Report on Measuring Software Engineering

Report on Measuring Software Engineering 1
Measurable Data **2**
Size-Oriented Metrics 2
Function-Oriented Metrics 2
Different Code Measurements 3
Project Burndown 3
Code Churn 3
Refactoring Rate 3
Ticket Close Rate 3
Git Commits 4
Measuring non-Code Factors 4
Job Satisfaction 4
Workplace Activity 4
Computational Platforms **5**
GitPrime 5
SonarQube 5
Waydev 5
Humanyze 6
Use of These Platforms. 6
Algorithmic Approaches **6**
Halstead Metrics 6
Software Maturity 7
Function-Based algorithms 7
Ethical Concerns **8**
Conclusion **9**
Bibliography **9**

Software is in demand in the modern world, and requires software engineers to build it. Software engineering is described most simply by Fritz Bauer as,

"the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

It follows then that software engineers are the people who carry out this task whether alone, in small teams, or as part of some large corporation. In order to judge the efficiency of their work, engineers or the people who manage them would be served by methods with which to measure both the progress of a software project, and also the efficiency or capability of engineering teams or single engineers. Here several methods of how this measurement is attempted as well as the difficulties and ethical concerns therein will be discussed.

Measurable Data

What data can be used to measure a piece of software's quality, staging, efficiency etc. Various types of data can be used to assess these ideas with various advantages and disadvantages.

Size-Oriented Metrics

These measurement is taken by checking the size of the software produced in some way, this includes how long the project has taken, how many people are working on it, the amount of money spent, the number of errors or defects in the project, the lines of codes used and more. When considering a project, for a company it would be important to know how much money it is costing, as well as how many people are working on the project. Lines of Code can be chosen as a normalisation value, this means that the other measurements are tested based on how many lines of code were produced. For example, the errors per KLOC (thousand lines of code), the cost per KLOC etc. This can be valuable as a quick shorthand for how much work is being done, and it's fairly easy to count, however there is no guarantee if all the lines of code are necessary, or even if they are of a decent quality. It's quite easy to pad out code with white space or rewritten code when compartmentalising to a function would have worked. It is also programming language dependent. If a manager wished to test which of his employees was doing better work, seeing that one employee had written 500 lines of ARM code vs another employee's 40 lines of Java doesn't mean that the second employee is producing less valuable work.

Function-Oriented Metrics

Function-Oriented metrics use the functionality of the product as their normalisation value. The function point metric was proposed by Allan J. Albrecht and is a means for measuring a system's functionality. It is a formula using countable measures of the software. These are:

1. Number of External Inputs.
2. Number of External Outputs.
3. Number of External Enquiries.

4. Number of Internal Logic Files.
5. Number of External Interface Files.

This report will discuss the formula in a later section but just for the possibility of measuring, it is a valuable resource. The data measured will be calculated into a number (FP). The idea is that this FP can give an idea of how much a system will cost, how much time it will take etc. The difficulty lies in how much of the data can be measured correctly. Computation values are subjective rather than objective and when the FP is found, it means very little on its own. Its value is being compared to other projects' FP value and estimating cost, and time and other factors.

Different Code Measurements

Instead of using Lines-of-Code as the measured value there are other things that can be measured to check the status of a project or engineer.

Project Burndown

Some tasks on a project are more difficult than others, so if the tasks can be split into "story points" or objectives that should all take a similar amount of time, effort, cost to finish. If the amount of time to finish the first 5 stories a team undertook not the current project was greater than the amount of time it took for the first 5 stories to be finished on previous projects, perhaps development is going poorly. This isn't definitive, the story points are only based on an estimation, and could be completely off.

Code Churn

Code churn is the percentage of a developer's code that is an edit to recent code. The recency is up to the measurer in a small project it could be a week, a larger one a month. Measuring the amount of code added, deleted, and rewritten gives an idea on a regular basis what the churn has looked like over time. If the project's code churn is regular up to a point, and suddenly increases, there may be something amiss. This could be valuable for measuring teams and engineers as their normal workflow can be evaluated over time.

