Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

CS33012 Software Engineering

Measuring Software Engineering

Patrick Dillon Ryan

Student number: 17340382

Table of Contents

Introduction

In this essay I plan on talking about the measuring of software engineering under 4 different headings, they are Measurable data, computational platforms available, algorithmic approach and then finally ethical concerns. Technology plays a major role in the world we live in today especially as software engineers. Software engineering is defined as

"the process of analysing user needs and designing , constructing and testing end user applications that will satisfy these needs through the use of software programming languages." From this definition we can see that in the 21st century software engineering plays a major role in making some of the biggest companies in the world such as Google, Microsoft and apple make a profit.

These companies are some of the most profitable companies in the world with their goal to maximise profits. It's no surprise that they would want to try and measure their workers to make sure they are getting the most from them. But all of this has to come at a cost and have some ethical implications.

Measurable data

Lines of code

One of the most common measurements of how much work is done by one software engineer is the LOC (lines of code)[1] that they write. It's a simple measurement to measure. Basically the more lines of code that they write the better they are. It sounds like the perfect measurement in theory but in reality it has its flaws. For example it's not fair to measure one software engineer who codes in Python against a software engineer who codes in C. If they were both to write the simple hello world program there is a big difference in the LOC

```
1. print("Hello, World!")
```

Printing Hello World in Python

```
1. #include <stdio.h>
2. int main()
3. {
4.    printf("Hello, World!");
5.    return 0;
6. }
```

Printing Hello World in C

It takes 6 LOC to do it in C whereas you can do it in one line in Python. This doesn't mean that the C software engineer is any better than the python one because they are typing more lines of code, this would be an unfair comparison. Although the LOC measurement can be biased towards certain languages it is one of the best measurements from measuring software engineering.

Number of bugs per 1000 lines of code

The industry average of bugs per 1,000 line of code is between 15 and 50 bugs, this figure depends though on when the code is actually typed. The basic human error rate is on average 5%[2]. This figure does depend on many different factors such as the level of testing within the code. This can also be another good level of judging the software engineer. If they are writing code that's riddled with bugs then there will be precious time wasted debugging the code leading to inefficiencies.

Debugging

Debugging is a very hard thing to measure. It depends on many different factors. In order to debug you first need to understand the problem within the code before it can even begin to get solved. Some bugs can take a long time to be resolved whereas some can only take a few second to resolve. This leaves it almost impossible to measure the rate of debugging.

Number of commits

Another way of measuring the software engineering process over time is the number of commits. The number of commits over time should be fairly consistent. It can also be good to gauge how someone works for example do their number of commits in a week deteriorate at the end of the week.

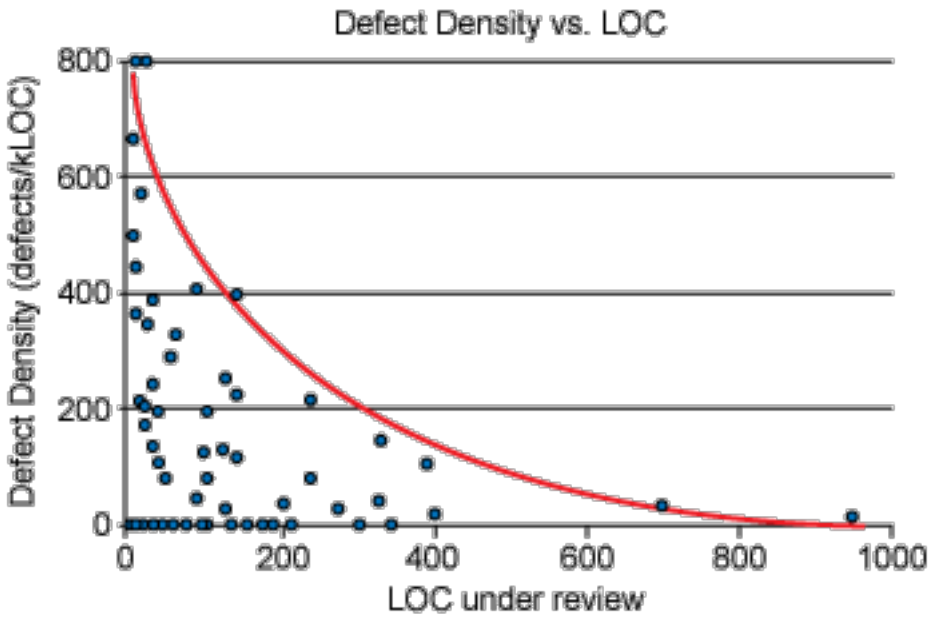
Test Coverage / code coverage

Another method used for measuring software engineering is the code coverage. Code coverage measurement simply determines which statements in a body of code have been executed through a test run. When writing tests for code the ideal code coverage is 100% this means that every line of the code has been tested. From our code coverage we can tell if we have enough testing in place.[3]

