Measuring Software Engineering

A Report by Conrad Oppermann

Abstract

This will discuss the various ways in which the software engineering process can be measured and assessed. I will be analysing this through the following topics: measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available and the ethical concerns surrounding this kind of analytics.

To complete this report I consulted a variety of academic and corporate material written on the subject of Software Engineering as well as the content discussed by Dr Stephen Barrett in SC3012.

Introduction

To fully come to grips with how to measure and assess software engineering, we should first define what we mean by it. In 1968, the first NATO Software Engineering Conference took place. This was the first time the term Software Engineering was first. The conference came about because before 1968 there had been no frame or discipline prescribed to the process of creating software. The industry experts present sought to rectify this. They came up with the novel idea that software should be approached in the same way as physical engineering. A systematic approach to the design, development, testing, and maintenance of software was needed and thus software engineering was born. (Rouse, 2016)

Since that 1968 conference Software Engineering has made tremendous leaps forward and has helped to shape almost all aspects of the world we live in. Software defines how many of us interface with society. Out of such rapid growth, pertinent questions arose regarding how we measure Software Engineering such as; how can we judge the productivity of a software engineer? How can we analyse the value of the end product? What is good quality software?

In this report I will analyse those questions and more under four main headings:

1. Measurable Data
2. Platforms Available
3. Algorithmic Approaches
4. Ethics of Analysis

Software Engineering Processes

We must cover current software engineering processes in order to see how it can be analysed. A software engineering process is, at its core, a set of steps that result in the development of software. Four general activities are part of these processes:

1. Software specification
2. Software design & implementation
3. Software verification & validation
4. Software maintenance

Each general activity can be broken down in smaller actions that take place, such as unit testing. How each of these activities is implemented depends on what process the software engineering team wish to use, examples of processes include waterfall and agile. The waterfall process was one of the original processes conceived in the early 70s. It is a plan-driven process where teams are given a clear brief at the beginning of the project and work to deliver on that brief following a sequential order of steps. It is not adaptable to change and most of the major decisions are made before any code is written.
(Elgabry, 2017)

Agile, in comparison, is a very flexible process that takes aspects of iterative and incremental methods. This approach has a set of core principles that highlight a people not process ideology. Agile has a lot of customer collaboration and works of an incremental delivery model where change is embraced and simplicity is maintained.
(Barrett, 2020)

The two most common forms of the Agile approach are Scrum and Extreme Programming (EP). In Scrum, programmers carry out a list of prioritised tasks of the course of a sprint during a set time frame. Regular, informal, meetings take place over the course of the sprint and the process breaks a project down into a set of iterative steps. EP has similar guiding principles: simplicity, communication, feedback, respect and courage. EP aims to be adaptable to changes requested by the client. Similar to Scrum, EP is broken down into small iterative steps. (Barrett, 2020)

In a recent study, Agile processes were found to be used in 97% of companies surveyed to develop software. Scrum was found to be the most popular Agile method. So it is clear that Agile is the most popular process in modern software engineering. This is because of the need for constant updates and maintenance as customer feedback is constantly coming in. As we have found Agile is by far the most common process I will base my analysis mainly on how companies assess and measure Agile software engineering methods. (StateOfAgile 2020)

Measurable Data

Before we analyse data it is important to know what type of data can be collected. That knowledge will show us the possible scope of our analysis. It is also important to ask why we are measuring certain data points and the metrics used to weight them.

Lines of Code

In the early days of software development, the number of lines of code developers wrote was used to value their work. But with a little thought, this is clearly a flawed way to understand the value of developers. It would be easy for a programmer to write a superfluous amount of code but that is not what should be rewarded - efficient, easy to understand code should be rewarded. This is what was argued by Bhatt, Tarey and Patel in their paper on measuring lines of code as a metric for value.

They outlined 9 flaws in using the number of lines of code to value programmers. I will give a summary of the flaws I find most pertinent:

DEVELOPER'S EXPERIENCE

As you do something longer you become better at it, you learn how to be more efficient. Skilled developers will be able to achieve the same results with far fewer lines of code than that of an unskilled developer. Thus logic should be assessed rather than the quantity of code.

DIFFERENT CODING LANGUAGES

Different coding languages use a varying amount of code to achieve the same goal. The code may do the same thing but some languages will achieve it in far fewer lines than others. For example, our lecturer says he can write 10x fewer lines of Haskell than java to achieve the same result.

LACK OF STANDARDS

There is no widely accepted standard for what a line of code is. Some lines could be hundreds of characters long while others just a few. What do comments count as - they don't affect the logic but will help others to understand. Some organisations like the SEI

have tried to standardise what a line of code is but it is proving difficult as a result of the constantly evolving nature of code and the introduction of new languages.