Refactoring Rate

The refactoring rate is the ratio of code that is new agains the amount of code that is rewriting legacy code(maybe code over a month old, or from the beginning of the project). Ideally in software engineering the refactoring rate will be low, a sign of quality software is its maintainability and resilience. If it works for a long time without extensive reworking, more time can be spent improving it, or on a different project. The refactoring rate is not an exact science, but it is valuable to keep track of to see if it is changing drastically over time.

Ticket Close Rate

Ticket Close Rate is measuring story points like Project Burndown, but just checking how many story points were closed in a certain period of time, if none or few stories have been closed in that time

perhaps the developers are having difficulty, or unfortunately maybe those stories just take more time than previous ones.

Git Commits

If the team is using a source control system, measurements could be taken about the number of commits individuals or the team are doing. This may not be useful on its own as engineers could be committing trivial changes, or even committing after large changes. Like the other measures however, if there is a large enough history of data taken about it, it could be useful to know if something very irregular happens like a team's commits drop off or increase substantially. It might not mean anything but it could be useful to discuss the reasons for the change. Committing to source control also allows the use of computational platforms that will be discussed later.

Measuring non-Code Factors

Sometimes instead of directly measuring the code written or the software, it's decided to measure something else that could give an idea of how the project is going.

Job Satisfaction

This measurement is based on the premise that a happy worker is a productive worker. A study from the University of Warwick suggests that this is true. If measuring productivity through measuring code is difficult, this correlation is potentially exploitable to check for productivity. Surveying engineers with the Net Promoter Score to check for job satisfaction could give a manager or team lead an idea of how likely the engineering team's productivity is. Methods like this are used because of the idea that measuring software engineering is so difficult or impossible that it is better to measure a proxy for it and extrapolate.

Workplace Activity

Instead of just measuring the product being made and the code put into it, employees can be measured by their activity in the workplace. For example, the hours that they work, the people they communicate with through workplace channels, the websites that they use, or how often they work with others. These kinds of measurements can be used to see who is influential within the team, maybe who is due a promotion or who has potential to leave the company soon. These measurements are more relevant to assessing the individual performance of team members as it can be seen who is adding value, whereas the amount of code someone writes might not mean as much.

Despite the difficulty with knowing whether data is appropriate or relevant, it is clear that measurement can and are taken in the workplace. The problem lies in knowing what data is useful to assessing the software engineering process and what can be cast aside. From the several examples laid out it would seem that it is best to gather data to keep a general record, and then use it to see if there is any obvious changes

Computational Platforms

There are a variety of computational platforms available to try and process the above data and give feedback about the engineering process. Some of these platforms are very successful as it is believed the analysis they provide is valuable to management, and enables them to make better decisions than they would using their instincts or more analogue management methods.

GitPrime

GitPrime is an organisational tool based around collecting data from the team's source control. It aggregates git data and presents reports and insight about them. It has three platforms it operates on. The code platform shows informations like the amount of commits, ticket activity, and presents graphs about how much the codebase has been impacted. It shows the activity of individual developers so they and their supervisors can see their contributions. The review platform shows the amount of pull requests, commits, comments etc. It's used to check for inefficiencies in the work process such as unused pull requests or disagreements in code reviews. Patterns are revealed about how the code review process is working over time. The collaborate platform shows how the engineering team is working together. If one person is a clear knowledge base for the team it is shown and a manager may then decide to try and train or team up other members to distribute the knowledge more evenly. It shows which team members are collaborating with others and which are not.

SonarQube

SonarQube is less of a team-analysis tool than GitPrime, its main purpose is to measure code quality. The code analysis it does is for detecting errors, or code that is repeated or untested. It checks for potential security risks in the code as well. When looking through the code it looks for what the Platform call "code smells" and points out that something, while maybe not wrong, is over-complicated, or could be looked over. It does offer a Pull-Request analysis system which is most useful for measuring how the engineers in the team are performing. If the new code is worse than the old code it will point it out, or show where it is failing.

Waydev

Waydev is another source control assessment platform and is used to show the work of engineers in a team. It operates with two metrics in mind, what it calls 'Impact' and commits per day. Impact according to Waydev is,

" a metric which shows you how much "cognitive load" did the engineer carry when developing these changes. For calculating it, we give each commit a score based on how much new work, legacy refactor, help others and churn did the programmer and, besides, we look at how many files did he modified and the points where these changes had been made."