Code Reviews

Code review is a software quality assurance activity in which a human will check a program mainly by reading and inspecting the source code. The ideal review is before 400 lines of code have been written since the last code review. There has been a study that showed after 400 LOC the ability to find defects diminishes. In the study it showed that over a 60-90 minute review the yield was on average 70-90% in defect discovery. It is also suggested that you shouldn't spend more than 60 minutes reviewing code. The optimal time and amount seems to be spending 50minutes reviewing 300 LOC.[4]

The code review should also give feedback to the developer about the readability of the code as well as how well commented it is. The more frequent the code reviews, the easier it is to tackle problems such as poor commenting of code.



Number of Defects detected compared to number of LOC reviewed

Computational platforms available

GitPrime

GitPrime is a commercial tool that can be integrated with git to provide visual insights and reports for individual developers as well as for entire projects. It creates visual for the likes of commits, pull requests and work habits.[5] It allows for easy tracking of your teams progress and measuring success. Using the platform you can easily spot bad trends that might be creeping into your team allowing you to easily eradicate those trends. You can also measure cycle time using GitPrime which allows you to measure progress and see if you are progressing. It also tells you exactly who's committing and how much they are committing. All of this data can be used to plan better sprints and reduce cycle time.

GitPrime can also be used to calculate statistics on a certain project such as how responsive the developer is to comments and which parts the developer has been focusing on most, over a specific time frame.

Slack

Slack can be described as one massive chatroom for a whole company. It removes the need for emailing between your colleagues. It allows for communication by channels for groups, it also allows private messaging to share information such as files and images as well as other features. Slack also has many third-party apps integrated into it such as Dropbox, Google Drive and even Github! The integration of these apps means that slack can be seen as a one app does all approach. This enables the ability to manage your workflow through one platform. [6]

Teams

Teams is similar to Slack but it has its differences, It is a cloud bases team collaboration software. It is part of the office 365 package. Teams allows calling, video meetings and file sharing. It doesn't have the same third party app integration that Slack has but it has some of its own such as Exchange, PowerPoint and SharePoint. [7]Businesses can use teams to collaborate on projects in real time across different devices including laptops and mobile devices all around the world once they have an internet connection.

Trello

Trello is a task management app. It gives an excellent visual overview of everything's that been completed, what's currently getting completed and what's left to be completed. It follows the Kanban system which was first developed by Toyota when they were trying to maintain high level of production while maintaining flexibility. Trello can be broken into three main parts, they are boards, lists and cards.

- Board

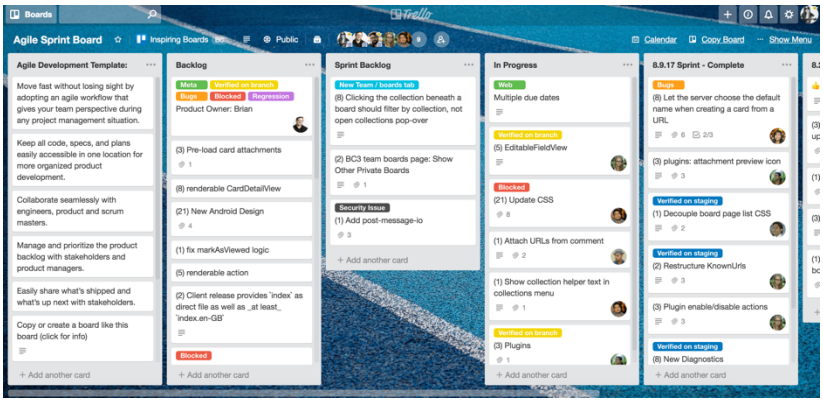
  Boards are usually made for projects or when a product is under development. Each board has members that can view the board and control the flow of cards between lists. They can add themselves or others to cards.

- List

  A list is used to divide a board into different categories. The list is typically broken into 3 categories, that is tasks that are completed, tasks that are currently being completed and tasks that are to be completed. This makes it very easy to visualize everything.

- Card

  A card represents a specific element of a project. Cards get moved through lists as they get completed. A card could be an individual task or it could be an item within a checklist made up of many smaller cards.[8]



What a sample Trello Board looks like

Algorithmic approach

Halstead's Software metrics

Halstead's in 1977 established an empirical science of software development. He noticed that metrics of the software should reflect the implementation or expression of algorithms in different languages but be independent of their execution on a specific platform. The metrics should come purely from the code itself.

