A Report on Measuring Software Engineering

Jack Manning

18325220

Table of Contents

How Can Software Engineering be Measured and Assessed?

Introduction

In preparation for writing this report I initially consulted with my dad who is a software engineer of 35 years now. “Software Engineering is an art, do you think someone could measure Leonardo DiVinci painting the Mona Lisa?” is the response I got unsurprisingly. I replied, “Well, possibly nowadays you could analyze the time he took to do it, compare the popularity of it with other pieces, the level of different colours, the shadows, or the depth complexity and that could give you an idea?”.

Going on from this initial conversation and researching further there appears to be two, very different, trains of thought. Firstly, the side of many developers who firmly believe that what they do is a creative art and any attempts to measure it is both completely fruitless and an insult to their profession. No doubt stemming from previous mismanagement in companies where vanity metrics were pursued in replace for effective measurement. And then secondly, the ever-evolving side who believe that in fact software engineering can be measured, now more than ever by using the wealth of data which is now readily available to us. Many of the leaders in this second school of thought say that new methods of collection and analysis do in fact measure software engineering accurately and should be implemented into any face paced development organization.

Why Bother Measure in the First Place?

Firstly, why should we even attempt to measure software engineering? In a report by PluralSight, one of the leaders in the “measurement is useful” train of thought write the comparisons between software engineering in organizations versus other departments such as marketing, sales, customer service, or finance. All these different functions within a business benefit greatly from closely tracking and monitoring their related Key Performance Indicators such as the cost to acquire a customer, sales growth or the number of new leads, tickets responded to, and return on investment (PluralSight, 2018). Software development has always been far more ambiguous leading to software engineering managers often basing their performance on their gut feeling or supposedly fruitless metrics such as lines of code. In many ways, software engineering is a lot less tangible than other fields, especially when considering that software is oft the company's primary product. So as an executive when the head of technology asks for $10 million in investment for a new product, they really would like to see some sort of metric they can report to the shareholders which can monitor the team's progress.

Secondly, any respectable software engineering team would also want to monitor their progress and see how far they've come? I'm sure everyone in the team would know roughly in the back of their heads who is pulling their weight, but this isn't scientific or really at all factual. I feel with proper measurement great software engineers could be rewarded for the work they do, rather than the engineer who gets on well most with the boss or is a people pleaser.

Moreover, poor software engineering measurement, mixed with poor requirements ends up costing companies millions of dollars each year. One report which studied over 100 companies and only including software projects costing over $250,000 found that poor business analysis resulted in 3 times as many project failures as successes and companies ended up wasting $1 out of every $3 due to poor requirements and measurement (IAG Consulting, 2009).

What can we measure?

In the past and currently, software engineering has been measured through what could be considered traditional vanity metrics such as source lines of code, commits, hours worked, bugs closed, defect rate, the accuracy of estimation, or function points (Nortal.com, 2018). Many of these can easily be "gamed" and cheated by teams of engineers. They rarely produce intelligent insight and have many negative by-products. In the wise words of Einstein – "Not everything that counts can be counted, and not everything that can be counted, counts".

Looking deeper into some of these metrics we find deep underlying flaws in measuring these, particularly on an individual or historic basis. Trying to measure software engineering based on hours worked will almost always never result in any useful data and according to several pieces of research the longer you work the less productive you become (Pencavel, 2014). This practice favors several poor practices in regards to "gaming" the system as well such as simply working slower or even working huge numbers of hours but not being productive.

Lines of source code is another antiquated measure of software engineering. In software engineering, the solution to a problem is often reducing the solution to its simplest form. It promotes unnecessarily long code when solutions could be done in fewer lines and it takes no consideration for the quality of the code or whether it is full of bugs. This method of measurement would no doubt send shivers down the spine of any software engineer and does not contribute to a healthy working environment within projects.

