Measuring Software Engineering

Jeremie Sajeev Daniel - 18323961

13/11/2020

1 Introduction

As a company, whenever a person is being hired, you should seek an adequate amount of work done for the pay that you give them. In many jobs, work done itself would be the catalyst to determine the pay that an employee receives, these are usually known as commission based jobs. They may have a baseline salary, however they are very much incentivised to keep on doing whatever the incentive is e.g. A travel agent might be incentivised to book for people and could receive 10% of the cost of the booking itself. However, what about jobs that don't offer commissions? In a lot of careers it's could be very difficult to know what person is providing more value to a complany. For the employee, you may want to know if you're undervalued in pay so that you have more leverage while negotiating for a salary rise. In journalism, you might be rewarded based on the amount of clicks your article generates for the site. However, that may lead to more sensationalist articles as they realise that it is the best way to have more people engaged and thus have a higher amount of income. This can be a bad look for a business in terms of reputation despite having employees that generate a lot of advertisement income. These are the types of challenges that employers face when analysing the performance of employees across a myriad of careers.

2 How can software engineering be measured?

Measurement of performace, exists in some way shape or form in almost all careers. It is the process of collecting, analyzing and reporting information regarding the performance of an individual in a group. It may be used to give bonuses to people to encourage work to be done.It is primarily used as a means of gauging the performance of an individual, to see whether they are doing what is necessary as the company or their contract demands. It can also be used to help developers develop their own skills based on showing engineers their own weaknesses. However, the ability to actually measure performance is often debated about. In the realm of software engineering, the earliest prototypes of measurement, naively used lines of code as the primary indicator to measure the performance of an individual. On the surface, this idea doesn't seem so

bad. The more lines of code performed the longer an engineer must be working right? There is a certain truth to that statement, an engineer does in fact have to write more code in order to achieve the performance that they want to achieve. However, what's stopping an engineer from elongating the code that they wrote. Suppose that an engineer wrote fifty odd lines of code to create a function for randomly generating a identification number for an individual. However, another engineer wrote the same function with the same specifications required in ten lines. Why is the engineer who wrote less getting punished, despite performing the same in terms of functionality of the program?

Perhaps, the answer to measuring software engineering lies in the number of functional programs an engineer has created? That definition also needs to be somewhat changed. For example, let us imagine an engineer who made the same function in a more optimal form. Does he get rewarded in that system for spending more time to find a more optimal function? Perhaps it's a mixture of all of these combined. There may also be engineers who test their code more rigourously than others. However, this would mean that they would have spent less time on creating new code. But rather, the engineer would be able to optimise their code further and also hunt for bugs at the same time. Another way of possible measurement could be the number of keystrokes an engineer uses. But, I believe that is a terrible way of measurement as it can be easily gamed in order to suit the engineers' needs or someone may just keep typing a character as a sort of stimulus to help them focus. There are also some problems taht are much harder to solve compared to others and thus while the solution may be smaller, the time taken to get a solution might be a lot more difficult than a first glance at the issue. There is also another method that depends on how many bugs and errors the developer can prevent. There are others which depend on how much impact your code has on others and its difficulty.

In my opinion the best way to measure how an engineer has performed is by doing a combination of everything and then peer reviewing code. The peer-reviewing code can be done in order to see whether the code complys with the coding standard that was agreed upon. So that it is easy to read and understandable, so that when in the future a bug may be found within the code, engineers in the future will be able to address the issue without the additional need to understand the previous author's intentions. This can also be helped by providing good comments, that shows what intention is placed with sections of code as well as any improvements or problems with the code that should be fixed later. Problems may also be found in code using code coverage tests which uses a dataset of usecases for a particular function. This can help reed out redundant code as well as fix bugs as it comes along.

Those were some of the methods that would best befit testing the performance of an engineer, however in the workspace, the majority of engineers would not be working alone. Instead they would be in a group or an organisation. As a

result, you may also want to calculate the amount of collaberative work they do. Whether that means going to meetings, talking with coworkers on the project or even discussing with customers on what their concerns are with a product like what they would produce. All of these factors must also be considered when quantifying the perforcance of software engineers.