By counting the tokens and determining which are operators and which are operands, the following base measures can be collected : [9]

n1 = Number of distinct operators.

n2 = Number of distinct operands.

N1 = Total number of occurrences of operators.

N2 = Total number of occurrences of operands.

n1* = Number of potential operators

n2* = Number of potential operators

The n1* and n2* are the minimum possible number of operators and operands for a module and a program respectively. E.g in C all programs have to contain at least a main() function. So in the case of C n1* = 2 since at least 2 operators must appear for any function. The first is for the name of the function and the second is to serve as an assignment or grouping symbol.

From these above numbers, several measurements can be calculated:

Program vocabulary: n = n1 + n2

This is the total number of unique operators and unique operand occurrences

Program length: N = N1 + N2 This is the overall length of the program

Program Volume: V = Size * ($\log_2$ vocabulary) = N * $\log_2(n)$

This represents the space necessary for storing the program.

Difficulty: D = (n1 / 2) * (N2 / n2)

This is used for measuring how difficult the program is to handle

Effort: E = D * V

The effort measure the level of mental activity needed to translate the existing algorithm into implementation on the specified program language

Time required to program: T = E / 18

Example program

```
Main(){
        Int a, b, c, average;
        scanf("%d %d %d", &a, &b, &c);
        average = (a + b + c) / 3;
        printf("average = %d", average);
}
```

In this program the unique operators are main, (), {}, int, scanf, &, =, +, /, printf, , and ;

The unique operands are a, b, c, average, %d %d %d, 3, average = %d

From this information we can calculate the following:

n = n1(12) + n2(7)

n = 19

N = N1(27) + N2(15)

N = 42

V = N(42) * $\log_2$(n(19))

V = 178.4

D = (n1(12) / 2) * (N2(15) / n2(7))

D = 12.85

E = D(12.85) * V(178.4)

E = 2292.44

T = E(2292.44) / 18

T = 127 seconds

There are 15 counting rules for the C language, the most important ones are :

1. Comments are not considered
2. Function calls are considered as operators
3. All brackets, commas and terminators are considered as operators
4. All variables and constants are considered operands

Halstead's algorithm was quite revolutionary for its time and can be still somewhat applicable to the modern day we live in. It doesn't quite hold for many modern programming languages and tools that are available to developers today though. This is due to the fact that many tools nowadays lead to code been written much quicker. Halstead's metrics can still be used as a useful benchmark though.

Cycle time

Cycle time is defined as “The total time that elapses from the moment when the work is started on a task until its competition”[10]. The difference between lead time and cycle time is that cycle time is from when the tasked is created through till the work is completed whereas the cycle time is from when the work is started until the time the work is completed. The cycle time data can be used for future projects for gauging how long it’s going to take a certain team to complete a required task. It can also be used to gauge progress in seeing progression or seeing if your cycle time is getting shortened. If your cycle time is shortening over time it’s a sign of good efficiency. Measuring your cycle time is a very straight forward process. All you need to do is record when the task was started and when the task was completed.

Waterfall

It is fairly straight forward to implement in a new software project. It is a step by step process. There is six stages to waterfall methodology. They are :

1. Requirements

   During this initial phase, the requirements of the project are noted and taken down. The document that it gets written on is called a requirements document. It defines what the application should do but not how it should do it.

2. Analysis

   During this stage the system gets analysed in order to properly generate the models and business logic that will get used in the application.

3. Design

   This stage covers the technical design requirements. In this stage you choose which programming language to use and technologies that you will use in the process. At the end of this stage a design specification report will be generated in which it explains how exactly the logic will be technically implemented.

4. Coding

   In this stage the source code is written. This will implement all models and logic that were specified in previous stages

5. Testing

   This stage involves QA's(Quality Assurance) and beta testers to test the source code that's been written to try and find any bugs within the code. These people report back with any bugs that are found.

