Complexity Metrics and Models

Halstead's Software Science

The Software Science [Halstead 77] developed by M.H.Halstead principally attempts to estimate the programming effort.

The measurable and countable properties are :

- n_1 = number of unique or distinct operators appearing in that implementation
- n_2 = number of unique or distinct operands appearing in that implementation
- N_1 = total usage of all of the operators appearing in that implementation
- N_2 = total usage of all of the operands appearing in that implementation

From these metrics Halstead defines :

i. the vocabulary n as $n = n_1 + n_2$
ii. the implementation length N as $N = N_1 + N_2$

Operators can be "+" and "*" but also an index "[...]" or a statement separation "..;..". The number of operands consists of the numbers of literal expressions, constants and variables.

Length Equation

It may be necessary to know about the relationship between length N and vocabulary n. **Length Equation** is as follows. " ' " on N means it is calculated rather than counted :

$$N' = n_1 \mathit{log}_2 n_1 + n_2 \mathit{log}_2 n_2$$

It is experimentally observed that N ' gives a rather close agreement to program length.

Quantification of Intelligence Content

The same algorithm needs more consideration in a low level programming language. It is easier to program in Pascal rather than in assembly. The intellegence Content determines how much is said in a program.

In order to find Quantification of Intelligence Content we need some other metrics and formulas :

Program Volume : This metric is for the size of any implementation of any algorithm.

$$V = N\log_2 n$$

Program Level : It is the relationship between *Program Volume* and *Potential Volume*. Only the most clear algorithm can have a level of unity.

$$L = V^* / V$$

Program Level Equation : is an approximation of the equation of the *Program Level*. It is used when the value of *Potential Volume* is not known because it is possible to measure it from an implementation directly.

$$L ' = n^*_1 n_2 / n_1 N_2$$

Intelligence Content

$$I = L ' \times V = (2n_2 / n_1 N_2) \times (N_1 + N_2)\log_2(n_1 + n_2)$$

In this equation all terms on the right-hand side are directly measurable from any expression of an algorithm. The intelligence content is correlated highly with the potential volume. Consequently, because potential volume is independent of the language, the intelligence content should also be independent.

Programming Effort

The programming effort is restricted to the mental activity required to convert an existing algorithm to an actual implementation in a programming language.
In order to find *Programming effort* we need some metrics and formulas :

Potential Volume : is a metric for denoting the corresponding parameters in an algorithm's shortest possible form. Neither operators nor operands can require repetition.

$$V ' = (n^*_1 + n^*_2) \log_2 (n^*_1 + n^*_2)$$

Effort Equation
The total number of elementary mental discriminations is :

$$E = V / L = V^2 / V '$$

If we express it : The implementation of any algorithm consists of N *selections* (nonrandom > of a vocabulary n. a program is generated by making as many mental comparisons as the program volume equation determines, because the program volume V is a measure of it.

Another aspect that influences the effort equation is the program difficulty. Each mental comparison consists of a number of elementary mental discriminations. This number is a measure for the program difficulty.

Time Equation
A concept concerning the processing rate of the human brain, developed by the psychologist John Stroud, can be used. Stroud defined a moment as the time required by the human brain to perform the most elementary discrimination. The Stroud number S is then Stroud's moments per second with 5 <= S <= 20. Thus we can derive the time equation where, except for the Stroud number S, all of the parameters on the right are directly measurable :

$$T' = (n_1 N_2 (n_1 \mathit{log}_2 n_1 + n_2 \mathit{log}_2 n_2) \mathit{log}_2 n) / 2 n_2 S$$

Advantages of Halstead :

i. Do not require in-depth analysis of programming structure.
ii. Predicts rate of error.
iii. Predicts maintenance effort.
iv. Useful in scheduling and reporting projects.
v. Measure overall quality of programs.
vi. Simple to calculate.
vii. Can be used for any programming language.
viii. Numerous industry studies support the use of Halstead in predicting programming effort and mean number of programming bugs.

Drawbacks of Halstead :

i. It depends on completed code.
ii. It has little or no use as a predictive estimating model. But McCabe's model is more suited to application at the design level.

McCabe's Cyclomatic number

A measure of the complexity of a program was developed by [McCabe 1976]. He developed a system which he called the cyclomatic complexity of a program. This system measures the number of independent paths in a program, thereby placing a numerical value on the complexity. In practice it is a count of the number of test conditions in a program.

