Measuring Software Engineering

Introduction

Software engineering refers to the systematic application of engineering approaches to the development of software. The term was coined in the 1960s by Margaret Hamilton during her time working for NASA on the Apollo program. Since then, software engineering has become ever more important in the modern world, with the IT and software industries accounting for an increasingly large part of most first world economies.

Unlike most other industries, measuring production in software engineering is not a simple or straightforward task; due to the nature of the craft it is difficult to quantify the magnitude of what has been done. Despite this, however, many efforts have been made to do so, as having an idea of the volume of work done or needing to be done can be invaluable not only to software companies, but also their clients and employees. The resulting measurements may not only indicate the programming productivity of programmers and development teams, but also the maintainability of software once it has been produced. I aim to outline the various ways in which people have tried to measure software engineering and discuss the strengths and weaknesses of each.

Measurable Data

Source lines of code (SLOC)

The most obvious way to measure the 'amount' of software engineering done is to simply count the amount of text written by the software engineers. This is generally done by counting the lines of code written rather than the character or word counts. When the lines of code of a program or programs' source code are used as a metric, this is referred to as 'Source lines of code' (SLOC). SLOC is generally used to approximate the effort required to develop a program but may also be a good metric of how difficult the program will be to maintain once it has been developed.

However, there are obvious problems with this approach; not every line of code is significant, for example a lone opening or closing bracket. It also fails to account for the time and difficulty involved in writing certain sections of code compared to others. Some code may be easy to implement because it is simple in nature or is quite common in programming, while other code may require more specialist knowledge and hours bug fixing to implement successfully. Programming is more akin to problem solving than a typing test.

Also, if SLOC is used as a metric for employee performance, then this encourages more verbose code or even inefficiency in many cases. For example, a programmer may be tempted to use a 'while' loop and declare and increment a counter variable on separate

lines where a ‘for’ loop would be appropriate, or even duplicate large sections of code as opposed to writing it in a function, which would cause issues for the program’s maintainability.

SLOC where the lines in the text of a program’s sources code are counted but comment lines are excluded is referred to as *Physical SLOC.* However, this another type of SLOC called *Logical SLOC* that was developed to address many of the problems with Physical SLOC I outlined above. Logical SLOC is a measurement of the number of executable statements in a program. This has many advantages over physical SLOC as it accounts for differences in formatting and style between programmers. For example, elements such as lone brackets are not counted, and if one were to write a simple for loop with both the condition and body on the same line, this would count as one Physical SLOC but two Logical SLOC.

Logical SLOC is still far from a perfect measurement, however. It still fails to account for duplicate code and for the difficulty involved in writing, bug testing, and maintaining some blocks of statements compared to others. It also has some disadvantages over Physical SLOC in that it is more difficult to automate.

Overall, it is clear that SLOC fails to be a perfect metric of software engineering. Measuring software engineering by its most quantifiable aspect, as is done in many other industries, will never capture the full nuance of the craft. As Bill Gates said, “Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” That being said, there is still merit to its simplicity and objectivity that more advanced metrics are unlikely to match.

Halstead complexity measures

Halstead complexity measures are one of the earliest methods of measuring software size and were devised in 1975 by Maurice Halstead of Purdue University. He said that the syntactic components or tokens of a computer program fell into two distinct categories, operators and operands. An operator is a token such as a plus, minus or equals or a function call. Any other token is an operand. To calculate Halstead complexity measures, one must first count the number of distinct operators (n_1), the number of distinct operands (n_2), the total number of operators(N_1), and the total number of operands(N_2). Then the measures may be calculated:

Program vocabulary: $\mathbf{n = n_1 + n_2}$

Program length: $\mathbf{N = N_1 + N_2}$

Calculated estimated program length: $\mathbf{\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2}$

Volume: $\mathbf{V = N \times \log_2 n}$

Difficulty: $\mathbf{D = (n_1 / 2)(N_2 / n_2)}$

Effort: $\mathbf{E = D \times V}$

Number of delivered bugs: $\mathbf{B = E^{2/3} / 3000}$

The difficulty refers to how difficult a program is to both write and understand, while the effort can be translated into time spent coding in seconds (T) using the formula:

T = E / 18

Halstead's measures have advantages over SLOC in that they provide more detailed information, such as estimated difficulty and effort, and can measure the overall quality of the program and predict the rate of error. However, it has its disadvantages as well: it requires the complete source code of the program to be effective, it is not useful for incomplete code or mere sections of completed code and because of this, it has no use as a predictive estimating model. It is also true that unlike SLOC, many of the measures, such as Difficulty and Effort, mean little in a vacuum as they are measures proposed by Halstead, and are only really meaningful in comparison to the same measures taken on another program.

ABC Software Metric

In June of 1997, a programmer named Jerry Fitzpatrick published an article in C++ Report proposing an alternative software size metric to SLOC. Named *ABC Software Metric*, it was intended to overcome the shortcomings of both Physical and Logical SLOC as a measurement of software size.