It is clear that this platform is using some of the same measurements mentioned above: code churn, refactoring, commits etc. It shows all of this data as well as their own impact rating, useful for both

clarifying what is meant by impact, and letting a manager ignore it if they feel it is irrelevant. Commits per day is more clear, simply showing how much the team commits, it provides the ratio of total commits and the number of days with git activity.

Humanyze

Humanyze offers “The Elements Platform”, a platform for pulling and collating data. The data it uses unlike Waydev or SonarQube etc, are systems such as Skype or email. While this data is not software engineering specific it allows team knowledge to be gathered for use and understanding. The kinds of factors it is trying to quantify are things like collaboration and communication. The company says that it is committed to using only ‘metadata’ to check for patterns in communication in order to analyse it. One of the case studies they advertise was that working with the European Bank, they found that the strongest branches were those with the most face to face interaction, the weaker branches had offices designed in such a way that discouraged communication between people this way. To gather data in cases like these Humanyze have used sociometric badges that track the locations of employees, to see when they are interacting with each other, and who they are interacting with. These have broad applications but are clearly applicable to software engineering in a workplace environment.

Use of These Platforms.

SonarQube seems to be a platform that is useful to both the engineer writing the code, and the manager of the team, though perhaps more useful to the engineer. Its primary focus is on code quality, and is useful for 27 programming languages. In comparison GitPrime and Waydev are more focused on team analysis and are useful for management. Showing the impact certain developers have lets management see who is leading the team and who may be lagging behind, or is stuck in a difficult area. The kind of coding they are doing may allow managers to link up team members to collaborate or learn from one another.

The work Humanyze does with the Elements Platform is more useful for changing the social environment of the workplace than the code specific measurements. The data mined from communication, or the tracking of employees with badges means that management can change how the workplace is set up to take advantage of positive interaction.

Algorithmic Approaches

Assessing data collected from software engineers is according to some a fools errand. The strategic technology company Nortal has suggested that there doesn’t exist an objective metric of developer productivity, and the problem of it will remain unsolved. While it may be difficult to find a truly objective metric of productivity, the use of algorithms to assess it may still be useful, especially as comparison points.

Halstead Metrics

Halstead metrics use size-oriented metrics to assess the length of a program, the volume(number of bits), language level, and factors like development effort, development time and projected faults. The measurements it uses are:

n_1 = the number of distinct operators that appear in a program.

n_2 = the number of distinct operands that appear in a program

N_1 = the total number of operator occurrences

N_2 = the total number of operand occurrences

The length N can be estimated according to Halstead with

$$N = n_1 log_2 n_1 + n_2 log_2 n_2$$

While the program volume can be defined as

$$V = N log_2 (n_1 + n_2)$$

For an example when applying the Halstead metrics to testing using definitions for Volume V and program level PL effort e can be calculated as

$$PL = 1/[(n_1/2)x(N_2/n_2)]$$

$$e = V/PL$$

These laws have generated substantial controversy but when subject to experimentation have revealed to be in general loosely correct.

Software Maturity

The IEEE Standard Dictionary of Measures of the Software Aspects of Dependability offers an algorithm for checking the Software Maturity Index, which provides an indication of the stability of a software product. The following factors are measured:

M_T = the number of modules in the current release.

F_c = the number of modules in the current release that have been changed.

F_a = the number of modules in the current release that have been added.

F_d = the number of modules from the preceding release that were deleted in the current release.

Then the software maturity index is computed as follows:

$$SMI = [M_T - (F_a + F_c + F_d)]/M_t$$

As SMI approaches 1.0 the product begins to stabilise.

Function-Based algorithms

Function-Based Metrics also have formulae associated with them to create projects estimations from them. To find the FP the following relationship is used

$$\text{FP} = \text{count total x } [0.65 + 0.01 \text{ x } \Sigma(F_i)]$$

where count total is the sum of all FP entries from the following picture

and F_i(i = 1 to 14) are Value Adjustment Factors based on14 questions about the type of program and system. The FP can be compared against precious Fps to make an estimation on the difficulty and length of the program.

The preceding algorithms all assess an aspect of the software itself, it's possible that a platform like SonarQube is making use of some of these algorithms.

