Measurement of Software Engineering

Chihun Lee

November 20, 2018

Abstract

A report on the practice of measuring software development. This report will focus on the metrics and methods used, technologies that are available as well as the ethics and validity of the practice itself.

Contents

1 Introduction

In recent times, within the Software Development industry there has been a growing push to set a standard for measuring the productivity of an individual developer or a development team. This has resulted in new concepts, metrics and technologies being invented and pioneered in order to quantify in terms or metrics and statistics the once thought intangible idea of measuring the value of programming code into numbers. There has of course been some push back to the growing tend. Questions of validity to the methods of measurement as well as questions regarding the ethics of the measurement has been raised.

2 Common Metrics and Methods

2.1 Lines of Code

Measuring the Lines of Code(LOC) is one of the most widely metric in the industry. LOC generally refers to the count of lines of code disregarding whitespace and purely lines of comments, that are in a project. Lines of Code is often also known as Source Lines of Code(SLOC). There are severe advantages and disadvantages to this metric.

Some advantages of LOC include:

- Easy to Understand: The metric of lines of code is inherently quite intuitive and therefore is easy metric to discuss without much level of expertise or knowledge.
- Easy to Measure: Counting the number of lines of code is a fairly easy task and is a statistic that is easily attainable. Some even simple technologies can such as text editors, IDE's and Git control's can provide this information.
- Broadly Used Standard: Due to the above two factors LOC has become a widely used standard in the industry. This has led to certain standards being set such as K-LOC which groups LOC into one thousand sets.
- Decent for Broad Measurement: LOC is mainly useful for finding a broad view of the size of a task or project. There is a general correlation with the time a task will take and the complexity of a task with the the amount of LOC. This correlation becomes stronger the bigger the task being measured.

Some disadvantages and weaknesses of LOC include:

- Variance of LOC for different Factors: It is true to say that the same task with the same functionality can be implemented, through many different methods and therefore the differing number of lines of code. Therefore it is hard to measure the complexity of a task using just LOC.
- Lack of Measure for Efficient, Quality and Redundant Code: LOC fails to measure how efficient a programmer can be or other factors of quality of code such as readability, maintainability and functionality of the code. It

also fails to account for redundant code, so a programmer who produces lots of redundant code could look more productive than one that produces cleaner code.

- Variance in Different Programming Languages: Two projects that are written in two different programming languages which have the same functionality will differ in size due to the nature of different programming languages which have different characteristics. On the more extreme example a task in a very low level programming language such as some assembly language will probably take much more LOC that that of a language using a much higher level language.

2.2 Git Activity

A set of metrics that can be measured is Git activity. There are a number of different pieces of data that can be gathered from this including:

1. Number of Commits
2. Frequency of Commits
3. Size of Commits
4. Number of Pull Requests
5. Stories Closed

These metrics share similar strengths and weaknesses that LOC has. The data that they provide are very broad and do not account for the complexities of software development.

- A developer could easily have a large and frequent number of commits and still be doing essentially nothing as each commit they submit might have no useful product to them.
- On the other hand a lack of commits is a much better indicator of an unproductive developer. If a developer is not commiting regularly then it is a good sign of someone that is not producing much useful code.
- A developer could hypothetically be doing lots of work and committing infrequently and therefore the size of each commit could be quite large, however this is widely regarded as bad practice, as it is detrimental to the developer themselves and is harder for others looking through the commit history to see when and how changes were made.

2.3 Tasks Completed and Bugs Fixed

The number of tasks finished and bugs fixed has to come with the consideration of the size of the task and the difficulty of fixing the bug. The task may be also divided into smaller sub-tasks and so forth. This again is a very broad metric much more suited to project managers who can manage when to plan to complete and move on to further tasks.

2.4 Testing

Ideally all code should be thoroughly tested and this means passing a number of factors. From testing we can in some ways identify the quality and the reliability of the code.

2.4.1 Code Coverage

Code coverage is an important metric as it shows whether or not to the line of code whether or not it has been tested. It might not show if has been tested thoroughly but it is an indicator of when lines of code have not been tested.

2.4.2 Passed Tests

Number of passed tests. Realistically all tests created should pass otherwise it shows that there is a flaw in the program or in the test. Now counting the number of passed tests is important and it is likewise with many other metrics to note that some quality of tests are better than others.

