Report on the Measurement of Software Engineering

Rachel Brady, 18322843

Introduction

The term “Software Engineering”, was first coined by renowned mathematician and computer science pioneer, Margret Hamilton, while she was working on the Apollo missions. The term gained a more general acceptance due to the role of the NATO Software Engineering conferences. The conferences were held in 1968 and 1969 and were attended by international experts on computer software, who had a common objective of defining best practices for software, grounded in the application of engineering.
The IEEE Standard Glossary of Software Engineering Terminology, defines Software Engineering as, "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software".
To gain the most out of any software engineering project, team members can adopt the use of software engineering metrics in order to increase productivity, identify areas of improvement, manage workloads and reduce cost.
This report will identify the software engineering processes that can be measured and assessed in terms of measurable data, give an overview of the computational platforms available to perform this work and the algorithmic approaches available. To finish I will discuss the ethical concerns surrounding this kind of analytics.

Software Engineering Metrics

Lines of Code

The Number of Lines of Code (NLOC) is a very common metric used to measure the source code program length. This is made fairly easily when a decision is made determining what qualifies as a line of code. For example, Conte has provided the following definition:

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line This specifically includes all lines containing program headers, declarations and executable and non-executable statements ".

We can then use the abbreviation NCLOC to refer to non-commented lines of code and CLOC to refer to commented lines of code. Using these we can define total length as:

$$\mathbf{LOC = NCLOC + CLOC}$$

We now have effectively 3 different metrics that provide an understanding of the quality of code written.
This metric has proven to be important when measuring productivity and obtaining cost/effort estimates as it is very easy to compute at any stage of the development process, and is intuitive and easy to visualise. However, even when we consider CLOC, its usefulness is somewhat exhausted when we take factors such as effort, functionality, complexity, redundancy and reuse into account. We cannot compare code written in different languages using this metric, after all LOC in an assembly language is not comparable in effort, functionality, or complexity to a LOC in a high-level language. As well as this, the most accurate LOC measurement is obtained upon completion of the project, there is no correlation between the LOC written and its optimisation and finally, dependent on the skill level or preference of the programmer, they may write programs more efficiently and using less LOC than another programmer, all while the functionality of both programmes are identical. It is clear that the problems of this metric greatly outweigh the good, and that managers should look to other metrics, to gain estimates and measure the productivity of their team.

Function points

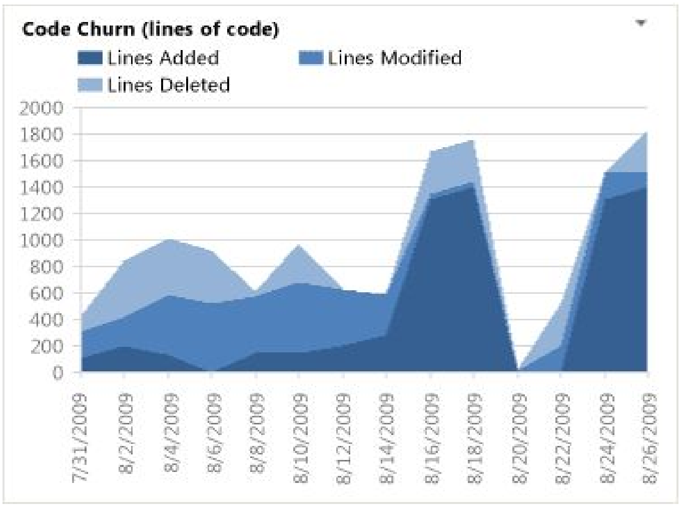
Function point analysis has become increasingly more popular since its development in 1979. It helps developers and users to quantify the size and complexity of software application functions.
By A.J. Albrechet’s definition, function points are calculated by counting the following number of :

1. External Inputs (EI)
2. External Outputs (EO)
3. External Queries (EQ)
4. Internal Log Files (ILF)
5. External Log Files (ELF)

Function points have been proven to be consistent, repeatable, technologically independent and help normalize data, enable comparisons and set project scope and client expectations.
In the algorithm section of the report, I will develop this point and demonstrate how these are used to calculate a reasonably good estimate of the required amount of development resources.