The number of commits metric would promote better development practice due to the fact small frequent changes usually result in a better outcome and more stable product. However, measuring software engineering based on this alone doesn't show how useful these commits were, whether they are full of bugs, whether it's core functionality, the size of these commits, or the quality of them. It might produce good practice, but it also can promote poor practice and rarely gives insight into how good or bad the software development is.

Bugs closed and bug rate are both closely linked. Measuring engineering-based off bugs closed isn't pointless but could promote poor code, to begin with. Bug rate could be helpful but it is very difficult to compare one bug to another and also relies on humans to track it. One bug might take an hour to fix, a marginal error whereas another is a huge undertaking requiring days or weeks. However, quick bug fixes for customers should be measured and helps provide overall business value.

What should we measure?

Nowadays there are a plethora of different metrics that are commonly measured. Some "quality metrics" by GitClear, a leading company in the software metrics space, are OKR's, Line Impact, Story Points, Escaped Defects, Pull Request Completion Rate, Lead Time/ Cycle Time, Test Coverage, Team Velocity, Code Churn, Release/Sprint Burndown, and Mean Time To Repair (GitClear, 2020). Another leading company: PluralSight considers that the 5 developer metrics which are essential to track are: Lead Time, Churn, Impact, Active Days, and Efficiency.

In today's world with an ever-expanding data footprint left behind developers, there is really no end to the number of metrics an organization could track. More importantly, it is probably most important to make the metrics work for the organization you're looking to monitor. In a talk at Github Universe 2018, Abi Noda, a senior product manager at Github gave his recommendations when using metrics. Firstly, measure process – not output. A high performing team is a responsive team, that collaborates well with one another and has good process within. Measuring these processes such as code review times, time to open, and code review turnaround time are important but not religion. Using absolute numbers is dangerous. Aim to use trends and remember that no two teams are the same. Secondly, aim to measure against your preset targets which are team unique. Lastly, never measure individual performance or individual performance based on historic metrics as this is very dangerous and could make a developer's lives hell (Noda, 2019).

What Platforms Can be Used to Gather and Process Data?

Introduction

Quick, efficient, and productive software engineering saves time and money. Anything which saves time and money creates an industry. Today many different platforms can be used to gather and process data about software engineering teams.

Possibly one of the most well-known pieces of work regarding measuring the software engineering process is the Personal Software Process or PSP. Developed by Watts Humphrey, “the father of software quality”, in 1997. However, I would question if anyone ever actually uses this anymore due to the sheer level of manual entry and lack of automated tracking. It all seems quite academic for the normal software engineer working at a company with over 12 different forms to fill out at the beginning. Upon researching this in further depth I learned that using PSP is like programming with a logbook and a stopwatch. No doubt that many elements in PSP are important to software engineering today but in this report, I’m going to look at the next wave of platforms that make use of automatically analyzing the data footprint software engineers leave behind.

PluralSight

PluralSight Flow, previously GitPrime, is a new way to measure and manage software engineering. It makes use of historic git data to provide aggregated, easy-to-understand insights and reports for software engineering teams. According to their website, it allows software teams to “Work better together with deep insights into your engineering workflow” (PluralSight, 2020). What I think is best about this is it requires very little manual input from the engineer themselves but rather helps to analyze what they’ve worked on and provides an easy place to collaborate as a team.

Some of their core measurement metrics are: “Productivity and Output”, “Commit and Pull Request Behavior”, “Code Churn”, “Legacy Refactor versus New Work” and “Collaboration Trends in Code Reviews”.

PluralSight Flow Impact Metric

Productivity in any software organization is important and it will also be wanting to be improved upon. Pluralsight uses their “Impact” metric to track this. They say that this metric answers the questions of “Roughly how much cognitive load was carried when implementing these changes?”. PluralSight baseline past team output levels and measure progress towards systematic improvement in a clean way *(PluralSight, 2018)*.