Conversely the number of non passed tests can be used to measure the number of bugs found.

2.5 Time and Speed

Measuring the hours worked by a software engineer is metric mostly useful for people in management and financial positions to decide aspects such as salary and other human factors.

1. Lines of Code
2. Git Activity
3. Tasks Completed
4. Bugs Fixed

These inherit most of the aspects of that come with the metrics with the additional benefit of being able to account for time.

3 Complex Metrics and Algorithms

3.1 Code Churn

Code Churn measured the rate a piece of code from a developer evolves as they re edit their existing code. This is generally done algorithmically through tracking the changes of the lines of code from commits over time. This is an important metric as if a developer has to go back and re edit their old code it generally means it was not the best quality of code in the first place, and is now wasting precious development time redoing work that should have been satisfactorily completed before.

3.2 Technical Debt

Technical Debt is the concept that a functionality that was implemented in a simple and easy manner instead of implementing the best possible solution which would of been harder as and taken more time in the first place, will lead to a 'debt' as the lower quality of code will hamper further progress as it will be harder to develop more work using the previous work or time will be needed to redo the work that should have been better in the first place. This leads to higher rate of code churn.

3.3 Impact

To measure the impact of a piece of code has on a project is another metric. This is a very complex metric to solve, but is essentially the holy grail of software engineering measurement. The goal to measure the impact of some code is attempted to be measured by a number of factors such as how often it is used by following pieces of code, how difficult it would have been to implement and the criticality to the project. These metrics are themselves complex to measure.

3.4 Code Quality

There are many aspects to code quality, some of which I've covered in previous sections. In this sections I will discuss possibly the two key traits of code quality. Efficiency and readability.

3.4.1 Efficiency

The efficiency of the code may refer to the collection of all the factors mentioned above where code written has low code churn, low technical debt and high impact. Code that matches all these criteria can be considered efficient and therefore generally very good quality code.

3.4.2 Readability

Another major aspect of good code is readability. This is essential for when maintenance is needed and the same programmer or another is in the future trying to modify the code they must first figure out what the code that already exists does. This is especially important when there is little documentation for the code.

4 Technologies and Organizations

4.1 Version Control

Version Control provides some tools that can show many of the metrics of git activity. It can also be used to abstract more complex pieces of data.

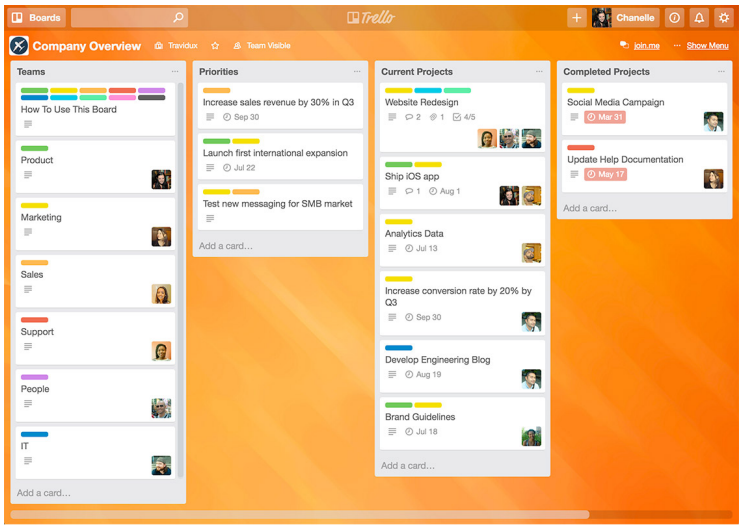

Figure 1: Trello Dashboard

4.2 Project Management Tools

Project management tools such as Trello and Astana can be used to keep track of how many tasks were finished when they were finished and the subtasks that are related to those tasks.

4.3 Hackystat - Automated Metric Collector

Hackystat Project was started in a research which it showed that even a small anount of work was required adoption of measurement software either failed or did not last for any significant period. It then developed a tool that would do the metric measuring automatically.

Hackystat is inherently useful as little additional work is needed for Hackystat to collect the data needed for the metrics. Its automatic sensors collects the data in an nonintrusive manner. Hackystat will then analyse the data and return relevant feedback from its findings.