Code Churn

Code churn represents the number of LOC that have been modified, added or deleted, typically over a short period of time. Managers use this metric as an indicator to when something is wrong with the development process. When churn begins to spike, it may indicate that a developer is experiencing difficulty in solving a particular problem, or is a perfectionist and is repeatedly making final touches to a feature that is ready for release.



Example code churn report, screenshot via Visual Studio

Agile metrics

The nature of agile software development (ASD) is very different from the traditional software development methods. It requires little documentation, and can handle user requirement changes at all stages of the software development process. The unpredictability of these change requests therefore make metrics that give cost and time based estimates extremely important. The following metrics can be employed in ASD projects:

Velocity

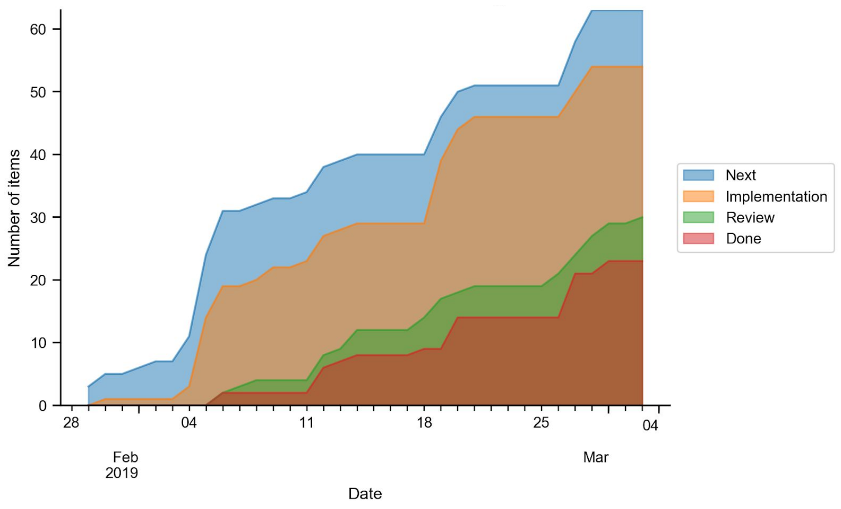
Velocity is similar to productivity. It is defined as the amount of completed user stories (requirements) in each iteration, and can be used to estimate the remaining time until completion of the project. This only works well for teams that consist of the same individuals working full-time. Therefore this metric is invaluable if more members are added or if any members leave the team. Managers should also note that velocity is just an empirical observation and does not guarantee future results.

Burndown Chart

This tool is used for planning and monitoring progress in agile methods. The chart commonly has two different plotted graphs: iteration burndown, which estimates the remainder of tasks to be completed in the iteration and release burndown, which illustrates the current releases. Analysis of this chart helps in the decision making process by comparing the estimated work and the remaining work. User stories can be added or dropped based on whether the team is ahead of or behind schedule.

Cumulative flow diagram

A Cumulative flow diagram (CFD) will often be used instead of a burndown chart as it is more specific. It demonstrates a quantity of work in a given stage.



Example Cumulative Flow Diagram

In the above diagram, the quantity of work in each iteration is illustrated in one or more states (Next, Implementation, Review, Done).
It is useful for many reasons including progress control by illustrating work in progress (WIP). WIP can then estimate the delivery date, the lead time and identify any problems before they become crucial. Another advantage of using CFDs is continuous improvement can be achieved, by monitoring and eliminating bottlenecks as they arise. The total size of the backlog can also be closely monitored which is useful for the team to see if the backlog is increasing/decreasing.