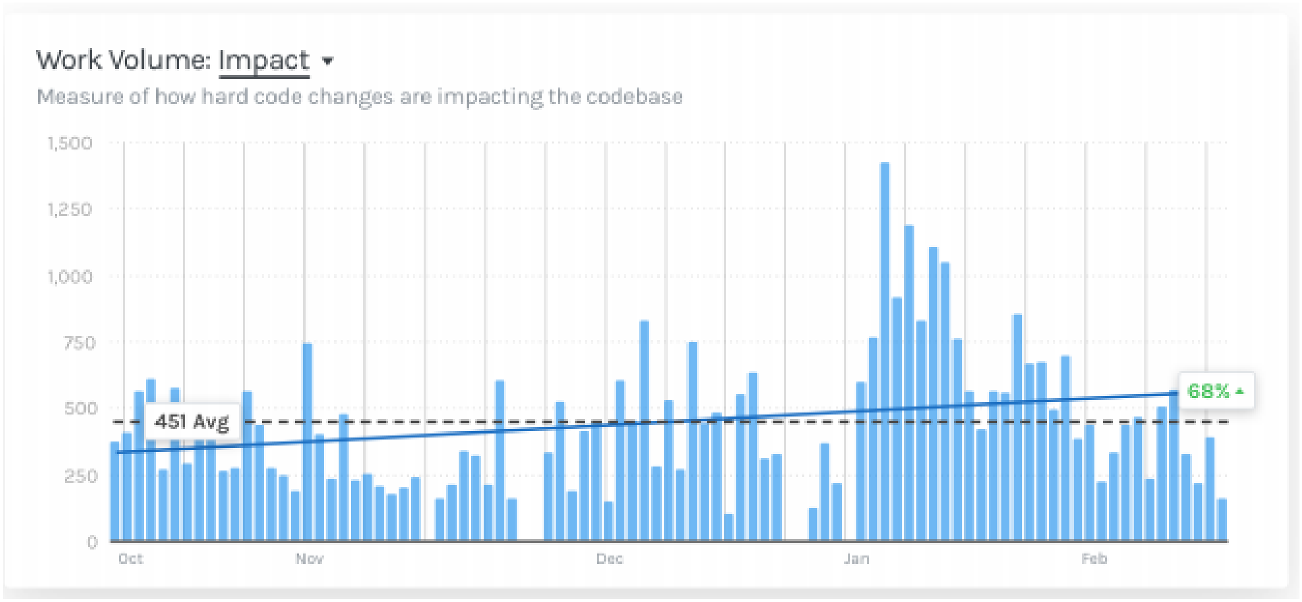
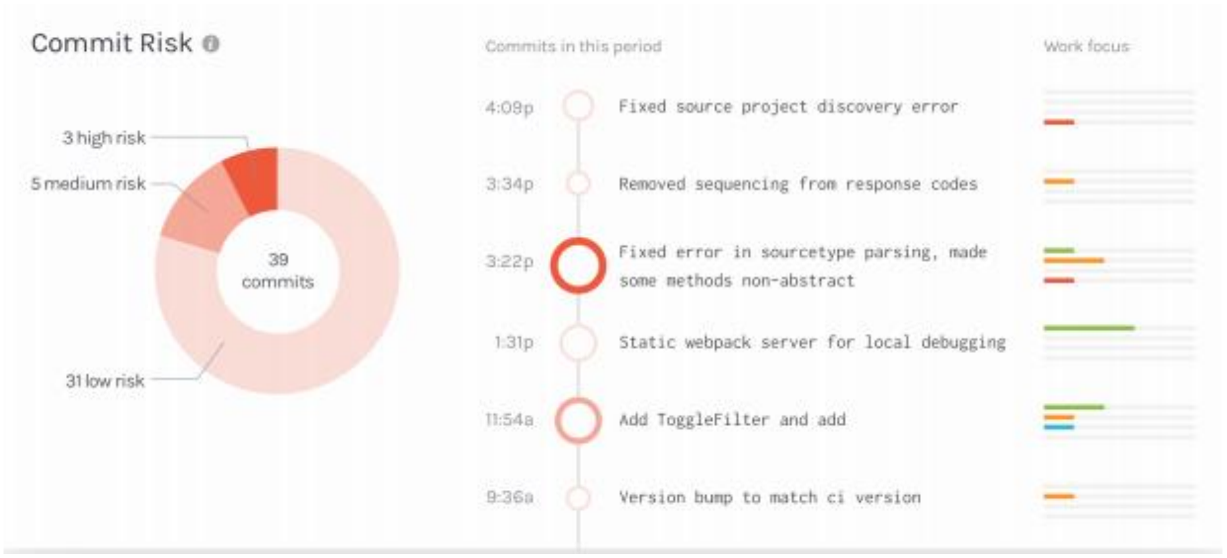


