially when working in a large team with lots of bug fixes and feature implementations to keep the integrity of the codebase and save a few hairs when stuff went wrong.
  A larger LOC per commit would mean more time spent, more mental toll to each of the reviewers, and a harder time to figure out if something is wrong, and where something is wrong.
  A larger team would enforce low LOC per commit (with exemptions) to keep each commit relatively lightweight, and better isolation of worries.
- **Pipeline success rate / Unit Testing**
  Engineering teams would most likely set up a pipeline for products in production or those in development.
  Depending on the agreement between the team, one might not even be able to commit code that did not pass the test suites. A commit would pass through a CI/CD pipeline when trying to create a pull request, an automated test will be run and environments will be emulated to ensure that it does work on specified environments.

- **Sprints**
  Sprint(s) is a term used in Agile product development, referring to a period where a specific part of work needs to be completed or up for review.
  We could measure the amount of tasks for a sprint cycle, the number of tasks that got pushed back, the nature of the task, and more. However, this metric is different for teams, the nature of the product, and the experience of scrum master in deciding the works of a sprint.
- **Bugs**
  A bug is a fault in a program introduced by programmers when writing a program.
  Bugs could be used as an indication of the quality of code and the strictness of a test suite. Depends on the product, the introduction of a bug could be considered severe or another item on the backlog.
  Measuring the number of bugs, the nature of said bugs, and the source of the bug would help in the allocation of resources to reduce the future introduction of said bugs, not only in the current product but also be shared cross-team.
- **Incidents**
  Incident(s) refers to an escalation of unplanned interruption or reduction in the quality of a service.
  There are manually and automatically escalated incidents. Automation could be introduced to deal with reoccurring incidents, and on-call engineers to deal with the rest.
  Allocation of resources and restructuring of service could be planned with the analysis of the frequency of incidents, the geolocation of origin, and the time frame of spikes in incidents.

Collaborative Data

- **Exchanges (Email/Direct Message/Call)**
  Exchanges could be tracked in two categories, exchanges within the organization, and exchanges outside the organization.
  Exchanged within the organization could be an indication of how collaborative an employee is, and analyzing the team of both sender and receiver could give an overview of the exchanges of information of both teams.
- **Number of attended meetings**

The number of attended meetings is the measurement of the number of meetings attended by an individual employee.
While it may give an overview of how eager an employee is in attending meetings, it is not very helpful in determining the productivity of said employee as many meetings are simply put, a waste of time for the developers.

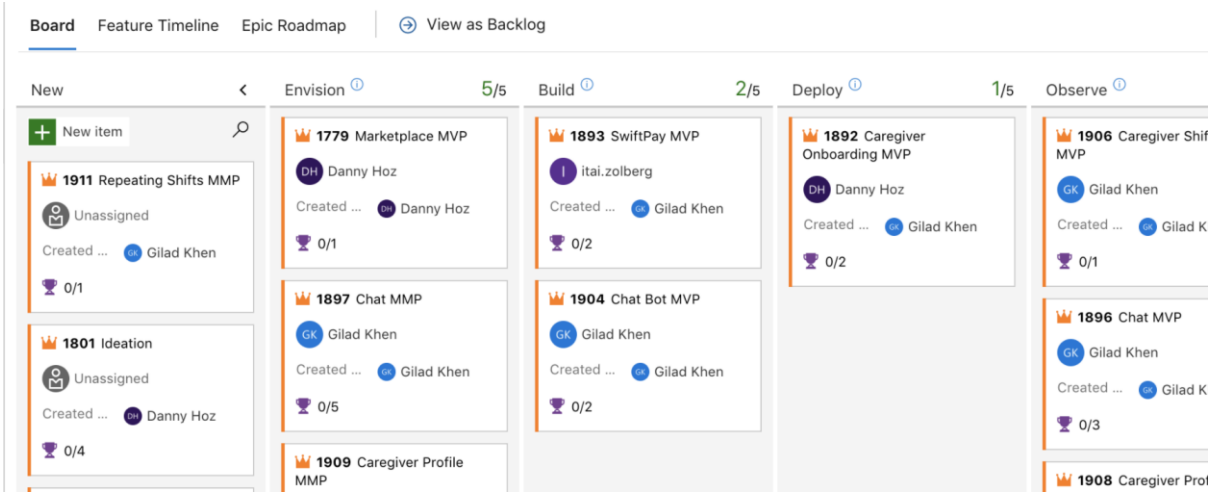
Personal Data

- **Health**
  The health of employees could be monitored.
  It will provide an insight into the productivity of employees corresponding to their health, the number of works employees took before calling in sick and arrange changes to prevent the situation from happening especially during peak time.
- **Working duration**
  Measurement of the work hours of an employee.
  It is quite an ancient measurement of work. However, it might not work as well in the software industry due to the nature of said industry (with exemptions). It might be able to give insight on changes in productivity for an individual employee, or the effect of increasing/decreasing working hours on productivity.
- **Activities on work**
  Tracking activities done during working hours.
  One could track activities employees done during working hours, including but not limited to browsing history, background tasks, activity/inactivity, and personal device usage.
- **Temperature/Brightness**
  Measurement of office temperature and brightness.
  Analyzing the effect of temperature and brightness on the productivity of work would be a quick and cheap way of getting more productivity.