So in order to summarize how we can measure software engineering, I believe that it can be done in a magnitude of ways, some of which are far more effective than others. This form of data gathering and analysis will allow a company to show employees what they need to improve on as well as showing the rest of a group, areas in which some may struggle in. It may also help people stay on track and perform as they see coworkers performing.

Since the measurement of software engineering is usually present as an attempt to increase or stabilize the productivity of the work, it may also be a good idea to monitor the happiness of a worker. In many countries, happiness has been considered to be an alternative national indicator to GDP or GDP PPP. According to a harvard study, it has been found that people who are happy have 37% work productivity and 300% higher creativity. This may indicate that allowing workers to have a healthy balance between work and breaks which would allow them to enjoy the work that they perform and the people that they perform with. This as a result may lead to an increase in work productivity as people are in a better state of mind. Companies like google, have used this work-life balance philosophy to provide facilities for their employees to relax and take a break when necessary.

3 What platforms can be used to gather and process data?

3.1 PSP

Personal Software Process (PSP) is a structured measurement platform that was described in Watts Humphrey's A Discipline for Software Engineering. It is designed to help software engineers to better understand and improve their performance by bringing a form of discipline in the way they develop code and tracking their predicted and actual development of the code. PSP's primary goals are "to improve project estimation and quality assurance". In PSP developers must fill out forms describing a: project plan summary, time reading log, defect recording log, process improvement proposal, size estimation template, design checklist and a code checklist. The forms contain over 500 unique values that the developer must manually calculate. Developers can modify it by choosing the analytics that best suit their personal requirements. When PSP is taught to students it is quickly abandoned due to the "extra work" involved in using it. So it fell short since it was not convinent for the majority of workplaces. It was supposed to help software engineers to improve their estimating and planning skills, make commitments they can keep, manage the quality of their projects and reduce the number of defects in theork. It was created in order to apply the underlying principles of the Software Engineering Institute's Capability Maturity Model to the software development practices of a single developer.

3.2 Hackystat

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. Users typically arrach software sensors to their development tools which collect and send data about development to a web service called the Hackystat SensorBase for storage. The SensorBase repository can be queried by other web services to form higher level abstractions of this raw data, and/or integrate it with other internet-based communication or coordination mechanisms, and/or generate visualizations of the raw data, abstractions, or annotations. In short Hackystat is tool meant to gather what they call raw data which can later be used for analysis. Unfortunately, it is no longer in development but the code is up on github if anyone wants to see.

3.3 GitPrime

Gitprime (Pluralsight Flow) compared to PSP is not a framework based idea, but rather an actual platform owned currently by Pluralsight. GitPrime is an analytics dashboard tool for code projects and as a result is used to process data. It can watch your team's code repositories on sites or services like Github or Bitbucket, tracking various metrics such as user-by-user code commits over

time, ticket activity and pull requests. The idea is to repersent the date in a visual. This can massively help by highlighting bottlenecks as well as efficiencies. This service in conjunction with frameworks like Scrum will allow for easy and fast insight on how each member of the development team is performing and what tasks need to be done next. GitPrime itself offers four main metrics that they say are the signs of productivity that was discussed earlier, active days - the number of times a person has been active on the project, commits per day, impact - the value of your code and how its referenced and efficency. It also gives a project timeline which shows the change in each metric over the course of the project as a whole. The best place to find inefficiencies, bottlenecks, and stuck engineers would be to look into GitPrime's work log. This is the application's homepage. It displays the amount of work a contributor produces in addition to commits, merges and pull requests.It can also be integrated with Jira to get a more broad overview.