6. Operations

   In the operations stage the application that has been developed is now ready to be released into a live environment. This stage also involves support and maintenance in order to keep the application up to date and free from bugs.[11]

Agile methods

Agile methods are used to find ways to enhance the software development process, both lead times and cycle times are taken into account. The lead time is the time taken by a team to generate ideas, then develop that idea before delivering the software product, whereas the cycle time is the time from developing the software and deploying it into production.[12]

Scrum

Scrum is a framework within which people can address complex adaptive problems while creating and delivering products to a customer at the highest possible value. Scrum involves a team working together. A scrum is the actual time period when the team works together to finish an increment. The team completes a scrum cycle which can range anywhere from one week to a four week cycle. Within that cycle the team will start off with a sprint planning in which they will set our their targets and the time they want to have it completed by. These goals are set out by the scrum master. By the end of the meeting the full team should be clear on what can be delivered in the sprint and how the increment can be delivered. The team usually will have a daily scrum which will involve a short meeting of 15 minutes in which they reflect on the work that's been produced in that day. This helps ensure that each member of the team is on the same page on what needs to be done. The team then has a sprint review at the end of the sprint. In this review they review the work that's been carried out. The last part of the scrum is called sprint retrospective. The goal is that after the sprint cycle the team will either increment(show what's been done in the scrum) or demo the project they have completed to the client. The team will also talk about what worked and didn't work in the scrum. This creates a place where the team can focus on what needs to be improved for the next time.[13]

Ethical concerns

One major ethical concern when it comes to measuring software engineering is the area of data protection. The measurement of such data could be really seen as an invasion of the developers privacy. If a manager or boss wants to measure such data it should be completely clear and transparent about what actual data they are taking and measuring.

Having developers under constant monitoring within a workplace can be a real ethical concern. And can be a major impact on the developers privacy. While some of the metrics that have been talked about above can be excellent for measuring software engineering we need to be careful that we don't over analyse the workers and breach their privacy.
A dictatorship style workplace could happen if the bosses gather metrics from the developers daily life such as who they talk too, how long they spend writing code and even how long they take in the bathroom. This could create a bad atmosphere within a workplace which would lead to a drop in productivity.

We have seen in the last few years with the likes of Facebook, Google, Microsoft that their offices are being advertised as cool friendly places to work in, with many perks such as free food. Although these are very tempting and will entice many software engineers to go work for these big tech companies these bonuses come at a price. The likes of google and Microsoft will offer the likes of free food but they carefully pick the times they will serve it. Breakfast in google could be from 7am – 8am and dinner might not be until 7pm. This is so that the workers won't want to leave the offices and stay there working longer and getting the most of their employees. This can very easily lead to employees having no life at all outside of work due to them being in the office working every hour of the day. This is very unethical as although they may advertise it as the employee getting to choose their working hours they are subconsciously getting tricked into working longer hours.

Conclusion

In conclusion it is clear to see that it is no easy task measuring software engineering. Previous metrics used throughout history have been very good and well worked but many of them are now outdated and a new system is needed for measuring. We must be careful though, that with whatever new metrics we come up with for measuring it that it can't breach the privacy of the software engineer nor can it be unethical.

Bibliography

[1] Diceus. (2019). *7 Metrics To Measure Software Quality in The Most Efficient Way*. [online] Available at: https://diceus.com/top-7-software-quality-metrics-matter/

[2] Quora.com. (2019). *What is the average ratio of bugs to a line of code? - Quora*. [online] Available at: https://www.quora.com/What-is-the-average-ratio-of-bugs-to-a-line-of-code

[3] Confluence.atlassian.com. (2019). *About Code Coverage - Atlassian Documentation*. [online] Available at: https://confluence.atlassian.com/clover/about-code-coverage-71599496.html

[4] smartbear.com. (2019). *Best Practices for Code Review*. [online] Available at: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

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

[6] Getcloudapp.com. (2019). *What Is Slack Messaging | CloudApp*. [online] Available at: https://www.getcloudapp.com/blog/what-is-slack

[7] SearchUnifiedCommunications. (2019). *What is Microsoft Teams? - Definition from WhatIs.com*. [online] Available at: https://searchunifiedcommunications.techtarget.com/definition/Microsoft-Teams

[8] Gray, K. (2019). *How we effectively use Trello for project management - WP Premium Support*. [online] WP Premium Support. Available at: https://wpcurve.com/trello-for-project-management/

[9] GeeksforGeeks. (2019). *Software Engineering | Halstead's Software Metrics - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

[10] Linnanvuo, S. and Linnanvuo, S. (2019). *Agile & Lean Metrics: Cycle Time — Screenful*. [online] Screenful. Available at: https://screenful.com/blog/software-development-metrics-cycle-time

[11] Powell-Morse, A. (2019). *Waterfall Model: What Is It and When Should You Use It?*. [online] Airbrake Blog. Available at: https://airbrake.io/blog/sdlc/waterfall-model

[12] Cgi.com. (2019). *12 Principles of Agile Methodology*. [online] Available at: https://www.cgi.com/us/en-us/life-sciences/blog/12-principles-of-agile-methodologies

[13] Scrum.org. (2019). *What is Scrum?*. [online] Available at: https://www.scrum.org/resources/what-is-scrum