Measuring and assessing the Software Engineering process.

Contents

Brief

To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Introduction

This report will look at how the software engineering process can be measured and assed in terms of measurable data, the computational platforms and algorithmic approaches available to measure this data and the ethics concerning collecting this information on software engineers. Firstly, I want to discuss what exactly Software engineering is, Ian Sommerville defines it as “Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system”. According to Sommerville it includes four fundamental activities which are:

- Software Specification
- Software Development
- Software Validation
- Software Evolution.

This means all four activities must be taken into account when measuring the software engineering process.

Secondly, I want to look at why one would want to measure this process? As a business measuring and assessing your software engineering process can identify, prioritize, track and communicate any issues to management in order to foster better team productivity. Businesses can use this assessment to communicate the status of software development projects, pinpoint and address issues, and monitor, improve on, and better manage their workflow. Both of these ultimately lead to better efficiency and most importantly for businesses reduced costs.

Using *Measurable data* to asses and measure the software engineering process.

Software Metrics - History

The measurable data in Software engineering is referred to as software metrics. In order to give context to the use of them in today's world we must look at the history of them. According to Norman Fenton this term describes the wide range of activities concerned with measurement in software engineering. The aim was to provide information to support quantitative managerial decision-making during the software lifecycle. These activities first started out according to Fenton as lines of code (LOC) as it was the simplest method to infer the amount of work done. However it is not very effective, see the below example of three code snippets, all of which store the maximum of two variables to a variable called max (R0). If we used LOC as a metric to measure the productivity of the two java snippets, it would appear the engineer who used the if-else statement was more productive however both code snippets achieve the same goal with the ternary statement being better practice. Similarly, one cannot compare the ARM code with java code as the ARM code gives the illusion of more productivity despite providing the same functionality.

Ternary Statement Java

```
int max = (a < b) ? b : a
```

If-else statement Java

```
int max;

if(a < b)

        max = a;
else
        max = b;
```

Arm Assembly

```
 MOV R1, #2

 MOV R2, #3

 CMP R1, R2
 BGT max1
 LDR R0, R2

 B END

max1

 LDR R0, R1
END
```

Other metrics were later introduced such as duration of testing process (Hours), number of defects discovered and time a programmer spent on a project. These metrics only measured and assessed the software development and software validation stages of the software engineering process and even then, they were not very successful at doing so. This resulted in numerous academics researching the area trying to come up with solutions that stopped software metrics being based on LOC or other ineffective metrics however unfortunately there was no improvement in the use of metrics in industry to match the exponential growth of its use in academia.

Why was this? Fenton gives 4 reasons as to why companies did not change the way in which they measured software metrics, they are as follows:

- **Most of the research was irrelevant** - Most of the academic research could only be applied to small programs and not large-scale projects companies actually worked on.
- **Metrics were too detailed** – Much of the research looked at very detailed code metrics that was of little interest to businesses.
- **Metrics was seen as a last resort** – Metrics were only really used as a 'grudge purchase' when things were not running smoothly or to satisfy a third party such as investors.
- **Low Priority** – it was the first thing to go when budget cuts were made, or funding was low.

Software Metrics – Present

The collection of software metrics has evolved overtime and we have fortunately stopped using LOC as the main metric. We have moved on to measure things much more meaningful about the code developers write such as:

Impact

Impact measures the effect of any code change on the software development project. A code change that affects multiple files could have more impact than a code change affecting a single file.

Lead Time

Lead Time is the time period between the beginning of the software engineering process and its delivery to the customer.

Cycle Time

A measure of the elapsed time when work starts on an item (story, task, bug etc.) until it's ready for delivery. Cycle time tells how long (in calendar time) it takes to complete a task.