Ethical Concerns

The use of developer data in assessing the performance of engineers is of ethical import. What kind of data should be allowed for use and what kind is off limits? When and where should a developer be measured? How can the engineer be made both aware of being measured and allowed and encouraged to give vocal consent or dissent.

The first thing that is important to note is GDPR guidelines, the EU General Data Protection Regulation that insists on data protection and privacy for all members of the EU. The effect of it can be seen clearly to Internet users when they use websites, but how does it affect software engineers. First of all any personal or private data that could be collected from an engineer should not be identifiable to that person. So when for example Humanyze are collecting communications data, they only collect metadata to avoid breaking this statute. Consent to having data measured must also be given freely, unambiguously, and without coercion. This means that an employer should not be able to force you into having your personal or private data collected by them for assessment purposes.

Despite that it is clear that monitoring will and is taking place. GDPR is not a catch-all for data protection and aspects of engineering work will be monitored and may begin to be monitored in new ways. Therefore it is necessary to not just discuss what is legally permissible, but ethically sound.

The workplace is where most working people spend a significant amount of their waking days, an 8 hour workday is roughly half the waking day an adult should have. So a large portion of the social life of a

person is spent at work. If for example the technology of Humanyze like their biometric scanners are used, workers are being measured not just on their working ability but on who their friends are, when they take bathroom breaks, do they leave to go for lunch or not. This is not all new, a watchful manager could be aware of this, or even CCTV security footage inside buildings could be checking this. It is just the same surveillance on an increased level.

The more worrying concern is with workplace and employee protections. If communication platforms and employee location are recorded, the ability for employees to discuss dissatisfaction with their workplace is jeopardised. The right for employees to unionise is strained when the people they may be unionising to be in conflict with can check who is talking to who, and who may be a potential cause of trouble for the company.

These are the more worrying aspects of the data gathering, but Platforms like GitPrime or Waydev have less potential for antagonistic use. Largely the data collected is just the work that the employees do, the code they submit and how often they submit it. As long as the data collection is relegated to the software they are working on the only issue that could arise is bad management decisions. If GitPrime shows that an employee is committing less than before and a manger decides to reprimand an engineer because of it, it is only a fault of the technology in so far that it is of course limited. The human element of decision making is flawed as well, but a competent supervisor should know when to take the findings of computational platforms literally, and when to think about underlying causes of discrepancies.

Conclusion

In conclusion, the measurement of software engineering is not a sure bet, there are various approaches to how to gather data and assess it. The products used by companies to track and assess the software and the software engineer have plenty of uses, but need to be properly understood to use, and the platforms that assess more than just an engineers work should be considered carefully as to how ethical their use is and how employees in the workplace might protect themselves from unfair management practices.

Bibliography

Pressman, R. (2006). Software Engineering: A Practitioner's Approach. 6th ed. New York, NY: McGraw-Hill.

Gitprime.com. (2019). [online] Available at: https://www.gitprime.com/ [Accessed 5 Nov. 2019].

Waydev.co. (2019). [online] Available at: https://wwww.waydev.co/ [Accessed 5 Nov. 2019].

Sonarqube.org (2019).[online] Available at: https://www.sonarqube.org/ [Accessed 5 Nov. 2019].

Humanyze. (2019). *Humanyze - Analytics For Better Performance.*. [online] Available at: https://www.humanyze.com/ [Accessed Nov. 2019].

Citizensinformation.ie. (2019). *Data protection in the workplace*. [online] Available at: https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/data_protection_in_the_workplace.html [Accessed 6 Nov. 2019].

Medium. (2019). *You Can't Measure Software Engineering Productivity, so Measure Job Satisfaction Instead*. [online] Available at: https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451 [Accessed 5 Nov. 2019].

Nortal. (2019). *Myth of developer productivity*. [online] Available at: https://nortal.com/blog/the-myth-of-developer-productivity/ [Accessed 7 Nov. 2019].

Best Online Tutorials | Source codes | Programming Languages. (2019). *Software Engineering-Function-Oriented Metrics*. [online] Available at: https://www.1000sourcecodes.com/2012/05/software-engineering-function-oriented.html [Accessed 8 Nov. 2019].