Halstead Complexity

Yegor Bugayenko

Lecture #4 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

“Any attempt to find a universal set of metrics that could be applied to any computer program, might at first glance appear destined to be unfruitful, if not merely difficult. But, without universal, measurable parameters, we would be in the position of trying to develop the science of thermodynamics before the advent of a temperature scale.”

— Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., 1977. doi:10.5555/540137

Inputs

- η_1 — the number of distinct operators
- η_2 — the number of distinct operands
- N_1 — the total number of operators
- N_2 — the total number of operands

Example from Wikipedia

Example [edit]

Consider the following C program:

```
main()
{
  int a, b, c, avg;
  scanf("%d %d %d", &a, &b, &c);
  avg = (a+b+c)/3;
  printf("avg = %d", avg);
}
```

The distinct operators (η_1) are: `main`, `()`, `{}`, `int`, `scanf`, `&`, `=`, `+`, `/`, `printf`, `,`, `;`

The distinct operands (η_2) are: `a`, `b`, `c`, `avg`, `"%d %d %d"`, `3`, `"avg = %d"`

Operators and Operands

"When a program is translated from one language to another, as from FORTRAN to machine language for example, the actual operators and operands may indeed change, but both versions must still consist of combinations of operators and operands. No other category of entities need be present."

Source: Maurice H. Halstead. Advances in Software Science. *Advances in Computers*, 18(1):119–172, 1979. doi:10.1016/S0065-2458(08)60583-5

Length and Vocabulary

- $N_1 + N_2$ = Length
- $\eta_1 + \eta_2$ = Vocabulary
- $\eta_1 \times \log_2 \eta_1 + \eta_2 \times \log_2 \eta_2$ = Estimated Length

Length vs. Vocabulary

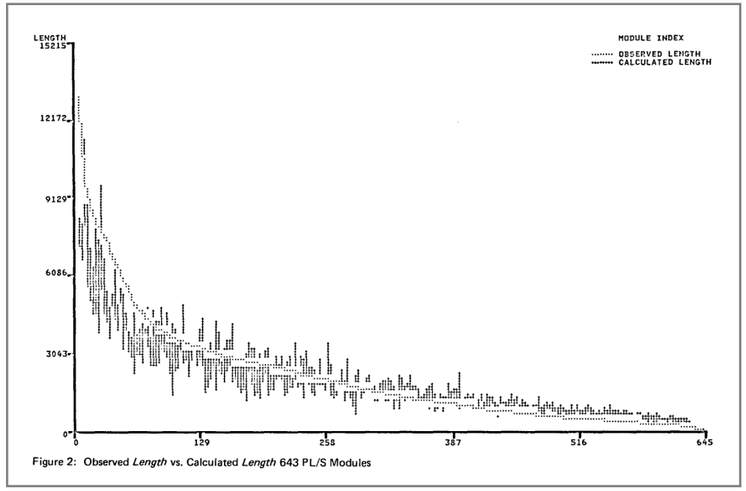


Figure 2: Observed *Length* vs. Calculated *Length* 643 PL/S Modules

"The size of a program, regardless of the metric used to measure size, is a function of the vocabulary of the program."

Source: Charles P. Smith. A Software Science Analysis of Programming Size. In *Proceedings of the Annual Conference*, pages 179–185, 1980. doi:10.1145/800176.809965

Volume, Difficulty, and Effort

- N = Length
- η = Vocabulary
- $N \times \log_2 \eta$ = Volume
- $\eta_1/2 + N_2/\eta_2$ = Difficulty
- $D \times V$ = Effort

Effort vs. Understandability

"We have independently tested the hypothesis that the mental effort required to create a program (measured by E) is related to a person's ability to understand a program or to find bugs in existing programs. The studies of Gould and Weissman as well as our work strongly support these hypotheses."