Figure 1 PluralSight Flow Impact Metric

PluralSight Flow Commit and Pull Request Behavior Metric

PluralSight Flow tries to reduce the need for 1-1 meetings and daily standups by focusing on creating a data-driven dashboard that shows commit progress and pull request reviews as they happen. Reducing the need for this verbal interaction by diagnosing risk early on and getting status on progress means engineers can^ spend more time programming.

The platform can gauge the content of the commit and pull requests as they occur and understand their code profile and complexity. It is a way for managers to easily understand the progress of their team members.



4:09p Fixed source project discovery error

3:34p Removed sequencing from response codes

3:22p Fixed error in sourcetype parsing, made some methods non-abstract

1:31p Static webpack server for local debugging

11:54a Add ToggleFilter and add

9:36a Version bump to match ci version

Figure 2 PluralSight Flow Commit and PR dashboard

PluralSight Flows Code Churn Dashboard

Code Churn is a natural part of the software development process. However, code churn levels that deviate significantly higher or lower than expected norms can represent “smoke that is an indicator of a potential fire”. As a part of PluralSights proof of concept, they analyzed 85,000 software engineers with over 7 million commits across 1.8 million active days and found empirical evidence that indicates that code churn levels most frequently run between 13-30% of all code committed, therefore 70-87% efficiency. So a typical team should aim to operate around the 70% to 87% efficiency mark.

However, Plural Sight is more interested in the deviations from the baseline levels here. By baselining the “natural” churn levels of typical projects, software engineering teams can actively monitor churn by an engineer or by a project to identify bottlenecks. This can indicate much earlier on whether a project is not progressing as planned. In addition to this, it can identify if engineers are possibly sacrificing speed for precision or if an engineer is stuck on a piece of the project.