Re-work hours

ASD at its core is about embracing change and being flexible to accommodate the client. Therefore many teams will use this as an indicator of the ability of the team to deliver product quality. Re-work will most commonly be defined as the amount of hours spent fixing flaws and defects. If this measurement is used in each iteration, managers can ensure that it does not get out of control. Displaying this data on a re-work graph can be useful for discovering bottlenecks.

Earned Business Value

Earned Business Value (EBV) is used in agile to track the “valuable” part of “valuable software”. It also measures from a business perspective, how much of a product is complete. It is most commonly used to track financial value by allocating an estimated ROI to each User Story - its weight. Rawdthorne defined a formula for estimating EBV of each story as the sum of weight for stories done. However this metric is only useful when the scope of the work is clear up-front.

Computational Platforms

Sonarqube

Sonarqube is an open-source platform that allows for continuous inspection of source code quality and security. It supports code written in 27 different languages and can be plugged in directly with existing tools such as Bitbucket and Github to notify the user of issues, directly in Pull Requests. There are three types of issues that a programmer will be notified of when running an analysis through this tool. A bug is a coding error that will cause the code to break. Vulnerability is a point in the code that is open to attack. Code smell is a maintainability issue that makes the code confusing and difficult to maintain. The use of this tool among programmers will ultimately lead to an increase in their velocity. Managers can also use it to identify the programmer’s strengths and weaknesses.

GitPrime

GitPrime is a very useful management tool for software engineering projects. It gives insights into project progress, productivity, workflow and much more by computing and analysing specific metrics. The majority of software development companies use Git-based repositories such as GitHub, BitBucket and GitLab. GitPrime is developed to analyse data from these Git-based repositories.
GitPrime can provide insights into individual engineers, as well as the team as a whole. A leaderboard can be produced to give an overall view of everyone’s individual contributions.
A daily snapshot provides a deeper insight into individuals’ performance. It reports on daily code commits, new work vs. code churn, legacy refactoring and work done helping others. This report has a lot of potential for managers to identify weaknesses within the team and find opportunities for growth.

The Work Log module collectively shows information regarding team performance, the contribution of members and work habits via graphs. Software engineers whose teams have made use of this tool have reported that they find it motivating to also be able to view their individual development and progress reports.

Jira

Jira, a software tool for agile development, was developed by Atlassian in 2002. It is used by organisations who want to track issues and bugs and integrate project management functions into their agile projects. The functionality of this tool can be broken down into four key elements:

1. Plan
2. Track
3. Release
4. Report

Planning enables the creation of user stories, the planning of sprints and the distribution of tasks across the team. Typically managers will use a Kanban board to display this information. Typically a kanban board will display the information under four headings - Backlog, WIP, Testing, Finished. At this stage, all user stories will be in the backlog or in WIP.

Tracking allows managers to gain constant feedback regarding the actual progress. Burndown charts, sprint reports, cumulative flow diagrams, scrum boards etc. can be created with the touch of a button to communicate this progress. Jira allows for seamless integration with other tools such as BitBucket and Confluence and so, these interactive dashboards and graphs will be automatically updated, as the team members progress.

The accuracy of the previous two tasks, allows managers to deliver the incremental work on completion of the iteration, to the customer, with confidence that the information is up-to-date.

Jira will generate reports to help improve team performance based on real-time, visual data that can easily be communicated to the team.

These features and the analysis of the generated metrics, help with the primary goal of agile development projects - to focus on delivering incremental and iterative value, as quickly as possible.

Code Climate

Code Climate is a software tool which offers many features that allow for the automatic collection and analysis of data, focusing particularly on the analysis of code, ensuring optimal code quality. It is a web-hosted software which can integrate into platforms such as GitHub using the browser extension. The amount of technical debt and test code information will be displayed on GitHub.