Source: Ann Fitzsimmons and Tom Love. A Review and Evaluation of Software Science. *ACM Computing Surveys (CSUR)*, 10(1):3–18, 1978. doi:10.1145/356715.356717

Bill Curtis

“In the current study, the complexity metrics were more highly related to the performance of less experienced programmers. Thus, the complexity metrics may not represent the most important constructs for predicting the performance of experienced programmers. These programmers probably conceptualized programs at a level other than that of operators, operands, and basic control paths.”

— Bill Curtis, Sylvia B. Sheppard, Phil Milliman, M. A. Borst, and Tom Love. Measuring the Psychological Complexity of Software Maintenance Tasks With the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering*, 5 (2):96–104, 1979. doi:10.1109/TSE.1979.234165

CC vs. Halstead Volume

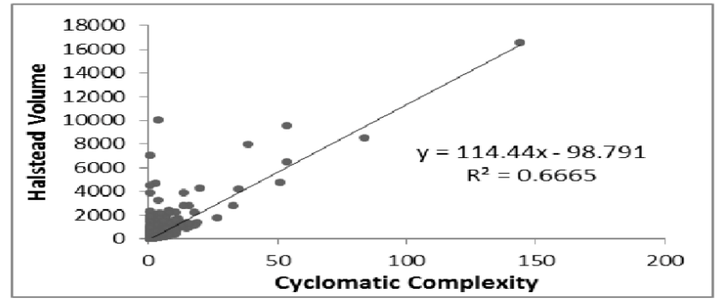


Figure 7: The correlation between Cyclomatic complexity and Hallstead volume

"From the Fig. 7, it is obvious that that the correlation between Cyclomatic Complexity and Halstead Volume is strong. The change in Cyclomatic Complexity will impact on Halstead Volume and vice versa."

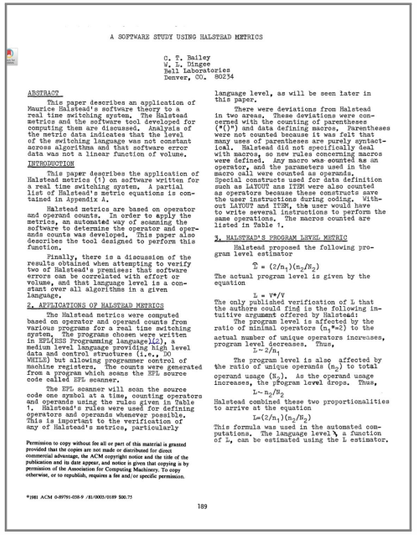
Source: Yahya Tashtoush, Mohammed Al-Maolegi, and Bassam Arkok. The Correlation Among Software Complexity Metrics With Case Study, 2014

Time and Bugs Estimate

- E = Effort
- $E/18$ = Time (in seconds)
- $V/3000$ = Bugs

“In studying some error data provided us by Rome Air Development Center, Phil Milliman and I found Halstead’s metric a remarkably accurate predictor of delivered bugs in a system developed with modern programming practices and tools.”

— Bill Curtis. Program Complexity and Software Errors: A Front End for Reliability, 1979



A SOFTWARE STUDY USING HALSTEAD METRICS

C. T. Bailey
W. L. Dingee
Bell Laboratories
Denver, CO. 80234

ABSTRACT

This paper describes an application of Maurice Halstead's software theory to a real time switching system. The Halstead metrics and the software tool developed for computing them are discussed. Analysis of the metric data indicates that the level of the switching language was not constant across algorithms and that software error data was not a linear function of volume.

INTRODUCTION

This paper describes the application of Halstead metrics [1] on software written for a real time switching system. A partial list of Halstead's metric equations is contained in Appendix A.

Halstead metrics are based on operator and operand counts. In order to apply the metrics, an automated way of scanning the software to determine the operator and operands counts was developed. This paper also describes the tool designed to perform this function.

Finally, there is a discussion of the results obtained when attempting to verify two of Halstead's premises: that software errors can be correlated with effort or volume, and that language level is a constant over all algorithms in a given language.