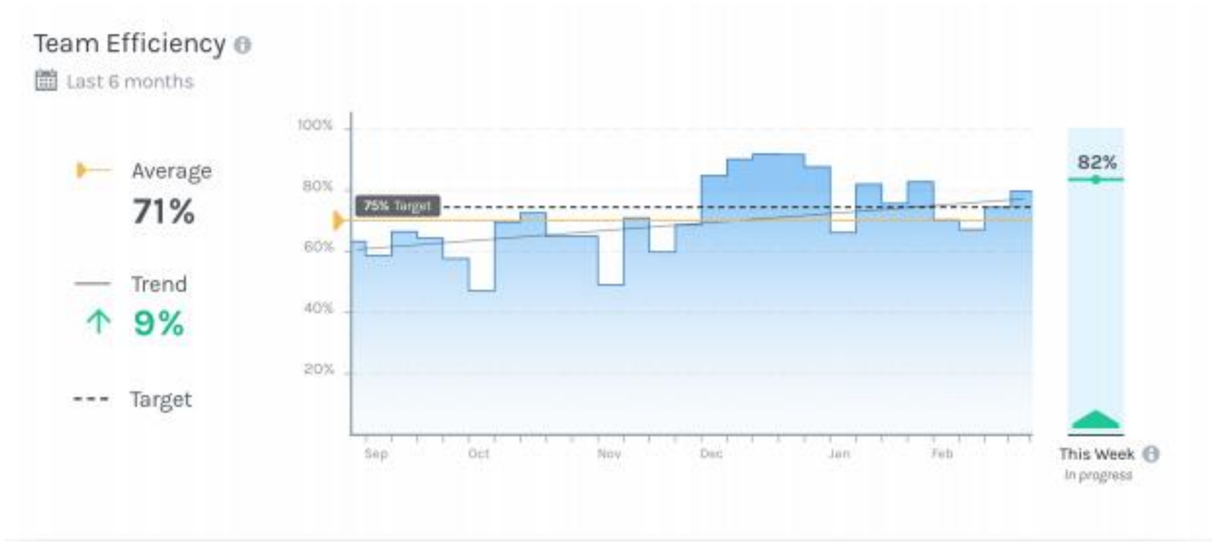


Figure 3 PluralSight Flows Efficiency Metrics

PluralSight Flows Collaboration trends in code reviews Dashboard

PluralSight offer a really easy way to understand code reviews by rating pull requests by their underlying code complexity and quantifying the number of reviewers and comments. It easily allows software engineers to understand the collaboration process which is so often overlooked.

By using metrics like this is can reduce the amount of time in code review for things which are not complex and allow for more time in reviews which feature high complexity pull requests. This method of reviewing will help streamline the code review process and also decrease the risk to the existing codebase.

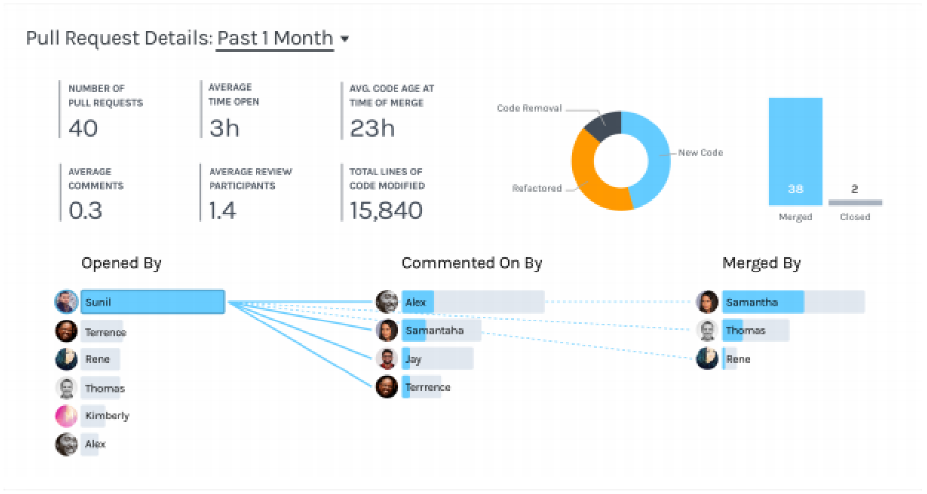


Figure 4 PluralSight Flows Collaboration Trends in Code Reviews

PluralSight Flows Legacy Refactor Versus New Work Dashboards

PluralSight Flow aims to answers the questions: “How much of the software engineering investment is spent on new work, versus supporting or refactoring legacy code?” This metric is calculated at the time of a commit and lets software engineering teams create hard data to see how much of a team's productivity is dedicated to new projects versus technical debt.

Using this metric, teams can see how much money they’re spending on old code rather than working on new exploratory work. Once again all the data is sourced from the source code repositories.

