1975

Toward a Theoretical Basis for Estimating Programming Efforts

M. H. Halstead

Report Number:
75-143

TOWARD A THEORETICAL BASIS FOR ESTIMATING PROGRAMMING EFFORTS

M. H. Halstead

Purdue University

May 1975

CSD-TR 143

143

TOWARD A THEORETICAL BASIS FOR ESTIMATING PROGRAMMING EFFORTS

M. H. Halstead
Purdue University

ABSTRACT

The measurement of static properties of small algorithms yeild data which, when combined with suitable assumptions, provide an equation for estimating the time required to program them. This equation, which contains no arbitrary constants, is tested against a small data sample, and the results do not invalidate the hypothesis.

TOWARD A THEORETICAL BASIS FOR

ESTIMATING PROGRAMMING EFFORT

M. H. HALSTEAD
Purdue University

The measurement of programmer productivity is certainly one of the most complex areas in Computer Science. Since Ida Rose [1] of the National Bureau of Standards noted, more than 20 years ago, that coding time approximated four instructions per man-hour, the field has been aware that longer programs usually, (but not always) take longer to program than short ones. As recently as his 1973 turing lecture, Dijkstra [2] noted that there was as yet no proof on the question of whether the time to implement a program increases linearly or as the square of program length.

The present note will borrow from the field of software physics [3 - 7] to obtain a theoretical relationship between a computer program and the mental effort required to implement it, and then test this relationship against one set of experimental data as reported by another author [8].

Given the four countable (hence measurable) parameters:

η_1 = Unique Operators Used,

η_2 = Unique Operands Used,

N_1 = Total Operators Used,

N_2 = Total Operands Used

in any algorithm in any language, and Letting:

$$\eta = \eta_1 + \eta_2$$

and

$$N = N_1 + N_2,$$

it has been shown [4, 5] and independently confirmed [8] that the relationship:

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (1)$$

yields a good estimate of the program length, N.

Further it has been shown experimentally [6] that the volume, V, and the level, L, when measured by:

$$V = N \log_2 \eta \qquad (2)$$

and

$$\hat{L} = \frac{\eta_1^*}{\eta_1} \frac{\eta_2}{N_2} \qquad (3)$$

where $\eta_1^* = 1 + M$ (4)

and M = Number of modules have a product

$$L \times V = V^* \quad (5)$$

which depends only upon the algorithm, and is reasonably invariant as that algorithm is translated from one programming language to another.

It can also be suggested that, to a first approximation, the total number of mental discriminations required to implement a preconceived algorithm in any language with which the programmer is fluent might be obtained in the following way.

Assume that each of the N items in a program is selected from its vocabulary, η, by means of a binary search. The number of comparisons required for each binary search will be, on the average, $\log_2 \eta$. Consequently, the total number of comparisons required to generate a program of length N will be $N \log_2 \eta$, which is nothing more than the program Volume, V, of equation 2. Now recall that the level as the term is intuitively used, and in the sense of Equation 3 also, is intended to represent the inverse of program difficulty. If these two assumptions are valid, then the total number of effective mental discriminations, E, required to generate a given program can be calculated from:

$$\hat{E} = \frac{V}{L} \quad (6a)$$

noting that $V^* = VxL$, it follows that equation 6a can also be expressed as:

$$\hat{E} = \frac{V^2}{V^*} \quad (6b)$$

In order to convert Equation 6 from units of effective mental discriminations to units of time, we may either proceed experimentally, or adopt previous results from Psychology. According to a most pertinent paper, "On the Fine Structure of Psychological Time", [9], if we let S represent the number of "moments" per second for the human brain, then:

$$5 \leq S < 20 \quad (7)$$

Equation 6 then becomes:

$$\hat{T} = \frac{V}{SL} \quad (8a)$$

or

$$\hat{T} = \frac{V^2}{SV^*} \quad (8b)$$

provided, of course, that we are dealing with a programmer who is enforcing the equivalent of the hardware instruction: "Inhibit all Interrupts". If he is not concentrating on the programming task, then Equation 8 should yield only a lower bound. Since computer programmers might be expected to fall near the high end of Stroud's range, and since an unpublished technical report on machine language rates [10] yielded the value experimentally, we will take S = 18 per second in the following analysis. (In an unreported, but seemingly valid test, Dijkstra apparently sustained a rate of 51/second for a three minute period, but then, even Stroud would not have expected a Dijkstra in his population.)

EXPERIMENTAL PROCEDURE

In an exhaustive report, Zislis [8] details the following procedure.

