# Measuring Engineering

Trinity College Dublin

Seamus T.J Woods

November 18, 2018

# Contents

## 0.1 Introduction

The purpose of this report is to consider the ways in which the software engineering process can be measured and assesed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approachs available, and the ethics concerns surrounding this kind of analytics.

## 0.2 What is Software Engineering?

*"Software engineering is a detailed study of engineering to the design, development and maintenance of software. Software engineering was introduced to address the issues of low-quality software projects. Problems arise when a software generally exceeds timelines, budgets, and reduced levels of quality. It ensures that the application is built consistently, correctly, on time and on budget and within requirements. The demand of software engineering also emerged to cater to the immense rate of change in user requirements and environment on which application is supposed to be working. "*[6]

## 0.3 Why Measure Software Engineering?

There are several reasons why you want to measure software engineering, and they all boil down to efficiency. Reasons include,

- Identify areas of improvement
- Reduce costs
- Manage workloads
- Reduce overtime
- Increase return on investment[7]

## 0.4 Measurable Data

The current most popular metrics for measuring and assesing software engineering are the following:

- **Lines of Code** This is simply the amount of lines that a software engineer produces.
- **Keystrokes** A single depression of a key on a keyboard.
- **Speed** How quick can a software engineer finish a job?
- **Number of Commits** How often does a software engineer commit?
- **Technical Debt** Technical debt is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer.[3]
- **Code Coverage** The percentage of code which is covered by automated tests.

### 0.4.1 Lines of Code

This was the original standard for measuring software engineering. At the time it may have been a decent method, but in the modern world this no longer holds, there are too many variables. For example if you have someone programming in a very high level language like Python, and someone coding in a very low level language like Assembly, of course the person programming in Assembly will produce more lines of code, but that doesn't mean that they get more work done. Apparently, this method originated back when programming in Assembly was a lot more common[1].

Although this is the case, it does give you a very broad sense on the complexity of a certain project. For example if you have a program which is 1,000 lines of code, and you have another program which is 1,000,000 lines of code, it's fairly obvious that the program with 1,000,000 lines of code is far more complex.

### 0.4.2 Keystrokes

Before I begin, I think it's worth mentioning that I'm assuming these keystrokes are only being tracked within an IDE or similar, and not literally every keystroke from the moment a computer is turned on. Keystrokes, in my opinion, is a bit of a step up from using lines of code as a metric to measure

the productivity of a software engineer, and I think is best explained with an example. If you were to measure software engineering by lines of code, it could be easily abused. You could purposely write more lines of code than you need to. However, if you measured software engineering by keystrokes, you would get a far better idea of the effort that the individual was putting in. They may get assigned a project and quickly program a minimum viable product, which may be far more lines of code than the finished product, and then start to optimise and reduce it from there, which of course will result in a higher keystroke count. Now, if you were measuring by lines of code, they would actually be getting penalised for this, even though they are being more productive than the individual that writes more lines than they need to.

Even though I believe that using Keystrokes as a metric to measure software engineering over lines of code, I don't by any means think it's perfect. For example, an individual that makes a lot of mistakes in their code will also have a lot of keystrokes. As well as this, if an individual chose excessively long variable names or similar, they would also have a much higher keystroke count, and of course that doesn't make them a great software engineer.

### 0.4.3 Speed

In my opinion, the speed in which a software engineer can get a job done is extremely important, but it isn't the be-all and end-all of measuring software engineering. If I gave two people a job and person 1 finished it twice as fast as the person 2, of course I would assume that person 1 is more productive than the person 2, but I really think you would need to look into it further. For example, person 1 may have just completed the bare bones of what I asked to be done, with terrible coding, while the person 2 may have been going a step ahead and written everything with beautiful code, so why should they be penalised for that?

On top of this, if a company revolved solely around how fast work got done, it would create a very stressful environment, and something that I don't think could be maintained for very long. I think the quality of the work would drop dramatically.