Its primary feature is its ability to check every line of code independently and give feedback on what the code can do, all the processes involved in it and overall test coverage. This is useful not only for developers in the immediate, but also for managers who want to track these metrics to gain an overall understanding of their team's progress. Code Climate will collect this data and provide a summary of the most important changes that were merged, as well as visualizing long-term trends in the codebase.
Managers can pay particular attention to these retrospectives in order to gain insights into code quality, and for comparison against other projects, to discover where time and effort needs to be allocated most.

Algorithms

Function Points

First the Unadjusted Function Point Count (UFC) is computed. This is done by rating the transactions (EI, EO and EQ) based on the number of files updated/referenced (FTR) and the number of user-recognizable fields (DET).

FTRs	DETs		
	1-5	6-15	>15
0-1	Low	Low	Average
2-3	Low	Average	High
>3	Average	High	High

The files (ILF and ELF) are rated based on DET and the number of user-recognizable data elements in an ILF/ELF.

RETs	DETs		
	1-5	6-15	>15
1	Low	Low	Average
2-5	Low	Average	High
>5	Average	High	High

Both transactions and files are given a low, average or high rating which in turn is converted to UFCs (a digit).

Rating	Values

	EO	EQ	EI	ILF	ELF
Low	4	3	3	7	5
Average	5	4	4	10	7
High	6	5	6	15	10

The value adjustment factor (VAF) is computed based on 14 general system characteristics (GSC). Each GSC is given a weighting on a scale of 0-5 based on the strength of its influence.
Finally the final function point count (FPC) can be calculated:

$$\mathbf{FPC = UFC * (0.65 + (sum\ (GSC) * 0.01)\ )}$$

This final function point figure can give a reasonably accurate estimate as to the amount of development resources required to carry out the project.

Halstead's Algorithms

According to Halstead, "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands." His algorithms therefore depend on the count of these tokens, classified into the following metrics:

n1= Number of distinct operators
n2= Number of distinct operands
N1= Total number of occurrences of operators
N2= Total number of occurrences of operands

Where **N = N1+N2** is the total number of operator and operand occurrences
And **n = n1 + n2** is the total number of unique operator and operand occurrences

Length N^ of a program can be calculated :

$$\mathbf{N\hat{} = n1log2n1 + n2log2n2}$$

Program volume V (the size in bits, of space necessary for storing the program) can be calculated:

$$\mathbf{V = N * log2\ (n)}$$

Program difficulty D can be calculated:

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

These algorithms have proven to be popular as they are simple to calculate. However, they are dependent on the complete code and so may not prove useful to managers who are looking to track the progress of their development team, over a period of time.

Productivity

Card's simple model of productivity (2006) :

Physical Productivity = Number of LOC / Man hours or days or months
Functional Productivity = Number of Function Points / Man hours or days or months
Economic Productivity = Value / Cost where, Value = f (Price, Time, Quality, Functionality)

This model considers the entities such as product, processes, sub-processes, requirements, value, cost and effort. Measuring the three will give teams a better overview of their productivity and where they should focus their efforts more towards.

Nogueria, Luqi, Berzins and Nada's Productive Ratio α (2000) :

α= % of Direct Development time / % of idle time

This ratio would be helpful for managers to identify whether motivation techniques are required to maximise the direct development time and minimise the idle time. However, in practice, it can be difficult to calculate idle time, without the programmers feeling like they are being constantly monitored.

Ethical concerns

In 2006, a survey was carried out by McKinsey Global on social technologies and their use in the workplace. One question that was posed to company leaders was to do with the most significant risks or concerns that they associated with the use of social technologies in their organizations. 55% of respondents noted that their major concern was that the use significantly increases risk of confidential information being leaked.
Therefore as the demand for these technologies increases, so does the demand for security and confidentiality within these platforms. The companies that provide the resources should advertise their security promises at the forefront of their

campaigns, as it is evident from the survey that security is a feature that companies should not, and will not forego. For example, Code Climate works with security firms, like NCCGrop, to “perform regular penetration testing and audits of Code Climate and its infrastructure.” They also ensure that employee access to customer data is extremely limited, and when offering support, explicit consent will be asked for at all times.