The cyclomatic complexity (CC) of a graph (G) may be computed according to the following formula:

$$CC(G) = \text{Number (edges)} - \text{Number (nodes)} + 1$$

The results of multiple experiments (G.A. Miller) suggest that modules approach zero defects when McCabe's Cyclomatic Complexity is within 7 ± 2.

A study of PASCAL and FORTRAN programs (Lind and Vairavan 1989) found that a Cyclomatic Complexity between 10 and 15 minimized the number of module changes.

Thomas McCabe who is the invitor of cyclomatic complexity has founded a metrics company, McCabe & Associates There are some other metrics that are inspritted from cyclomatic complexity. You can find them at McCabe metrics

Advantages of McCabe Cyclomatic Complexity :

i. It can be used as a ease of maintenance metric.
ii. Used as a quality metric, gives relative complexity of various designs.
iii. It can be computed early in life cycle than of Halstead's metrics.
iv. Measures the minimum effort and best areas of concentration for testing.
v. It guides the testing process by limiting the program logic during development.
vi. Is easy to apply.

Drawbacks of McCabe Cyclomatic Complexity :

i. The cyclomatic complexity is a measure of the program's control complexity and not the data complexity
ii. the same weight is placed on nested and non-nested loops. However, deeply nested conditional structures are harder to understand than non-nested structures.
iii. It may give a misleading figure with regard to a lot of simple comparisons and decision structures. Whereas the fan-in fan-out method would probably be more applicable as it can track the data flow

Fan-In Fan-Out Complexity - Henry's and Kafura's

Henry and Kafura (1981) [from Sommerville 1992] identified a form of the fan in - fan out complexity which maintains a count of the number of data flows from a component plus the number of global data structures that the program updates. The data flow count includes updated procedure parameters and procedures called from within a module.

$$\text{Complexity} = \text{Length} \times (\text{Fan-in} \times \text{Fan-out})^2$$

Length is any measure of length such as lines of code or alternatively McCabe's cyclomatic complexity is sometimes substituted.

Henry and Kafura validated their me tric using the UNIX system and suggested that the measured complexity of a component allowed potenetially faulty system components to be identified. They found that high values of this metric were often measured in components where there had historically been a high number of problems.

Advantages of Henry's and Kafura's Metic

i. it takes into account data-driven programs
ii. it can be derived prior to coding, during the design stage

Drawbacks of Henry's and Kafura's Metic

i. it can give complexity values of zero if a procedure has no external interactions

Sencer's Home Software Measurement Page

sencer@hun.edu.tr

This page is maintained by Ümit Karakaþ and Sencer Sultanoðlu

Last modified Oct 12, 98

References

- Albrecth(1979) and Gaffney : Software function, source lines of code, and development effort prediction: a software science validation, IEEE Trans. on Softw.Eng., 9(6),pp639-648, 1983.
- Banker(1994) Kauffman, Wright, Zweig, Automating Output Size and Reuse Metrics in a Repository-Based Computer Aided Software Engineering (CASE) Environment IEEE Trans. Software Eng, 20(3), pp 169-186,1994.
- Boehm(1981) : Software Engineering Economics, Prentice Hall, 1981
- Chidamber and Kemerer(1994) : A Metrics Suite for Object Oriented Design, IEEE Trans. Software Eng., vol 20, no 6, June, pp. 476-493, 1994.
- Jeffery(1993) DR, Low GC and Barnes M, A comparison of function point counting techniques, IEEE Trans. Software Eng, 19(5), pp 529-532, 1993.
- Kemerer(1993) : Empirical studies of assumptions that underlie software cost estimation. Information and Softw. Technol., 34(4), 211-18, 1992.
- McCabe(1976) : Complexity Measure, IEEE Transacions on Software Engineering, Volume 2, No 4, pp 308-320, December 1976
- Morris(1989) : Metrics for Object-Oriented Software Development Environments. Master's Thesis, M. I. T. Sloan School of Management, 1989
- Halstead(1977) : Elements of Software Science, New York, Elsevier North-Holland, 1977
- Humprey(1996) : Estimating With Objects, Part 1 Part 2 Part 3 Part 4 Part 5 Part 6 Part 7 Part 8 Part 9 Part 10 Part 11
- Putman(1978) : A general empirical solution to the macro software sizing and estimating problem. IEEE Trans. on Softw. Eng., Volume 4, No 4, pp 345-61, April 1978.
- Sommerville(1992) : Software Engineering Addison Wesley Publishing Company, Workingham, England, 1992.

Sencer's Home Software Measurement Page

sencer@hun.edu.tr

This page is maintained by Ümit Karakaþ and Sencer Sultanoðlu

Last modified 24 Jul, 98