4.4 Static Object - Example of Service Available

Static Object is a company that offers services which they say can accurately measure software development productivity. They attempt to accomplish using a number of different concepts and data such as:

1. Line Impact. This is their key metric which much of their other metrics also rely on. Line impact attempts to measure the value of code by combining a number of different stats that are measured and see the effect each line of code has on the development of a piece of software. They try to make a more accurate measure by combining a number of variables which paint a more complex but truer picture. The key factors include lines, files, commits and branches.

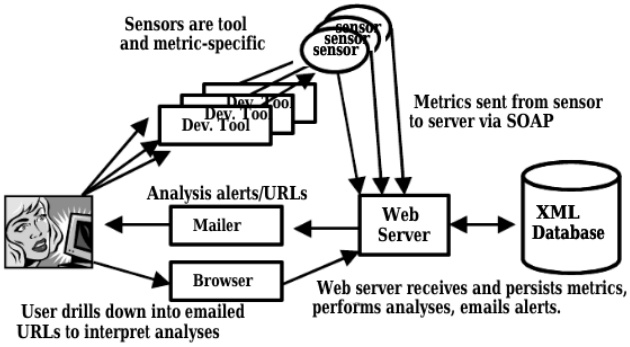

Figure 2: The basic Hackystat architecture and information flow

2. Velocity Measurement: Static Object measures the speed of software development by measuring the amount of the Line Impact per day and for longer periods of time. This allows managers and developers to see how quickly the code is being developed and the production pace of development. Velocity Measurement while better than measuring just the lines of code per day still is not a complete stat in its own right as it needs to be weighed with other factors such as quality of code and other factors that may come into play later.

3. Technical Debt: Static Object implements its own version of trying to measure technical debt by weighing up different statistics and factors.

Services like these can be a useful tool for project manager to easily get data on the productivity of the developers and see where work are going well and where productivity could be improved.