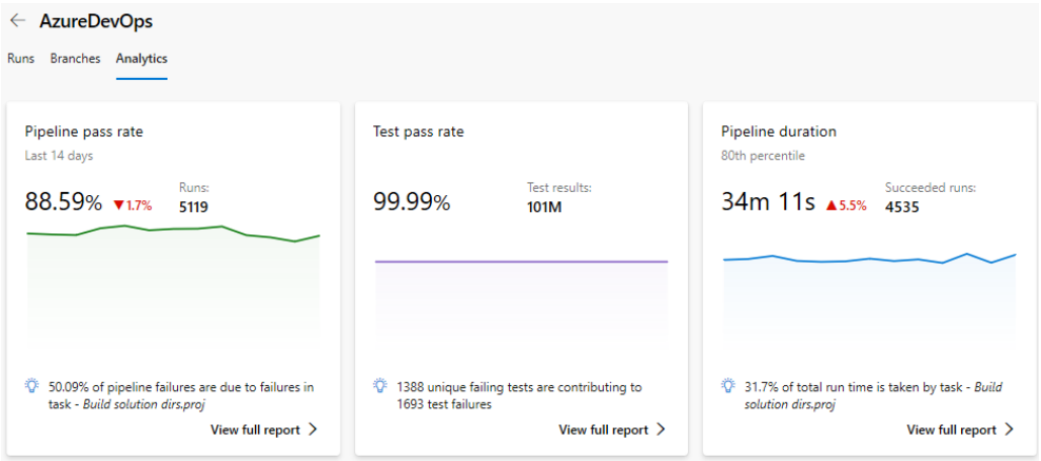
Computational Platforms

- Azure DevOps

Azure DevOps is a decent platform for tracking works, setting up pipelines for CI/CD and automated tests, analysis of success rate etc.