2. APPLICATIONS OF HALSTEAD METRICS

The Halstead metrics were computed based on operator and operand counts from various programs for a real time switching system. The programs chosen were written in RPL(Ring Programming language)[2], a medium level language providing high level data and control structures (i.e., DO WHILE) but allowing programmer control of machine registers. The counts were generated from a program which scans the RPL source code called RPL scanner.

The RPL scanner will scan the source code one symbol at a time, counting operators and operands using the rules given in Table 1. Halstead's rules were used for defining operators and operands whenever possible. This is important to the verification of any of Halstead's metrics, particularly language level, as will be seen later in this paper.

There were deviations from Halstead in two areas. These deviations were concerned with the counting of parentheses ("()") and data defining macros. Parentheses were not counted because it was felt that many uses of parentheses are purely syntactical. Halstead did not specifically deal with macros, so new rules concerning macros were defined. The macro name was used as an operator and the parameters used in the macro call were counted as operands. Special constructs used for data definition such as LAYOUT and ITEM were also counted as operators because these constructs save the user instructions during coding. Without LAYOUT and ITEM, the user would have to write several instructions to perform the same operations. The macros counted are listed in Table 1.

2. HALSTEAD'S PROGRAM LEVEL METRIC

Halstead proposed the following program level estimator

$$\hat{L} = (2/n_1)(n_2/N_2)$$

The actual program level is given by the equation

$$L = VV^*/V$$

The only published verification of L that the authors could find is the following intuitive argument offered by Halstead:

"The program level is affected by the ratio of minimal operators (n_1=2) to the actual number of unique operators increases, program level decreases. Thus,

$$L \sim 2/n_1$$

The program level is also affected by the ratio of unique operands (n_2) to total operand usage (N_2). As the operand usage increases, the program level drops. Thus,

$$L \sim n_2/N_2$$

Halstead combined these two proportionalities to arrive at the equation

$$L=(2/n_1)(n_2/N_2)$$

This formula was used in the automated computations. The language level, λ a function of L, can be estimated using the L estimator.

*1981 ACM 0-89791-050-9 /81/0003-0189 $00.75

“The authors analysis indicated that software errors were not a linear function of program volume. It is felt that errors are a result of many other factors besides those which make up the volume metric. Perhaps the correlation exists on a very large scale (over an entire software system).”

— C. T. Bailey and W. L. Dingee. A Software Study Using Halstead Metrics. In *Proceedings of the Workshop/Symposium on Measurement and Evaluation of Software Quality*, pages 189–197, 1981. doi:10.1145/800003.807928

Halstead Complexity is supported by a few tools:

- multimetric for C++, Java, Python, and many others
- JHawk (not free) for Java
- Halstead Metrics Tool for Java
- PhpStorm for PHP

Bibliography

C. T. Bailey and W. L. Dingee. A Software Study Using Halstead Metrics. In *Proceedings of the Workshop/Symposium on Measurement and Evaluation of Software Quality*, pages 189–197, 1981. doi:10.1145/800003.807928.

Bill Curtis. Program Complexity and Software Errors: A Front End for Reliability, 1979.

Bill Curtis, Sylvia B. Sheppard, Phil Milliman, M. A. Borst, and Tom Love. Measuring the Psychological Complexity of Software Maintenance Tasks With the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering*, 5(2):96–104, 1979. doi:10.1109/TSE.1979.234165.

Ann Fitzsimmons and Tom Love. A Review and Evaluation of Software Science. *ACM Computing Surveys (CSUR)*, 10(1):3–18, 1978. doi:10.1145/356715.356717.

Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., 1977. doi:10.5555/540137.

Maurice H. Halstead. Advances in Software Science. *Advances in Computers*, 18(1):119–172, 1979. doi:10.1016/S0065-2458(08)60583-5.

Charles P. Smith. A Software Science Analysis of Programming Size. In *Proceedings of the Annual Conference*, pages 179–185, 1980. doi:10.1145/800176.809965.

Yahya Tashtoush, Mohammed Al-Maolegi, and Bassam Arkok. The Correlation Among Software Complexity Metrics With Case Study, 2014.