### 0.4.4 Number of Commits

I think that this would be a great metric to measure software engineering if it was done right, if it wasn't done right then I think it could be abused. If you were to literally just count every single commmit an individual made, then everyone would just commit every time they wrote a single line of code. However, if you were somehow able to filter out all of the non-meaningful commits, and just count the commits that were meaningful, then I tink this could be very promising. By meaningful I mean, is this commit telling me that a single line of code was added, a new function was added, or an entire class? Perhaps a weight could be attatched to convey the importance of the commits, This way it wouldn't matter how often or rarely somebody commit.

### 0.4.5 Technical Debt

Technical debt should certainly be taken into consideration when measuring software engineering. If a software engineer were to choose a better and more complex approach which would take longer, as opposed to an easier one which could be done in a shorter amount of time ('quick fix'), then they would penalised if any of the other metrics were taken into consideration instead of this one, and obviously that's not right. Tecnical debt can be avoided through the following methods:

- High Test Coverage: If we have high test coverage we ensure that the code is working as intended.
- High Modularity: It is far easier to manage code that is very modular.
- Code Complexity: Writing clean and simple code is easier to come back to and understand if a problem arises.
- Documentation: If you've ever jumped into a large code base with minimal documentation, you'll understand.

If we can somehow measure the amount of technical debt an individual saves us, this would be a very good way of measuring software engineering.[4]

### 0.4.6 Code Coverage

It's very important for software engineers to perform tests on their code. This way they know that their code works as intended. In an ideal world you

would have 100% code coverage with good tests. The idea is that the higher code cover percentage you have, the more reliable that software is, the less likely it is to have bugs.

### 0.4.7 Conclusion

To conclude, I don't believe you can just choose a single one of these metrics and run with it, since every one has pros and cons. I think the most reliable and accurate way to measure software engineering, is to use all of them.

## 0.5 Computational Platforms

There are a number of computational platforms available to measure software engineering, including:

- **GitPrime** GitPrime uses data from GitHub, GitLab, BitBucket—or any Git based code repository—to help engineering leaders move faster, optimize work patterns, and advocate for engineering with concrete data.
- **Waydev** Waydev uses data from GitHub, GitLab & BitBucket to help you improve the developers' work, without their input in a project management tool.
- **Humanyze** An employee ID badge which measures how you interact, your tone of voice, conversations, location in the office.
- **Basecamp** One organised place for projects, teams and company wide communication.

### 0.5.1 GitPrime

Since the vast majority of software engineers are hosting their projects on either GitHub, GitLab or BitBucket, this is a great tool for nearly everyone. The tool collects data for individuals such as the ammount of commits, but then it goes into far more detail with that. It will tell you what percentage of those commits are new work, legacy refactoring, helping others and churning. It will also tell you the risk of those commits (low, moderate, high). It will

also compare what one individual was doing compared to the rest at that time and give you feedback from that. To quote from their website,

*"Jonathan*
*Feedback for this Period:*
*Develop a clear implementation path before starting work.*

*Jonathan spent more time churning code and less on legacy refactoring compared to others in this period.*

*The team's average code churn was 16%. In contrast, Jonathan's churn was 31%."*

I think it's clear from this quote that this information can be extremely useful and has the potential to help the work environment become a lot more efficient, if the advice is heeded. It would also be very useful in finding out who isn't pulling their weight, resulting in you being able to targeting and resolving that problem much faster, as well as finding out who is going above and beyond their job and rewarding them for it.

### 0.5.2 Waydev

There's not too much to say here, Waydev is very similar to GitPrime. They also collect information about all the contributors to a project and tell you what their activity has been like. They don't seem to collect quite as much information as GitPrime, but their price is much lower at $19 per month per active developer versus $749 per month for a team size of 10-25 contributors for GitPrime. It seems like Waydev is for far smaller teams whereas GitPrime may be for larger companies.