3.4 Velocity

In comparison to GitPrime, Velocity (Code Climate) has a focus on pull requests. It has three main features that allow for progress tracking. Overview, which is the hoime dashboard which also summarizes the progress your team has made so far, based on metrics like impact, pull requests merged or push volume. The overview lets you see whether productivity is trending up or down over a given period. Another feature snapshot, allows you to have a real-time look into the current iteration that your team is working on. You can see workload distribution through the work in progress or contributor count, total pushes for the current iteration and pull requests ordered by activity level. You are also able to set targets which means that the team can set clear goals for improvement as the team can look at shared, metric based objectives. It also features integration like GitPrime.

3.5 Jira

Jira is an agile project management software tool developed by Altassian. Jira allows for integration with external systems such as github and bitbucket, provides an overview of team activity, complete scrum project and customise important workflows. It also provides bug tracking. Like a trelloi, a software by the same company that was used in the software engineering project last year, jira provides methods for users to plan and track changes throughout their product. Alongside the integration with applications like Gitprime analytics can also be used to see team contribution by members as well as help others by giving notices on what needs to be done.You can also break down the project into components which can then be further investigated through issues. The tool allows you to view the workflow of any issue and see how many times it has been investigated and what was done to see if it needs any more investigation. If you're working based on a Scrum framework there is a plugin for it, that allows for easy integration offering reports as well as a backlog, that allows the team

to plan sprints and estimate what needs to go into what sprint. The application allows for easy data gathering as well as team management.

4 What algorithms can be used?

The most basic way to generate an algorithm would be to grab the performance metrics of everyone and derive ther mean and see wo is doing well on the curve and whoever is below the curve. According to "Network effects of Worker Productivity", this would actually help individuals better assess what they need to do and they would be encouraged to further develop. Other tools would use more metrics such as the ones listed below which is used by GitPrime. Some others would use deep learning to establish patterns and predict possible outcomes and ways of improvement. However depending on the data that it receives it may be wildly inaccurate and may make mistakes since a compute cannot make a judgement call, or a call based on instinct. I assume that some of the algorithms that companies like google use to analyse the general public can be used for the workplace as well.

4.1 GitPrime & velocity

GitPrime has its own set of analytics and algortihms that it provides. The list below are the metrics they provide for individual developers. Velociity also shares similar metrics.

4.1.1 Coding Days

Any day where a developer contributed code to the project.

4.1.2 Churn

Code which is deleted or rewritten shortly after being written (less than 3 weeks old).

4.1.3 Commits per Coding Day

The number of commits that a developer made on days when they were able to commit.

4.1.4 Efficiency

The percentage of all contributed code which is productive work.

4.1.5 Help Others

Code where a developer modifies someone else's recent work (less than 3 weeks old)

4.1.6 Impact

Severity of edits to the codebase, as compared to repository history.

4.1.7 Legacy Refactor

Code that updates or edits old code (older than 3 weeks).

4.1.8 New Work

Brand new code that does not replace other code.

4.1.9 Productive Throughput

Any code that is not churn is productive work

4.1.10 Raw Throughput (Raw LoC)

All code contributed, regardless of work type.

4.1.11 Risk

A measure of the cognitive load associated with rectifying a bug found in a commit. The calculation includes how large the commit is, the amount of files touched, and how concentrated the edits are, etc

4.1.12 tt100 Productive

The amount of time consumed to contribute 100 lines of productive code, after churn.

4.1.13 tt100 Raw

The amount of time consumed to contribute 100 lines of raw code, before churn.

4.2 Network effects on Worker Productivity

A study that was published in 2015 called "Network Effects on Worker Productivity" used several algorithms to examine the productivity of employees based on pressure through working in a network. The third section of the article called "Theoretical Framework" contains subsections that curtail some formulas that are used throughout the implementation of the article. These include some formulae such as the key player, that is defined to be the person that once removed results in the highest loss in total activity. As a result, it can be seen as the critical worker in a company. Going through the publication, you are able to see different algorithms that are used top create a worker network, wherein you'll be able to see the impact of different individuals on their coworkers. From there you can see the influence that some of these algorithms would have on a company.

