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

Software Engineering metric validity and ethics of analytics

An insight of Software Analysis

Microsoft Office User

Many techniques have originated to quantify the process of Software engineering and software products. A few of many measurements are seen below provided by wikipedia:

- Balanced Scoreboard
- Bugs per line of code
- Code coverage
- Cohesion
- Comment density
- Connancent software development
- Constructive Cost Model
- Coupling
- Cyclomatic complexity (McCabe's complexity)
- DSQI (design structure quality index)
- Function Points and Automated Function Points, an Object Management Group standard
- Halstead Complexity
- Instruction path length
- Maintainability index
- Number of lines of code
- Program execution time
- Program load time
- Program size (binary)
- Weighted Micro Function Points
- CISQ automated quality characteristics measures

The most frequently used metrics in the industry are SLOC (Source Lines Of code, Halstead Compleixity, Cyclomatic Complexity, Cohesion, Coupling and Oman's Maintainability Index. These are commonly used as all of them together can give a sense of the other not so commonly used metrics. Additionally, these main metrics

give a sense of all the other metrics as they intersect with each other. Further, these metrics are used as they quality metrics consisting of quantifiable metrics.

Strengths, weakness and what other metrics do they consider

SLOC:

SLOC, an abbreviation for 'Source Lines Of Code' is a measurement of effort put into the coding of the software. SLOC gives an accurate measure of programming effort put into the development of the software although in total is considered one of the weakest of all code-based metrics. For examples the Linux kernel in 2008 had 15 million lines of code. This gives a software analyst an understanding of the coding effort put into the software. There are similar measurements such as number of files and classes that give an idea of the size of the software. Although The most accurate sizing measure of software is "Program binary size" which also gives a sense of loading time. All these measures measure quite accurately how much programming effort a SE put into the programming of his software, they do not consider the effort spent designing or redesigning the approach, function points or utility of his software. SLOC alone cannot give us an idea of software effectiveness. It needs to be combined with other software metrics to truly get an insight into the software's capability.

Function points McCabe's Cyclomatic Complexity:

McCabe's CC is a software metric that quantifies the complexity of a program. It is computed by measuring the number of linear paths through a program. McCabe's Complexity increases as the number of paths increase. The complexity is defined as

$M = E – N + 2P$

Where:

E = Number of edges of a graph

N = Number of Nodes of a graph

P = Number of Connected Components

Cyclomatic Complexity can be calculated with respect to functions, modules, methods or classes within a program. A qualitative description of the quantitative results of CC are as follows:

Complexity Number	Meaning
1-10	Structured and well written code High Testability Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

Linux sums to a whopping 798,092 total MCC with 161k of function. 54k function scored 1, 144k scored 10, 764 are in the scary over-50 category and with 125 falling into the "untestable" over-100 category. Linux kernel has a 0.88 correlation for MCC with LOC (Lines of code).

The qualitative description of these quantitative analysis does not give the program full justice if it is well written code with appropriate error checks as Linux kernel is. These descriptions are based on average programs and are not program specific. A less experienced developer's program will require more tests than a more experienced one. Another defect of McCabe's CC as a measure of complexity is that it fails to mention recursion or case statements. This is why during the development of Linux kernel the MCC requirements were waived if the inclusion of such statements were used.

Derivation of Cyclomatic complexity have been developed such as:

- Actual Complexity Metric (*ac*)
- Module Design Complexity Metric (*iv*(*G*))
- Extended Cyclomatic Complexity[V(g')]
- Essential Complexity Metric (*ev*(*G*)
- Essential Complexity Metric (*ev*(*G*))
- Pathological Complexity Metric (*pv*(*G*))
- Design Complexity Metric (*S0*)
- Integration Complexity Metric (*S1*)
- Object Integration Complexity Metric (*OS1*)
- Global Data Complexity Metric (*gdv*(*G*))

These used when appropriate is a more accurate way of computing a fitting metric that describes complexity.

Halstead Maurice Complexity:
Halstead's observation that software metrics need reflect the effectiveness of algorithms in different languages whilst being independent of platform specific execution. Maurice wanted to find empirical measurable properties of software and the relationships between them. Consequently, the metrics he found whilst doing.