We can not discuss security and privacy of data without mentioning the General Data Protection Regulation (GDPR). This was introduced in 2016 across all EU member states which standardizes data protection laws. Until then despite the growing presence of the digital world, there was no single unifying act that would regulate data protection issues. GDPR gives EU residents the right to know what is going to be done with their data. As well as this, they can restrict or object to data processing and they have the right to obtain copies of their data that is being processed. In the world of software engineering, to avoid complications and breaches, project managers, along with everyone else involved in data processing must strictly adhere to these laws. Therefore, careful consideration must be taken when choosing what data to be collected, how to legitimately process, store and access it. This is with regard to their customers as well as their team members. Only the relevant information that will be used exclusively for project development purposes should be collected.

Issues arise when platforms, tasked with collecting data, are not measuring soley the work being done by the team. GitHub can only monitor code written/pull requests/merges, etc. There should be no ethical concerns relating to these, as it is simply analysing the members’ work.
Hitachi, the Japanese engineering and electronics company is one of few companies whose employees are using wearable technology, namely the Hitachi Business Microscope. This device looks like an ID-card that is worn around the neck in order to collect data. It contains a microphone, an accelerometer, infrared sensors and other data-collection tools and is alerted and begins data collection when 2 or more come within a specified distance of each other. This constant level of surveillance in the workplace will undoubtedly sound unnerving to many people. Employees may not feel comfortable having private conversations with each other if they know it will be recorded. Even worse, employees who wish to discuss dissatisfaction regarding their workplace, will unlikely do so, under this level of surveillance. These kinds of privacy concerns will impact staff morale and drive employee-management relations down.

Conclusion

In this report, I have discussed the multitude of metrics that can be gathered to analyse the software engineering processes. I explored a few of the many platforms and tools that can do this gathering and analysis of data, leading to reduced effort and time spent by management. I analysed a couple of algorithms that are available to compute useful metrics. However, it is clear that these metrics and tools should not be used without careful consideration of the ethical concerns involving personal data collection and processing.

Bibliography

Introduction:

- https://en.wikipedia.org/wiki/NATO_Software_Engineering_Conferences
- https://en.wikipedia.org/wiki/Software_engineering

Metrics - L.O.C.:

- https://www.cs.purdue.edu/homes/cs307/slides/lecture14_2up.pdf
- http://doras.dcu.ie/18544/1/Paul_Doyle.pdf

Metrics - Function Points & Code Churn:

- https://ieeexplore.ieee.org/document/582971
- https://eecs.qmul.ac.uk/~norman/papers/jss_1999.pdf
- https://www.ukessays.com/essays/information-technology/importance-of-software-measurement-and-metrics-information-technology-essay.php

Metrics - Agile:

- https://www.researchgate.net/publication/235009023_On_the_Current_Measurement_Practices_in_Agile_Software_Development

Platforms - Sonarqube:

- https://www.sonarqube.org/

Platforms - GitPrime:

- https://www.softwareadvice.com/ie/app-development/gitprime-profile/
- https://www.pluralsight.com/blog/platform/how-i-use-gitprime-metrics-with-my-small-team

Platforms - Jira:

- https://www.atlassian.com/software/jira
- https://kanbanize.com/kanban-resources/getting-started/what-is-kanban-board

Platforms - Code Climate:

- https://www.blissfully.com/code-climate/

Algorithms - Function Points:

- https://www.tutorialspoint.com/software_quality_management/software_quality_management_albrechts_function_point_method.htm

Algorithms - Halstead:

- https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

Algorithms - Productivity:

- https://www.researchgate.net/publication/237050541_Measuring_productivity_of_software_development_teams

Ethical concerns:

- http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf
- https://www.zdnet.com/article/wearables-in-business-five-companies-getting-real-work-done/
- https://mobidev.biz/blog/gdpr-compliant-software-development-guide