### 0.5.3 Humanyze

In contrast to the last two examples this is a physical device that doesn't track what you do in the online world, but in real life. This device will log and upload 4gb of data every day about your location, conversation, etc. On their website, they state *"Data Privacy as a first principle."*, however in my opinion, it is still a little bit over the line. I can undestand from a data perspective, wanting to know how many of your employees are in on an average day, and where they may spend the majority of their time, but to

record their tone of voice and the conversations they're having is a bit too far, in my opinion. I can't imagine that this would make anyone in the work place feel very comfortable and may actually throw them off from getting their job done.

### 0.5.4 Basecamp

Basecamp looks like an amazing web application to keep on top of everything. You can create a board for a project you're working on, and on that board you'll find different sections. These include,

- Campfire: A groupchat for everyone involved in the project.
- Message Board: Used to post updates and gather feedback on work being done.
- To-dos: Ability to assign work and see what's done and not done.
- Schedule: Shows important dates and deadlines.
- Docz & Files: Place for images, assets and files to be easily accessible.

In the home screen there is an area that everyone in the company can see called the HQ. There is an area here called *'Automatic Check-Ins'*, here a question is asked every day such as *'What did you do today?'* and your response is posted on the board. This way everyone in the company knows what everyone else is doing. This tool is useful for management as they can see exactly what an employee has to do for the day and can see what they've gotten through. A review from a user of Basecamp shows me that it really is a good product.

*"I am using Basecamp from last two year. I love it because of its Team Chat, the average price which is very affordable, Message Boards/News which have so many options to edit the text, such as bolding, coloring, bullet points, etc which I personally didn't see in any other project management tool."*

## 0.6 Algorithmic Approaches

I believe there are many algorithmic approaches to measure software engineering, but I think they would mainly be split into two groups:

- **Algorithms to retrieve the data.** For example, algorithms to calculate the percentage of commits that are new work, legacy refactoring, helping others and churn.
- **Algorithms to understand the data.** For example, using machine learning to try to understand the information and potentially return some useful information.

### 0.6.1 Algorithms to Retrieve the Data

Since the algorithm for calculating churn is very simple to explain we will start with that. Churn is defined as the lines added, modified or deleted to a file from one version to another, the alogrithm to retrieve churn is very simple, so long as git has been used to the file you are calculating churn on. All the algorithm would need to do is add up all the new, modified and deleted lines from one version (or commit) and see what the difference of that is from the other version. High code churn can indicate bad coding practice, which is information an analytical tool will take from this and maybe give you feedback or advice on it.

### 0.6.2 Algorithms to Understand the Data

I believe the most common way for an anlytical tool to understand the data would be through machine learning. Machine learning is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.[8] This would be exaclty how GitPrime for example would analyze all of the data they have, and give people solid recommendations that will work for them. I'm not sure if this is implemented yet, but would definitely be possible for them to also use machine learning to make predictions.

## 0.7 Ethics

In my opinion, I don't see anything wrong with the majority of this kind of analytics, but of course there are some exceptions, for example, Humanyze recording conversations. I think as long as an individual gives consent to collect data on them, and they know exactly what is going to happen to that data and what it will be used for, I don't see a problem. If the opposite was true, there was no consent or the employee didn't know where the data was going or what was happening to the data, then of course I completely disagree with it.

Regardless of how private or not the collected data of employees is, it should be stored very securely, and then it would have to be considered who has access to this information.

## 0.8 Conclusion

To conclude, there is no perfect way to do anything. In my opinion a combination of all these metrics and platforms to analyse it may be good, but I think if you wanted excellent data, you would have to pick a combination of metrics based on the individual, and I think thats a long way off. I'm sure there's a good in between for everyone. In terms of computational platforms to analyse the data, I think GitPrime in conjunction with Basecamp would be really good. With GitPrime you could see all the statistics and some feedback on how to improve, and Basecamp would keep you up to date and engaged. This way I don't think there would be any information collected that may be too intrusive to the employee, since it would only be your activity on the project.