The software size is first calculated by counting the number of assignments (A), branches (B), and conditions (C) in a section of code. A scalar ABC size value or "aggregate magnitude" can then be calculated:

$$|ABC| = \text{sqrt}((A*A) + (B*B) + (C*C))$$

The A, B and C values also provide useful information on their own. B, the count of branches in the program, is virtually identical to *cyclomatic complexity,* which is a common metric of complexity in a program. It is a measure of the number of linearly independent paths through a given section of code.

A big advantage the ABC Software Metric has over SLOC is that it ignores differences in formatting and is unaffected by duplicate code. Another advantage is that unlike Halstead's measures, it does not require the complete source code of the program to be effective and can be applied to individual methods, functions, classes, modules or files within the program and still produce useful information. It is seemingly a best of both worlds solution to measuring software.

A disadvantage of the ABC metric, however, is that its counting rules must be tweaked to fit the syntax of the programming language. This is why in the original article Fitzpatrick outlined three distinct set of rules for C, C++, and Java. It is also generally unsuitable for non-imperative languages such as Prolog.

Platforms

Many platforms are available today that may help with the measurement of software engineering.

GitHub

GitHub has existed since February 2008 and is a website that provides hosting for software development and version control using Git, the open-source version control system that was developed by Linus Torvalds. Users upload code to a repository that can be committed to by multiple collaborators. In the last ten years the platform has become extremely popular and is now used by over 50 million developers. GitHub is generally used by software companies to host private repositories and by users who wish to host open source projects.

The GitHub Insights tab provides numerous tools that can be used to measure software engineering. There is a contributors tab that displays every contributor to that repository and attempts to order them by contribution. It displays their number of commits, as well as the number of lines of code they have added and removed, along with a graph that shows their commits over time. The pulse tab provides more detailed information, and over a shorter period of time – between 24 hours and one month. It lists the total number of commits, lines added, and lines deleted, but also the number of pull requests merged and how many remain open. It also details the number of issues that have been closed and how many new issues have been raised within the set period of time. Overall, Github Insights can provide a basic overview of how a project is being developed but is not very meaningful without a greater understanding of the project.

Jira

Jira is a tool for agile development developed by Atlassian and released in 2002 that allows for bug tracking and agile development. It provides numerous tools for planning, tracking and releasing software, but its main use is for issue tracking.

Jira is designed to support both Agile methodologies, Scrum and Kanban, and allows for both Scrum and Kanban boards. Scrum and Kanban refer to two different strategies of implementing agile development. A Scrum board allows the team to plan their work in greater detail before they begin than a Kanban board. Users can create sprints and give story points to user stories in order to plan which story can go to which sprint. A list of items called the backlog is created, versions and sprints can be created, and issues can be moved from backlogs to sprints. A Kanban board is different in that it is less plan focused. The status of tasks can be tracked but these cannot be organised into sprints. Unlike Scrum boards, Kanban boards will work well without having to give your user stories time estimates. Both of these boards have their uses for planning and organising software development and can provide useful for information on how a project is progressing.

Jira also offers a Roadmap feature which displays a broader overview of the project plan. It can help multiple teams collaborate together, identify dependencies across large pieces of work, and plan for team capacity.

Jira's most relevant feature to the topic of software engineering is its Agile reporting feature. This generates a variety of charts and reports to display information in a way that is easy to understand. For example, the Sprint report displays the work completed or pushed back on each sprint, which can help determine if the team is overcommitting or if scope creep is becoming an issue.

Overall, Jira's tools provide a greater insight into the development of a piece of software than GitHub Insights, but not all the information it provides can be considered an objective metric than can be easily understood at a glance or or be used to compare to pieces of software. However, the agile reporting feature addresses these shortcomings somewhat with its well-illustrated graphs.

Ethical Concerns

One major consideration that must be considered when discussing the topic of measuring software engineering is the ethics involved. I feel that when it comes to collecting data relating to employees, a middle ground must be found between gathering and analysing data that will be of benefit to the company and respecting the employees' privacy. Of course, laws and regulations such as the European Union's General Data Protection Regulation (GDPR) aim to protect employees' personal information, but it is still possible for an employer to legally collect information in a way that may be considered invasive. In general, however, I think most employees would not deem measurements, such as SLOC or Halstead measures that are taken to assess productivity, invasive. In the case of information recorded by software development platforms such as GitHub or Jira, it is very transparent what data is being collected when a developer makes a commit, and indeed the developer is very much in control of the data being collected; for that reason I would not deem such data collection invasive or unethical either. I do believe however, that employees should be made aware of exactly what data is being collected on them and what it is used for. For example, if the data is being sold to a third-party organisation as part of a study, this should be made clear to all affected employees.

Sources

https://en.wikipedia.org/wiki/Source_lines_of_code
https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm
https://wiki.c2.com/?AbcMetric
https://www.atlassian.com/software/jira/features