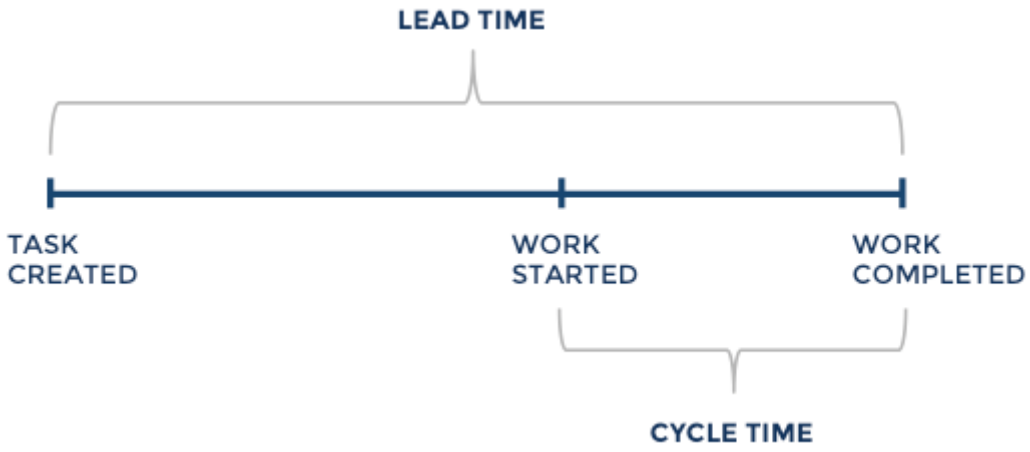

Figure 1 - source: screenful.com/blog/software-development-metrics-cycle-time

Code churn

The percentage of a developer's own code representing an edit to their own recent work. It is normally measured in LOC that were modified, added and deleted over a short period of time. When churn starts to spike, this can be an indicator that something is off with the development process and the particular developer may need assistance.

Code Coverage

A measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. A program with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.

Not only do we look at measuring development and testing metrics but companies are now measuring far less traditional metrics such as:

Influence monitoring

Measures the effect and importance a certain employee has based on 4 metrics; Size of their immediate circle, how business critical the person is, how connected a person is and then measures their influence to give the metric mentioned above. This will be discussed this in further detail under computational platforms.

Location

Keeps track of where in the office an employee goes and what time they go there at.

Tone of Voice

Uses technology to decipher employees tone in order to see how effectively they are communicating.

Other basic and common sense metrics are also kept such as number of employees, work hours, cost of project etc.

An overview of the computational platforms available to gather these software metrics.

Computational Platforms

It would take way too much resources and time for companies to develop their own applications to gather these metrics. There are many different companies that specialise in building these platforms to collect and display software metrics. Listed below are some of the tools available:

Github

Github is the most popular online version control system and used industry wide. Supplies users with analytical tools to monitor the productivity of users that commit to induvial

repositories. Github provides metrics such as commits, code frequency, code frequency, and dependency graphs. The advantages of Github is that is already widely used so companies do not have to change anything they are already doing in order to integrate Githubs analysis into their software engineering process. The disadvantages are that it lacks complex observations that GitPrime and Hackystat provide.

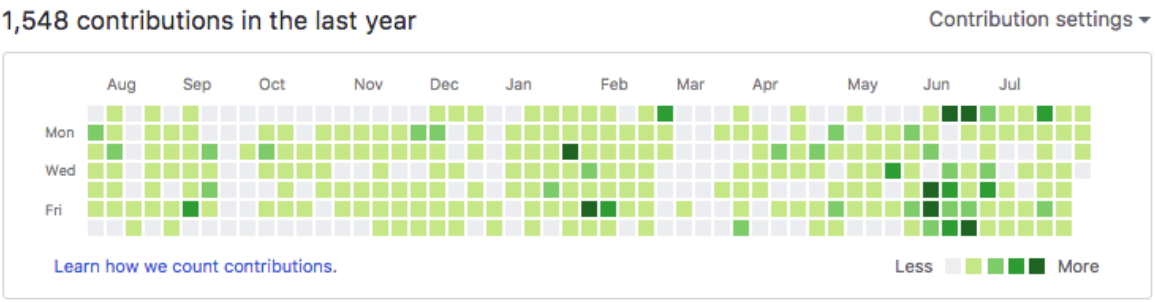

Figure 2 – Github contribution heat map.