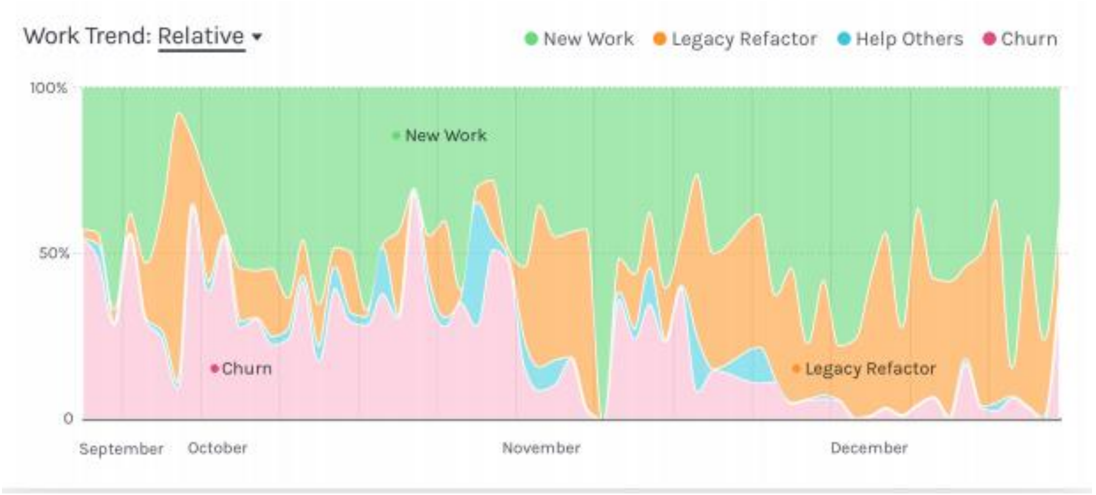


Figure 5 PluralSight Flows Legacy Refactor Versus New Work Dashboard

Others in the field

There are many different companies offering platforms like PluralSight Flow, such as LinearB, Jira, Code Climate, Stackify, GitClear, and Jira. I feel this method of platforms that connect on top of existing online repositories such as Github and BitBucket is where this space is going. I would imagine we are going to see one or two platforms emerge as the go-to connection platform for all software engineering measurements and analytics information. A platform that provides teams with in-depth analysis based on big data from previous repositories which links with all the other tools developers use such as project planning tools like Jira or Trello and communication tools such as Slack.

Eventually, linking to payment processing, text editors, testing tools, and marketing tools so software engineers know exactly what parts of their work give the most business value.

What Algorithms are Available to Use?

Introduction

Now more than ever we are leaving a huge data trail behind us, both as people and as software engineers. The issue now becomes quickly how can we analyze this data and glean meaningful insight from it to improve software engineering products and processes. Using machine learning techniques, I think we could understand this data footprint even better.

Machine learning algorithms generally fall under 3 main categories: unsupervised learning, supervised learning, and reinforcement learning. All of these three categories could be used to measure software engineering.