He research has found where:

$n1 = the\ number\ of\ distinct\ operator$
$n2 = the\ number\ of\ distinct\ opereands$
$N1 = the\ total\ number\ of\ operators$
$N2 = the\ total\ number\ of\ operands$

$Program\ vocabulary: n = n1\ +\ n2$

$Program\ length: N = N1 + N2$

$Calculated\ program\ length: N = n1 \log_2 n1 + n2 \log_2 n2$

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

$Difficulty: D = \frac{n1}{2} \times \frac{N2}{n2}$ (difficulty to write or understand)

$Effort: E = D \times V$

$Time\ Required\ to\ Program: T = \frac{\mathrm{E}}{18}\ seconds$

$Number\ of\ delivered\ bugs: B = \frac{E^{\frac{2}{3}}}{3000}$ $or\ most\ recently\ B = \frac{V}{3000}$

it is simple to compute and inquisitively correct or reasonable. Whilst being intuitive it identifies many of McCabe's CC shortcomings by accounting for the length and data complexity of the program. Many Studies such as have empirically validated it such as Oma's works. The intuition and validation encourages software project managers to use it as a tracking and efficiency index.

With these advantages, Halstead's metrics have drawn much criticism As the intuition of his calculation do not accurately represent the empirical measurements developers require to measure software development effectiveness. Critics also say that Halstead's metrics are far outdated as it was used to validate programs averaging 800 Lines which do not represent the large-scale projects of today.

Additionally, Many of Halstead's metrics are based on Effort which is an arbitrary unit of measure. It is very much like saying "how long is a piece of string." Effort is a subjective term and will vary from human to human.

Oman's maintainability index:

Using a combination of these metrics Oman Proposed an index know as MI (Maintainability index), calculated as a polynomial using the aforementioned metrics (HV, MCC, LOC/SLOC) to quantify the maintainability of programs.

The Maintainability Index is computed as follows:

$$MI = 171 - 5.2\ln(\overline{HV}) - 0.23\overline{MCC} - 16.2\ln(\overline{LOC}) + 50\sin(\sqrt{2.46\,perCM})$$

where:

$\overline{HV}$	is the average Halstead Volume per module for the program,
$\overline{MCC}$	is the average McCabe's Cyclomatic Complexity value,
$\overline{LOC}$	is the average number of lines of code per module for the program, and
$perCM$	is the percentage of lines containing comments in the program.

from this it is possible to deduce that Oman's MI is determined by (size, complexity, and comprehension effort).

MI model gives a correlation coefficient of 0.94. he validated both HV's and MCC's metrics around 20 years later whilst conducting his research. Oman also accounted for the usability of 2 varieties of the CC which ever one may be more appropriate.

Off course as Oman uses MCC, HV and LOC it falls under the same criticism of these metrics. And the five metrics are to a limited degree interrelated which enables the correlation we observe to be indirect. Though, these metric combined can compensate for the shortfalls identified in each independently and are used in an interdependent way in the MI calculation to give a more precise metric.

Cohesion and Coupling:

Common coupling is usually inversely proportional to cohesion. Common coupling is known as the gold standard and can is used to validate the above metrics. "degree of

interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between module"
Usually a "low coupling" indicated that a computer system is well structured. High coupled systems are usually exhibit:

1. system wide vulnerability and instability when a change is made in one module
2. module assembly or deletion require more time due to dependency on other modules
3. harder specific module testing as other modules need to be used

Tools Available:

There are many language specific CASE tools one of the main ones being DMS. These have available metric tools for C#, COBOL, Java and VBScript. The cloneDR tool can also be used to customize different metric tools for different languages. These CASE tools were used to gather the above metrics and more on the Linux kernel. These tools automate the task of software development analysis. and are essential for large scale projects today.

Ethics of software development measuring:

Software development metrics have advanced significantly over the years although the ways of acquiring the information for this data has even gone further. The above metrics are primarily dealing with the software itself and analysing its effectiveness. Today this is becoming easier and easier to analyse and the analysers want to make the metrics more empirical. To do this corporations have started to track the effort, time management and effectiveness of their employees down to the seconds, SLOC written by a developer, function points created etc. This raises many ethical concerns on whether corporations should be able to collect this data. Whether it is a breach of privacy.

Bibliography

https://www.guru99.com/cyclomatic-complexity.html

https://en.wikipedia.org/wiki/Halstead_complexity_measures

https://etd.library.vanderbilt.edu/available/etd-07082008-001537/unrestricted/Larry_Thomas-Dissertation-Final_Version-2008-07-07.pdf

Halstead, Maurice H. (1977). *Elements of Software Science*. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

https://en.wikipedia.org/wiki/Coupling_(computer_programming)

https://www.guru99.com/cyclomatic-complexity.html

https://www.semanticdesigns.com/Products/Metrics/index.html