The paper also highlights the fact writing code is only ~35% of a programmers job. They have to be creative and come up with efficient code and much of this is done without actually writing code. A lot, of the cleverness, occurs before code is written. Also, in agile, code gets replaced all the time. What if your code is modified so much it is not attributed to you - that can't be fair.

Number of Commits

A commit is when a developer adds to or modifies an existing codebase. This can be as simple as adding punctuation or as large as adding whole classes of code. In the case of the Agile process, developers may be committing code multiple times a day.

I believe the number of commits should only be used to demonstrate the activity by developers. I don't think size or frequency should be analysed to the same extent. How a programmer's commits may be a matter of personal preference so I don't think this metric should be used to compare two programmers. It would be flawed to do so.

Lead Time

The amount of time it takes to reach a programmers goal is important. In agile programming, there is usually a very short, almost daily, lead-time. In that developers may be given a requirement in the morning and expected to have it completed by the end of the business day.

The amount of time required to write code is a very important measurement for management. It allows them to budget properly and gives them a good indication of how long certain types of projects take.

The process of measuring lead time is simple as all it required is a stopwatch or taking note of the time - in Agile the Scrum Master will take care of this measurement.

Lead time measurement can be used on the macro and micro scale. It can show how quick entire firms can create the software as well as individual programmers. It can show the efficiency and the efficacy of teams and individuals.
(Screenful 2015)

Code Coverage

Code coverage measures the amount of code that is tested by unit tests. This metric can be used to denote how well a piece of code has been tested for bugs, it is a form of quality assurance.

The level of code coverage will infer how much care a programmer has taken to ensure their code functions properly. The only problem is it can be inefficient to test for all eventualities but this is a decision for management. A programmer could spend weeks testing code for every single case but this would be a waste of time. A balance must be reached between testing all cases and efficient use of time.

Code Churn

This metric works by measuring the amount of times code has been changed or deleted by a programmer. This metric tends to be used as an *indicator* to management that something is not going as planned. Spikes in code churn usually are as a result of programmers trying to fix code.

Management should investigate the cause of spikes and try to make them less frequently to maintain efficiency.
(Pluralsight 2020)

Code Complexity

One way to measure how complex the code is to measure *cyclomatic complexity*. One method of doing this is to calculate the number of linearly independent "paths" through the source code.

CYC = Number of Edges - Number of Nodes + 2(Number of disconnected parts of the flow graph)

In the formula above the higher the CYC result, the more complex the code is. Complex code has a higher likelihood of bugs and defects which means developers should aim to make code as simple as possible. Complex code is also more difficult to test, read and maintain.
(Britton, 2016)

Computational Platforms

Personal Software Process (PSP)

The PSP platform was developed by the Software Engineering Institute in 1995. The process has been attributed to American software engineer Watts Humphrey who wrote a book outlining PSP. The technique helps software engineers to improve and better understand their performance by the difference between their actual and predicted code development. Watts stated his belief that PSP would allow an engineer to make their coding more efficient by streamlining their process.

The PSP employs forms to collect information from software engineers; this includes a project plan summary, a defect-recording log, a design checklist and a code checklist etc. As PSP is focused on manual input, the engineer will personally fill out the forms, the process is prone to human error. This lead to data quality issues so a similar, but refined, system, the LEAP toolkit, was developed in response.
(Johnson, 2013)

The Leap Toolkit

The University of Hawaii developed the LEAP toolkit to address the data quality issues of the PSP. This was achieved by automating and normalising the data analysis process. The technique still required human input but the analysis was automated. The LEAP process also went further in its analysis an example of this is the addition of regression analysis.

After its introduction, it reached widespread use in the industry. But questions were soon asked about how viable it would be in the future. As there would never be anyway to fully automate the process as it would always be based on manual input into forms. This lead to the development of automated tools such as Hackystat.
(Johnson, 2013)

Hackystat

Hackystat works by attaching sensors to tools used by developers, it is easily integrated into tools such as GitHub or Bitbucket. Data is collected “unobtrusively” so the developer does not know when it will be taken. Hackystat provides a visual representation of the data which allows the manager to have a greater understanding of the work being completed by their employees.

Hackystat collects similar data to PSP such as the time spent on the project, how many commits are made, the scale of the project, code coverage and code complexity. Many

developers didn't like how their work was being monitored as they felt the process lacked transparency.
(Johnson, 2013)

Code Climate

Code Climate is a software product that was built to analyse the work of software engineers. "Code Climate incorporates fully-configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous, and ubiquitous", according to their website. (Code Climate, 2020)

Code Climate analyses code to catch errors and flag potential security issues. It can collect, analyse and present various quantitative and qualitative metrics that summarise the code. This gives managers greater insight into what their programmers are working and the quality of their work. It also ensures that the quality remains at a set level by preventing poor quality code. (Code Climate, 2020)

GitHub and others

GitHub is the prominent web-based-version-control platform. It has 40 million active users. It allows managers to see an engineer's code and the quality of their commits. Through an API it allows a whole host of other platforms to monitor and analyse code written on GitHub. Examples of this include Pluralsight - Flow which is used by Google, AWS and Microsoft. Other examples include CAST, Codacy, Sonas, DevCreek, Codebeat.