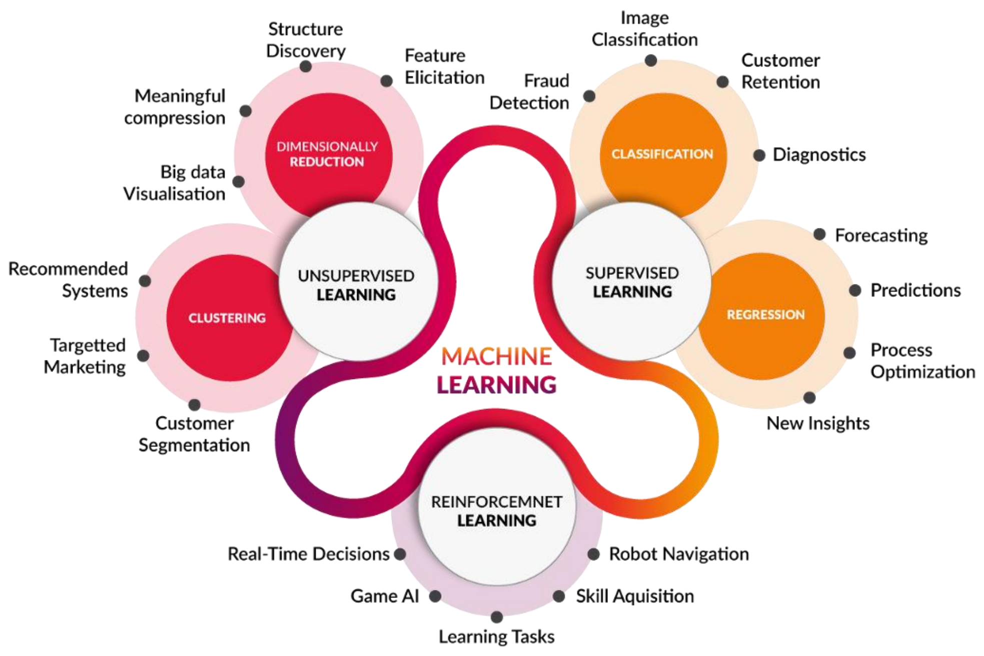


Figure 6 TowardsDataScience.com

Semmle's LGTM ('looks good to me') dataset and related papers

In researching this topic of using machine learning techniques to measure software engineering I stumbled upon lgtm.com. LGTM is an open-source dataset that contains over 39 million commits by more than 700,000 developers analyzed for 135,821 open source projects. LGTM has its own “deep semantic code search engine” and has its own query language called CodeQL (LGTM, 2020).

Building on top of this excellent dataset, a paper titled “Measuring Software Development Productivity: a machine learning approach” (Jean Helie, 2018) aims to attempt to understand software engineering measurements such as quality, coding time, and quantity by using this dataset and machine learning algorithms.

In the study, they found that Code Quality is “an ensemble of nonlinear quantile regression models”. Using the alerts which are built into the LGTM platform, this detected the code quality via run-time bugs and technical debt. In their ML model, they could then accurately give a score from A to E of the code quality of a new project.

Other interesting findings from this study showed that the coding time of a developer is a neural hidden Markov model. From analyzing such a large dataset, they trained a neural hidden Markov model to predict the probability of an individual developer is currently coding for each 1 minute of their recent history (approximately 2 years). They could then use this trained model to estimate the expected actual coding time between two successive commits.

Using Machine Learning to Cost Projects More Efficiently

There is no doubt that eventually machine learning will be used to accurately determine the impact of individuals within a software engineering team. But in researching for this report I came across a few other uses where machine learning will probably be implemented soon.

One of the main reasons attempting to measure software engineering is a focus for many organizations is since so many projects either run overtime or end up costing far more than initially thought. Imagine if an algorithm could tell exactly how much this software would cost to develop? Well, it might be much closer than what we think. By using “genetic programming, neural networks, and genetic algorithms, alongside a distributed programming project data set” (Md aaquib nawaz, 2020). In this study, they used a COCOMO 81 data set to test their models. The result was a model whereby software engineering projects could find better software cost estimates.

Using Machine Learning to make the best development team

Another interesting use of machine learning in measuring software engineering is the fact that by modeling the skills of the different members of the organization it would be possible to make teams with higher productivity (Franciele Beal et al, 2017).

In this particular study, they aimed to classify team participants in different profiles so that during the team's formation and task distribution they could work more productively. The paper presents a developer modeling approach based on software quality metrics using a machine learning-based method. They found that it was possible to use quality metrics to model developers into different profiles.

I could see how some version of this could also be used to see which developer is performing the best and which is performing the least.

Is this Ethical?

Upon researching the topic of measuring software engineering and employees in general, I have concluded that although there are ethical concerns with any sort of data collection, this field doesn't particularly stick out to me as any different from the existing data collection and data analysis we have in other fields such as sales, marketing, or even sports.