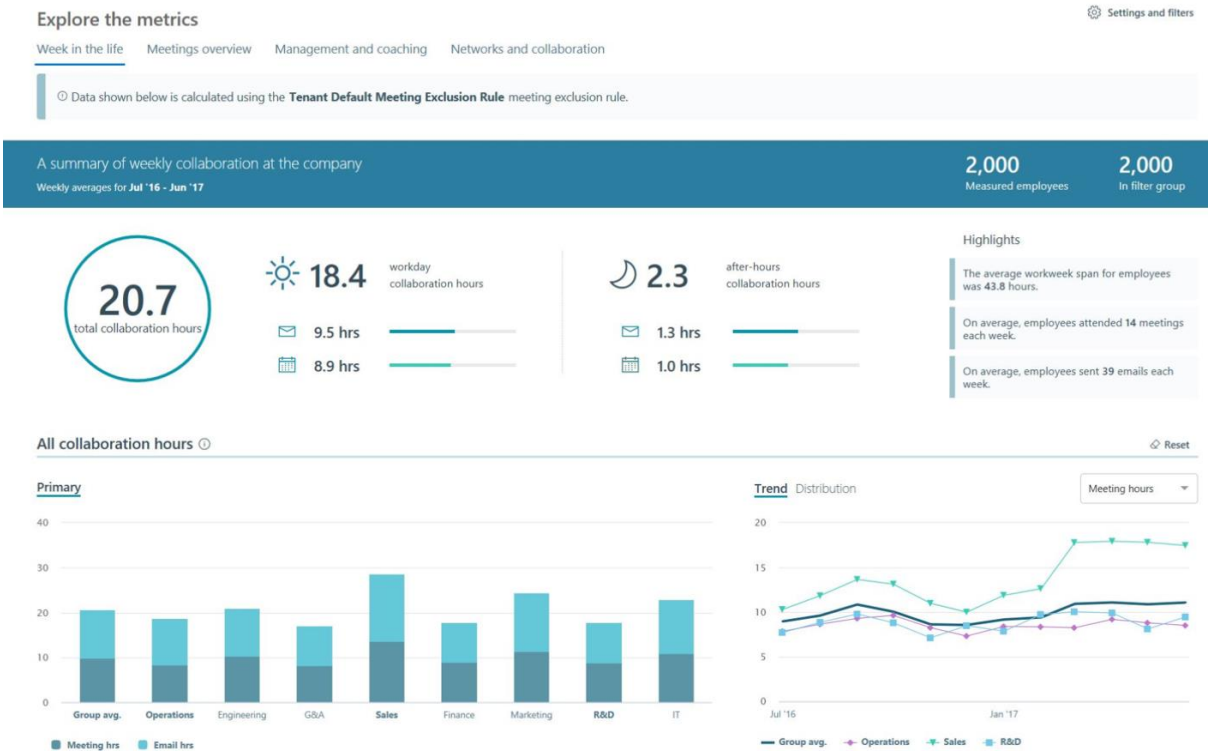
Boards to track task



Pipeline success rate analysis

- # Workplace Analytics

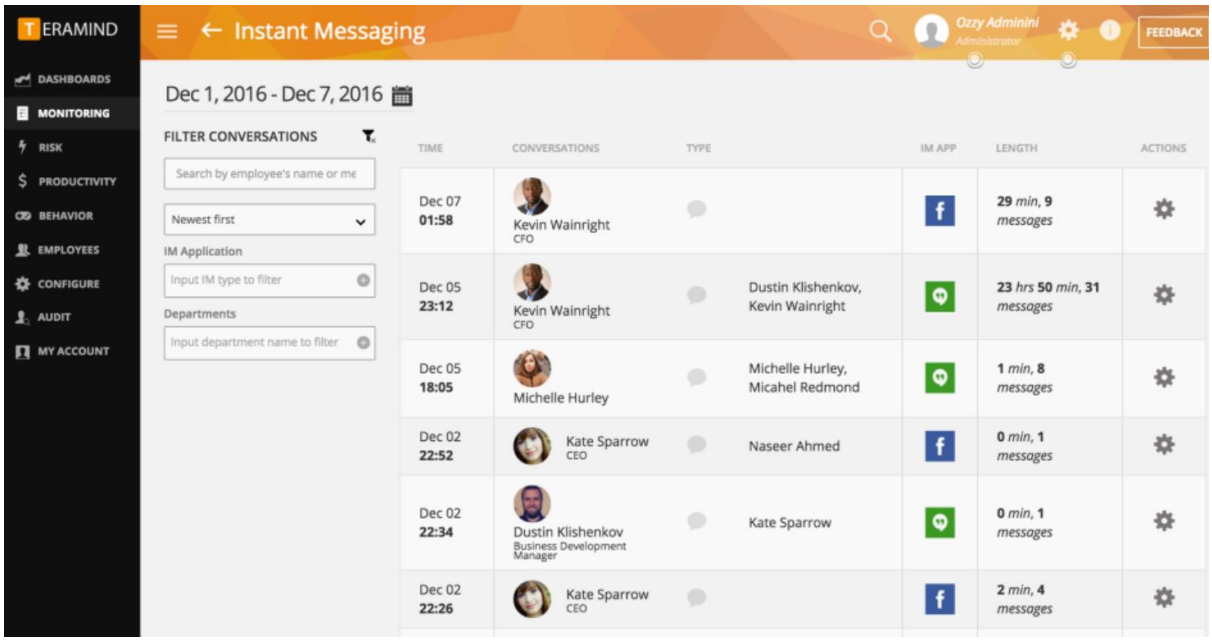
Workplace Analytics is a solution in measuring the collaborative data of each employee.



Collaboration analysis

- # Teramind

A solution to track personal data on a more intrusive and extensive fashion.



Algorithmic Approaches

- ## Static code analysis

  Static code analysis is a method of automatically debugging code to find potential bugs or exploit and ensure some degree of code quality.
  Tools like Infer checks for underlying bugs by running against predetermined rules without requiring input. It is usually run before the software testing phase, which meant it is very helpful in taking out bugs early on in the development process as fixing them right before or after shipping is extraordinarily expensive, timewise, and resources.

- ## Machine learning analysis

  Machine learning could be used to analyse code or to build tools that analyse code (Infer for example).
  Started with rules provided by the engineer, the machine learns to create newer ruleset to better ensure code quality. It helps as the ruleset fed to a tool is rather language-specific and use case-specific. It is also hard to gather ruleset for a new language through open source as verification of the integrity of said ruleset is nigh impossible. Having the machine generate would take off a lot of burden off engineers shoulder and would be transformable to other languages/use cases.

Ethical Concerns

Not including privacy laws, quite a few of the metrics from personal data category collected are already considered unethical (browsing history for one).

Collecting such metrics is a breach of privacy, and in the case of a data breach would endanger tracked employees. Even if told before signing the contract, it would still be extremely unethical to collect such personal and identifiable data. It would also cause unnecessary stress and fear to the employees while at the same time possibly decreasing the overall productivity.

Disclosing collected metrics and possibly ranking on the productivity regardless if they meant anything would also create a toxic work environment and encourage score padding to increase themselves on the arbitrary ranking.

That is not to mention that ethically, changing condition and measuring the effect on employee is akin to experimenting on lab rats.

Some measurement on behavioural data on the engineering category might be discrimination to those people. There are people out there performing better even if it is exactly opposite for the rest of the population. Giving suggestions based on the analytics without considering the employee itself would be a very bad move.

Bibliography

Facebook's Code Checker – Computerphile

https://www.youtube.com/watch?v=tKR2UZdRpV0

Program Correctness - Computerphile

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

Machine Learning Analysis

https://medium.com/pvs-studio/machine-learning-in-static-analysis-of-program-source-code-21bdc7e8ce6d