All of these platforms have a similar goal; to give managers insight into the code their team is producing.

Algorithmic Approaches

As the nature of Computational Platforms has evolved away from manual towards automated so have the algorithmic approaches. This can be seen in the rise of Artificial Intelligence and in particular, machine learning. Computers can now analyse situations and make decisions on how best to approach them, this is known as machine learning. I think machine learning is the most pertinent algorithmic approach for this paper, so I will explore the types of machine learning techniques that help developers automate analysis of the software engineering process.

As an MSISS student, I will outline a few techniques I have learned over the past three years of learning statistics and probability.

Machine learning algorithms have two main types: supervised and unsupervised.

Supervised Machine Learning

Supervised Machine Learning is when the output of a given input is known. Similar to a function, y=f(x), where x is the input and y is the output. The assumption behind supervised learning is that the output, in this case, y, can be so effectively deduced through our knowledge of how we reach the output that we can predict future input and output values.

This is achieved through a process of trail and error. Users must monitor the machine learning algorithm so that it is constantly predicting the correct values. A user would tweak the algorithm if they notice values straying away from expected.
(Brownlee, 2016)

LINEAR DISCRIMINANT ANALYSIS

LDA is a technique that classifies multivariate data depending on the nature of each data point. This is a widely used technique in machine learning and AI as it is simple to realise. This is because the process involves definite "bounds" to which data is characterised relative to the bound.
(White, 2020)

K-NEAREST NEIGHBOURS

K-Nearest Neighbours works on a non-parametric classification basis. It classifies new data by analysing its proximity to existing data points. "K" is the number of close points we use to make our determination of what group the new data is part of. If K=3 and 2 of the nearest points are from the same group, the new data will be characterised as part of that group. The challenging part is that K will have to be recalibrated for each data set.
(White, 2020)

Unsupervised Machine Learning

The opposite to supervised, unsupervised machine learning is a process where the input value has no definite output value. This means that we rely a lot more heavily on how we model the data. How we choose to model the data will have a large effect on the analysis and the results. As there are no "set" outputs, the process cannot be monitored and is, therefore, a lot more difficult to see whether it is working correctly or not. This

means it takes a lot more experience and expertise for this process to be effective, as the structure output will vary depending on what model we use.
(Brownlee, 2016)

K-MEANS CLUSTERING

K-means clustering is a similar method to the nearest neighbours algorithm except that it functions off a clustering and re-centring method. Similar again, we must pick a number of clusters we would like to analyse, K. Then each cluster's centre point is computed and each data point is reassigned a cluster based on which centroid it is closest to.

The central point for each cluster is recalculated after each reassignment session and the process is repeated until there are no improvements possible.
(White, 2020)

Conclusion

All of these methods are statistical methods I have learned in MSISS and are not specific to computer science. They don't even need a computer to compute them as they are merely computation algorithms. But they do clearly apply to machine learning as they are tools programmers can use to allow computers to make decisions. With all of the above examples, the data is "hard", meaning there is little nuance to it. So it is most effective with numerical data and will be less effective with softer data which in many ways is more human data. As humans are less binary creatures and nuance is an inherent part of humanity, techniques will have to be innovated to suit humans. That being said machine learning is a growing area and many people can interact with a computer without being able to distinguish whether it is human or not.

Ethics

The ethical implications of collecting another person's data are complex and decisive. I will give a brief overview of how the industry can and should use data to improve the lives of their stockholders and stakeholders.

LEGISLATION

The EU released the General Data Protection Regulation (GDPR) In May 2018, to replace the previous Data Protection Act (DPA). This made it a legal requirement to inform people what data of theirs is being stored, how it is being stored and how it will be used. This was an overhaul of the previous standards that were lax and opaque. Companies must now seek consent to hold someone's data and they must also have proof of that consent.

The law applied to any firm that stores European customers' data. This would include most software companies, Google, Facebook, Amazon and Apple being the biggest amongst them. If firms are found in breach of the regulations they can be fined 4% of their revenue or €20m whichever is *higher*. This has implications for any firm looking to measure their software engineering process as they have to do it in a way that complies with the GDPR.
(Citizens Information, 2020)

DATA COLLECTION

How data is collected is a prime ethical concern. Most concerns surround the fact that data can be collected unobtrusively, without the knowledge of developers. This was something I discusses earlier in the report when I described developers' issues with Hackystat. Developers don't like to feel like they are being watched. They didn't like how big an insight into how they work that it gives management.