For example, take a marketing team within an organization. They will be assessed based on how good their campaigns are, the click-through rate, the landing page clicks, the cost per conversion. I feel like the issue with measuring software engineering lies in the fact that historically there haven't been any great metrics specifically for software engineering. However, it is likely that with all the data which is available and different analysis techniques, better and better metrics will begin to emerge.

Another example is sportspeople, they're tirelessly measured on the number of goals they scored, the distance ran, their strength, and their timings. If we can accurately find suitable metrics for software engineering which reflect their performance, then I think the ethics of measuring software engineering is the question of should we be measuring employees at all?

It depends on the quality of the metric and the context

Obviously in the past software engineering has been notoriously difficult to measure and the measurement of vanity metrics such as lines of code had never gleamed proper results. Until we reach a point where there is a solid set of metrics to judge software engineers on, or maybe we will never reach that point, any sort of metric should only be used for insight rather than punishment.

I think it would be unfair to judge software engineers on a metric in which we are even slightly uncertain of its accuracy in predicting high performance and high productivity. Instead, I would argue that measuring software engineering should be taking an engineer first approach and be used within teams to help determine bottlenecks, save time, and help each other improve.   Firing someone based on a software-related metric sounds a bit ridiculous but then again anyone working in finance would be routinely fired due to low performance. But then again maybe software engineering is a different beast altogether and it can never be compared like for like with finance.

Possibly the team data should be anonymized

Judging based on a team-by-team basis sounds a lot better than singling out individual employees in my opinion. There is no doubt that software is quite a collaborative effort by all parties and that promoting a purely performance-based work environment probably isn't conducive to the overall end goal of great software engineering.

I feel that if teams were monitored against other teams completing similar tasks out there in the world then it would be far more beneficial to all and reduce employee stress or burnout. Two situations when the best engineering doesn't happen at all.

GDPR compliance – and measurement should be done with the knowledge of the employee

One primary concern of tracking or measuring anyone is that the person being tracked or monitored should be opted in and aware it is happening. I also believe that employees in companies should have the option to opt-out from any of this individual tracking. Possibly platforms where individual developers can see their own stats but managers can only see team-wide analytics would be best.

Under GDPR compliance, the employee would have to voluntarily agree to be tracked and have read what data is being collected about them. Other data such as how long it is stored for and if it is ever allowed to be sold to others are major concerns.

References

Franciele Beal, P. R. (2017). Developer Modelling using Software Quality Metrics and Machine.

GitClear. (2020, April 17). *Popular software engineering metrics, and how they're gamed*. Retrieved from https://www.gitclear.com/popular_software_engineering_metrics_and_how_they_are_gamed#mttr

IAG Consulting. (2009). *Business Analysis Benchmark*. Retrieved from https://www.iag.biz/resource/business-analysis-benchmark-full-report/

Jean Helie, I. W. (2018). Measuring software development productivity: a machine.

LGTM. (2020). *Homepage*. Retrieved from https://lgtm.com/

Md aaquib nawaz, M. k. (2020). A Methodology for Software Cost Estimation Using Machine. *International conference on Recent Trends in Artificial Intelligence, IOT, Smart Cities*.

Noda, A. (2019, December 14). *The elusive quest to measure developer productivity - GitHub Universe 2019*. Retrieved from https://www.youtube.com/watch?v=cRJZldsHS3c

Nortal.com. (2018, November 20). *The myth of developer productivity*. Retrieved from https://nortal.com/blog/the-myth-of-developer-productivity/

Pencavel, J. (2014, April). *The Productivity of Working Hours*. Retrieved from http://ftp.iza.org/dp8129.pdf

PluralSight. (2018). *Software Engineering: Why has it eluded data-driven management*. Retrieved from https://www.pluralsight.com/content/dam/pluralsight2/landing-pages/offers/flow/pdf/FLOW_2018-ebook_ar_vf.pdf

PluralSight. (2020). *PluralSight Flow*. Retrieved from https://www.pluralsight.com/product/flow