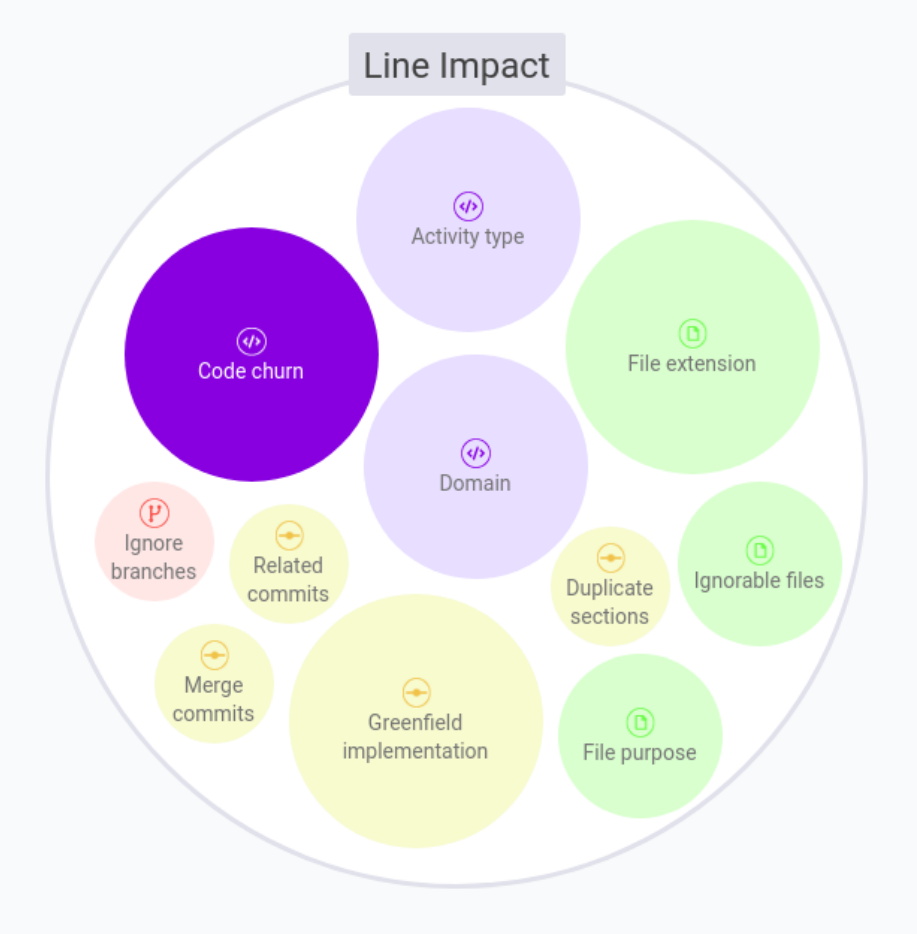

Figure 3: How Static Object determines Line Impact

5 Ethics, Validity and Human Factors

One should probably consider the question, 'if it were possible to accurately measure the productivity of a software engineer or a whole development team, would it be good for the industry as a whole?' Certainly from an ideal perspective where quality code measurement was accurate, it would seem fantastic. Companies and managers could reward and promote those who they can clearly see are the most productive. Individuals could actively see themselves how productive they themselves were and would thrive to improve themselves accordingly. However we must realize ourselves that we do not live in the ideal world and issues and arise accordingly.

Possibly the major problem that comes from using a specific set of metrics to evaluate the value and productivity of a software engineer is that, it will immediately set an incentive for the developer not to write what he/she believes is the best code, but to write code that best improves his/her metrics. Now one could argue that with enough well chosen and balanced metrics one could theoretically very accurately make out the value of the code that is developed.

The issue of over relying on a single metric has occurred in even massive billion dollar tech companies. Steve Ballmer in an interview stated how the practice of counting lines of code as a metric of productivity had become intrinsic to the work culture inside IBM and the company even wanted to pay based on LOC. Ballmer was of course appalled by this and tried to convince the IBM that this was a horrible method of valuing code.

"In IBM there's a religion in software that says you have to count K-LOCs..... And IBM wanted to sort of make it the religion about how we got paid.....How many K-LOCs did you do? And we kept trying to convince them - hey, if we have - a developer's got a good idea and he can get something done in 4K-LOCs instead of 20K-LOCs, should we make less money? Because he's made something smaller and faster, less KLOC. K-LOCs, K-LOCs, that's the methodology.."

The pitfalls into relying too much on metrics that are not suitable for the information one wants to gather is a key problem that one that is careful enough that can usually be avoided but yet companies still fall into these traps.

Like in any large industry the treatment of workers is a major issue in the industry. One must question whether a strict reliance on metrics which reward employees that provide good metrics and punish those with bad metrics. This will almost undoubted add additional pressure and stress on an individual which may not always be beneficial for productivity. Companies probably do not want to create an hostile environment and using metrics to directly compare one employee to another may create an unhealthy competition and rivalries between workers.

One might even suggest the job satisfaction and the good work environment may be a better metric to measure.

With all this new technologies and data the most important metrics today are still the results that are attained from other humans. Results such as the rate of customer satisfaction and peer reviews and still are essentially the bottom line for most businesses therefore outweigh almost all the other metrics discussed in this report.

6 Conclusion

The measurement of software engine and turning the data into metrics has become so important that it has become an industry in itself to create new and better tools and metrics in order to try and measure productivity and the quality of code. There are still major questions on the ethics to address but as the industry builds better and more insightful metrics we seem to heading in a direction and in a future where we can measure the activity of a individual or a team and there value.

References

[1] K. Grove-Rasmussen og Jesper Nygård, *Kvantefænomener i Nanosystemer.* Niels Bohr Institute & Nano-Science Center, Københavns Universitet

https://www.computer.org/csdl/proceedings/gas/2012/1768/00/06225927.pdf

https://people.engr.ncsu.edu/txie/publications/foser10-si.pdf

https://blog.gitprime.com/why-code-churn-matters/

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.138.6806

https://www.researchgate.net/publication/4016775$_{B}eyond_{t}he_{P}ersonal_{S}oftware$ Process$_{M}etrics_{c}ollection_{a}nd_{a}nalysis_{f}or_{t}he_{d}iff erently_{d}isciplined$

https://www.staticobject.com/line$_{i}mpact_{f}actors$

https://stackify.com/measuring-software-development-productivity/

http://www.pbs.org/nerds/part2.html