GitPrime

GitPrime is the first and one of the largest companies in performance measurement, have clients such as Disney, Adobe, Tesla and Atlassian. GitPrime targets non-technical managers by converting complex software developmental work into concrete data and analysis to help leaders make decisions. GitPrime records and analyzes data in many metrics, including personal metrics such as number and time of commits, amount of technical debt, and defect rate, as well as organizational metrics including level of activeness in team and progress on current tasks. By looking at the analysis rather than the actual code, managers are able to identify developers who are struggling and need help. In addition, GitPrime not only monitor developer activities on a personal level, but also view the team as a whole, providing services to track the progress of the team and team members, as well as setting development goals.

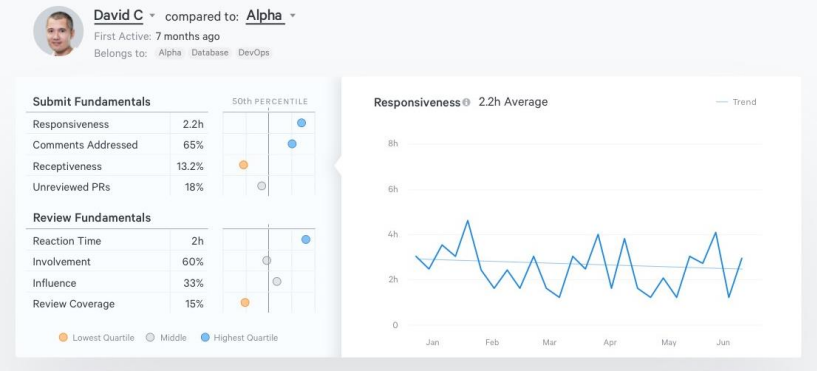

Figure 3 – GitPrimes dashboard on a particular engineer

GitClear

Launched in 2017. GitClear targets managers with a technical background, so GitClear is more focused on codes as well as have a more clear and precise view. GitClear provides managers with a more in-depth examination into the specific code and lines, and a powerful code review tool for more efficient code review process. GitClear calculates different metrics such as code churn, activity type, related commits and ignorable files, which are factors based on a single line, a commit, a file or a branch, resulting in a single metric called “Line Impact” that is used for evaluation. Being a tool that targets technical leaders, it could allow managers to modify the calculation of Line Impact based on their own knowledge or to their preference.

Status Today

A London startup that is building out AI tech that it claims can help companies better understand their employees and in turn improve productivity. Status Today currently plugs into various online company tools, such as those from Microsoft or Google. It then uses meta-data pulled in from these systems and artificial intelligence to analyse employee actions, thus enabling companies to gain better visibility of how their workforce is operating and to make improvements accordingly. They do this by utilising collaboration network charts and workplace wellbeing dashboards.

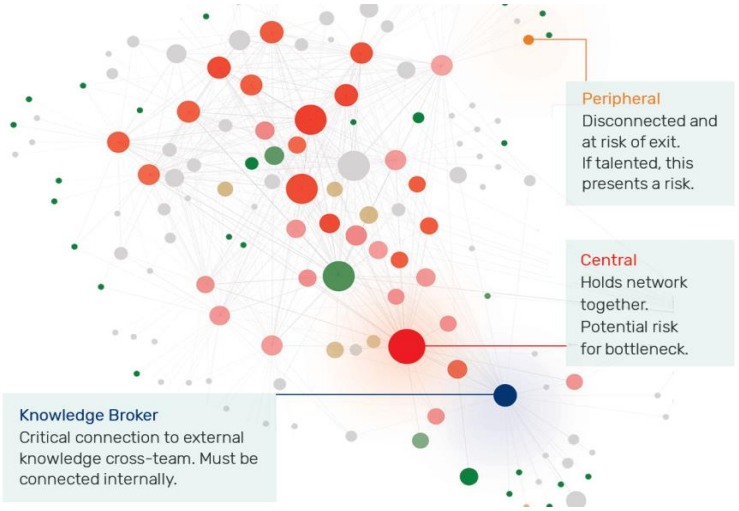

Figure 4 – Shows Status today collaboration network

Steelcase

A company that measures posture, sitting habits, stress levels, heart rate and breathing by using a mechanized chair that reads these metrics using technology built into the chair. This information is fed into an online dashboard and can help employees look after their physical and mental health thus leading to more productivity.