5 Is it Ethical?

This is a very difficult question to answer in my opinion. It feels like I can go with both sides in this argument. This problem is extremely similar to the privacy debate going on. You have many companies that want to capture data and sell it to advertisers. In exchange we mostly get free products. However as a result, we becoime the product in a way as companies will be able to process and sell data about anyone who uses their services. Sometimes you have companies like Facebook, that are able to analyse who you are even if you're not using any of their products. Now back to the topic in hand, regarding the use of this type of data in the workplace. Now my general sentiment around this issue is that since, the company will pay you the employee to do work and be productive. It is in their right to use algorithms to analyse the data that they gather from your work patterns. It is perfectly fine to be able to judge someone as having higher value in comparison to others in the company based on their productivity.

The main reason being is that, humans are capable of doing the same thing, although to a much lesser degree since we are not able to process data as quickly as computers. It is perfectly acceptable for people to do the same in which we would judge people based on how they work. In fact, we may be even more biased in comparison to computers. People may dislike someone for a reason that is not work related. But as a result, they would devalue the work that another person does. Whereas algorithms likely won't take into consideration intrapersonal relationships. Perhaps, there are also situations where someone is being toxic but still does the work that they are assigned to do with an above average capacity. However, as a result of their toxicity, the general workplace atmposphere may be demotivating and as a result general productive output may have decreased. Here an algorithm for the same reason above is not able to figure out the specific toxic person however a human may be able to make a call based on reports from other coworkers or while interacting with that person.

Another factor that we should take into consideration is that algorithms are very influenced by the data that they receive, back in 2016 Microsoft had an AI experiment where they wanted it to become a twitter account and behave like a twitter account would. Some people heard of this and decided to influence the bot. As a result, it started to behave like a far-right nazi. Another example was that the NYPD used an AI to predict future crime locations. However as a result of the data that they used. The prediction would almost always be around black communities and wouldn't predict any crime by white people. This while unexpected intially, wasn't too surprising based on the data. In China technologies like this is used in order to track social credit score as well as predict future crime on a governmental level.

These factors make it difficult to trust in the systems that companies could use as a result of the data that they gather from the workplace. Another issue that

pops up would be that data is not safe from an unauthorised individual or group accessing it. There would be a significant problem if workplace analytics was leaked to other companies, revealing secrets about employees as well as company secrets from different repositories from the indivduals working on them. This is a deeply concerning problem that needs to be addressed with any discussion regarding data gathering. It is also a problem that resulted in the creation of GDPR in the EU that tries to protect the privacy rights of individuals. Although that came from a different controversy. Laws like GDPR in the EU require some of the data gathering to be monitored and to be able to be deleted or requested whenever the user demands. The user should be able to receive requested information after a few days or so.

In general, I don't actually have any objection for companies to gather data since in my opinion it is fine for a human being to do it morally. However the usage of the data may lead to some immoral situations. But, in my opinion that is not the source problem behind the issue of data being used in an immoral manner. I don't know what the solution to the answer is, but not allowing data being collected feels like using cellotape to fix a leak. It doesn't address the general problem of data being used in an immoral manner. In the EU since we can request the data to be deleted as long as its not vital information I believe that companies are well in their right and the law gives some sort of a moral check to it.

References

[1] Kazuo Yano, Dr. Eng.Tomoaki Akitomi Koji Ara, Dr. Eng.Junichiro Watanabe, Dr. Eng.Satomi Tsuji Nobuo Sato, Ph.D.Miki Hayakawa Norihiko Moriwaki, Dr. Eng., *Measuring Happiness Using Wearable Technology*, Hitachi, Japan, 2005.

[2] S. Achor, *Positive Intelligence*, Harvard Business Review, 2012.

[3] Watts S. Humphrey, *Discipline for Software Engineering*, 1994.

[4] Ken Schwaber, Jeff Sutherland, *The Scrum Guide*, 2017.

[5] Sasha Rezvina, *Velocity vs Gitprime*, 2020.

[6] Philip M. Johnson, *Hackystat*, Researchgate, 2007.

[7] Matthew J. Lindquist, Yves Zenou, Jan Sauermann, *Network Effects on Worker Productivity*, Researchgate, 2015.