Hawthorne said, the act of observing a process inherently changes it, this could not be more pertinent in the case of observing software engineers. Employees should know about how their data is being collected and this will reduce the distrust. It is important for employers to strictly follow GDPR.

The type of data employers are collecting is equally important to study. Firms may have access to a variety of an employee's data. Not all of it will relate to their job. Firms only have the right to collect data pertaining to their employment. This can be a difficult line to draw.

An example of this could be whether a firm should collect the fitness data of employees using Fitbits or Apple watches. This would give them an insight into the general level of fitness in the firm and could affect how much the firm's health insurance premium is.

I would argue a firm cannot collect an employees health data as it does not have a direct link to the work they are employed to do. But this is a contentious issue in the United States where some teachers are being forced into wearing a Fitbit and do a certain amount of exercise to receive health benefits.

I think that it is important employers are only collecting data which will help improve business decisions, and in particular improve the software engineering process.
(Johnson, 2013)

DATA USAGE

How a firm uses data is just as important as how they collect it. Data must be used in such a way as to motivate workers rather than impinge their work. They should see it as a support rather than a hindrance.

A firm must use data impartially and fairly. Data should not be used to discriminate between gender-identity, ethnic background or sexuality. Data should also not be used to infer aspects of an employee's life that do not apply directly to the work they do.

Software engineering is not just a logical process, it is a creative process too. Creativity must not be stifled. Engineers should not be motivated to write as many lines as possible, this will lead to waste and inefficiency, it will also not get the most from your developers. Creativity takes time.

Data should only be used to improve the software engineering process at both a team and individual level. Data usage should absolutely not negatively impact on the work of developers and engineers.

Conclusion

This report has discussed the way in which software engineering processes are measured. I have analysed measurable data, platforms available, algorithmic approaches and the ethics surrounding this type of analysis.

As we have discussed, the process of measurement has evolved from a very human based perception approach to heavily automated data and machine-driven. We should expect great evolution in this space as big tech innovate to make this process more effective.

The main focus of this innovation should be to not create conflict between engineers and management and to ensure that the analysis helps their employees. This needs to be achieved through transparency and consultation with developers to ensure its success.

Bibliography

A. Sillitti, A. J. (2003). Collecting, integrating and analyzing software metrics and personal software process data. Proceedings of the 29th Euromicro Conference: IEEE.

Brownlee, J. (2016, March 16). Supervised and Unsupervised Machine Learning Algorithms. Retrieved from Machine Learning Mystery: https://machinelearningmastery.com/supervised-and- unsupervised-machine-learning-algorithms/. Accessed 12/11/2020.

Citizens Information. (2018, September 5). Overview of the General Data Protection Regulation (GDPR). Retrieved from Citizens Information:https://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_of_general_data_protection_regulation.html Accessed 13/11/20

Code Climate. (2020). About Us: Code Climate. Retrieved from Code Climate: https://codeclimate.com/about/ Accessed 09/11/2020

Elgabry, O. E. (2017, March 17). Software Engineering — Software Process and Software Process Models. Retrieved from OmarElGabry's Blog: https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc Accessed 07/11/2020.

White, A. (2020, November). STU33011: Lecture Notes. Retrieved from Blackboard. Accessed 12/11/2020.

Johnson, P. M. (2013). Searching under the Streetlight for Useful Software Analytics. Hawaii: IEEE Software.

Kan, S. (2003). Metrics and Models in Software Quality Engineering. Addison-Wesley Professional.

Littlefield, A. (2016, Septemeber 12). The Beginner's Guide To Scrum And Agile Project Management. Retrieved from Trello: https://blog.trello.com/beginners-guide-scrum-and-agile- project-management. Accessed 12/11/2018.

Barrett, E. (2020, November) CSU33012: Lecture Notes Retrieved from Blackboard. Accessed 05/11/2020.

Rouse, M. (2016, November). Software Engineering Definition. Retrieved from WhatIs: https://whatis.techtarget.com/definition/software-

engineering#:~:text=Software%20engineering%20is%20the%20application,and%20management%20of%20software%20systems.
Accessed 10/11/2020.

VersionOne. (2020. 14th Annual State of Agile Report.

Bhatt, Kaushal & Tarey, Vinit & Patel, Pushpraj. (2012). Analysis Of Source Lines Of Code(SLOC) Metric. IJETAE. 2.

Pluralsight (November 13, 2019). INTRODUCTION TO CODE CHURN. https://www.pluralsight.com/blog/tutorials/code-churn Accessed 11/11/20

Screenful (October 28, 2015). Cycle Time. https://screenful.com/blog/software-development-metrics-cycle-time Accessed 11/11/20

Britton, J. (October 22, 2016). What is Cyclomatic Complexity? https://www.perforce.com/blog/qac/what-cyclomatic-complexity Accessed Accessed 11/11/20