As shown above there are so many platforms to choose from with each one offering something different to the next with new businesses entering the market every year it will become a very competitive sector which will result in the constant improvement and evolution of current tools.

An overview of the algorithmic approaches available to gather software metrics.

Algorithmic Approaches

There are also numerous algorithms that utilises raw data collected by programs like Gitprime and Gitclear.

These following algorithms are the ones I found most interesting:

Halstead Complexity Algorithm

A computer program is considered to be a collection of tokens which can be classified as either operators or operands. Halstead's metrics are included in a number of current commercial tools that count software lines of code. By counting the tokens and determining which are operators and which are operands. The following equivalences are made:

n1 = Number of distinct operators.
n2 = Number of distinct operands.
N1 = Total number of occurrences of operators.
N2 = Total number of occurrences of operands.

Halstead then uses various algorithms/equations to calculate numerous metrics from the above. The metrics are:

Halstead Program Length

The total number of operator occurrences and the total number of operand occurrences.

N = N1 + N2

Halstead Vocabulary

The total number of unique operator and unique operand occurrences.

n = n1 + n2

Program Volume

Proportional to program size, represents the size, in bits, of space necessary for storing the program.

V = N * log2(n)

Potential Minimum Volume

The potential minimum volume V* is defined as the volume of the most succinct program in which a problem can be coded.

V* = (2 + n2*) * log2(2 + n2*)

Program Level

The higher the level of a language, the less effort it takes to develop a program using that language. The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

L = V* / V

Program Difficulty

This parameter shows how difficult to handle the program is.

D = 1 / L

Halstead also provides metrics to calculate many other metrics such as:

- Programming Effort – E = D * V
- Language Level – L' = V / D / D
- Intelligence Content – I = V / D
- Programming Time – T = E / (f * S)

Cyclomatic complexity

Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code.

The following formula is used to calculate cyclomatic complexity (CYC):

CYC = E – N + 2P

In this equation:

P = number of disconnected parts of the flow graph (e.g. a calling program and a subroutine)

E = number of edges (transfers of control)

N = number of nodes (sequential group of statements containing only one transfer of control)

The ethics concerns.

Data collection and analysis is not just a concern for software developers but for most individuals across all forms of industry. The need to increase workplace efficiency in order to maximise productivity and profits, and the most widely used practice for this is to collect data on exactly how every step of your work is done in order to highlight errors or bad practices that can be improved upon using a similar platform to the ones mentioned previously. This has led to nearly every workplace being monitored by some sort of automated data collection. This raises questions about what data should you be allowed to store and analysis and what crosses that privacy boundary for that individual.

I believe analysis of developers is necessary and beneficial, however I do believe there is a line in which companies should not cross. Tools such as Humanyze, that tracks tone of voice and location, crosses this line. They collect roughly 4GB of data per day on each employee which I find absurd and demonstrates a clear lack of trust between employer and employee. In my opinion this creates a toxic work environment which lowers efficiency thus meaning Humanyze has the adverse effect of what it aims to achieve. Companies such as Gitprime and Gitclear collect meaningful data that can show management how each of their team is working, any competent developer shouldn't mind this monitoring as it will show the amount of work they do and can help them improve on their skills by highlighting their errors and flaws.

Bibliography

https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software%20Engineering%20by%20Somerville.pdf

https://pdfs.semanticscholar.org/6a21/992265d7589096b729dc606bc6e10db214e1.pdf

https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/

https://stackoverflow.com/questions/2028213/is-hackystat-only-academically-interesting

https://techcrunch.com/2018/02/20/statustoday/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=0C52xzhDOUmEoWTvIhxIVA

https://www.perforce.com/blog/qac/what-cyclomatic-complexity

https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/