Non-procedural specifications were written for the twelve algorithms numbered 14, 16, 17, 19, 20, 21, 23, 24, 25, 29, 31 and 33 published in the Communications of the ACM [11]. He then programmed each of these algorithms in a language selected at random from the set: FORTRAN, PL/1 and APL, recording the times spent in coding, coding declarations, desk checking, and correcting errors revealed by desk checking. Times were recorded to the nearest minute, and summed for each program. (The experiment was then repeated with a second, a third, and the original language, but because of uncertanty in eliminating the effect of learning from those data, they will not be treated here.) After all programming had been completed, Zislis measured the parameters η_1, η_2, N_1, N_2. His data, taken from Appendix D, interation1, and Appendix F, η and N counts, are reproduced in Table 1.

Table 1. Data from Zislis Algorithm Implementation Experiment

Algorithm (CACM Nr.)	Implementation Time (Minutes)	η_1	η_2	N_1	N_2
14	33	15	17	64	51
16	135	20	35	223	303
17	33	15	15	78	81
19	7	10	6	25	19
20	12	14	19	59	38
21	43	23	25	106	97
23	21	17	13	50	50
24	16	15	14	81	82
25	62	26	34	179	163
29	25	7	11	72	67
31	20	17	25	53	54
33	4	6	8	9	15

For purposes of calculation, Equation 8 may be expanded, algebraically, to:

$$\hat{T} = \frac{1}{\eta_1^* S} \times \frac{\eta_1}{\eta_2} \times N_2 \ (N_1 + N_2) \ \log_2 \ (\eta_1 + \eta_2) \qquad (9)$$

where, for $\hat{T}$ in minutes, S = 18 x 60 = 1080/Min.

Now η_1^*, the only parameter not recorded by Zislis, is defined as the number of operators required to express a procedure call upon a given algorithm. Since in most cases this may consist merely of a single grouping or assignment operator, plus the name of the procedure itself, its value is usually 2. However, a procedure call may need to specify another procedure among its operands, and since any procedure or function name is an operator, this has the effect of increasing η_1^*. Examining the twelve algorithms in the sample, we find that ten of them are indeed single procedures, for which $\eta_1^* = 2$. Algorithm 16, on the other hand, consists of the three procedures CROUT, INNERPRODUCT, and SOLVE, hence for it $\eta_1^* = 1 + 3 = 4$. Similarly, algorithm 25 specifies both the procedure ZEROS and FUNCTION, for an $\eta_1^* = 1 + 2 = 3$.

Table 2 contains the result of applying Equation 9 to the data of Table 1, and in addition it also contains a count of the number of executable statements in each of the original algol inplementations. The latter can be taken as a measure of "Program Length" in its historical sense, hence the algorithms have been ordered according to that parameter. The steps performed in the calculation of coefficients of correlation between "Length" and observed programming times, and between observed and theoretically calculated programming times are shown at the bottom of the table.

Table 2. Analysis of Zislis Experiment

Algorithm (CACM Nr.)	Number of Statements	T(obs) (Minutes)	$\hat{T}$(Eq.9) (Minutes)
33	1	4	0.5
19	5	7	2.6
20	6	12	6.3
23	7	21	14.9
24	8	16	32.2
17	9	33	29.3
21	12	43	46.8
29	14	25	11.4
31	15	20	9.8
14	22	33	12.0
16	35	135	121.9
25	57	62	77.7
ΣX	191	411	365.4
$\bar{X}$	15.92	34.25	30.45
ΣX^2	5779	28027	25261.
$n\bar{X}^2$	3041	14077	11126
$\sqrt{\Sigma X^2 - n\bar{X}^2}$	52.33	118.11	120.40
$\Sigma X_{obs} X_i$		10834	26054
$n\bar{X}_{obs}\bar{X}_i$		6543	12515
$\Sigma X_{obs} X_i - n\bar{X}_{obs}\bar{X}_i$		4291	13539
$r = \frac{\Sigma X_{obs} X_i - n\bar{X}_{obs}\bar{X}_i}{\sqrt{\Sigma X_{obs}^2 - n\bar{X}_i^2}\,\sqrt{\Sigma X_i^2 - n\bar{X}_i^2}}$		.694	.952

The analysis in Table 2 clearly indicates two things. First, while the coefficent of correlation, r, between the times required to program these algorithms and a classical measure of their lengths is both positive and acceptably high, 0.694, the correlation between the observed times and those calculated with Equation 9 is considerably higher, 0.952.

Second, in the case of $\hat{T}$, the units are also in minutes.

Clearly, just as one robin does not make a spring, one experiment can not validate a theory. As had long been recognized in the natural sciences, however, additional experiments at one installation can never gauranttee their reproducibility. All that can be said is that the results presented here appear to be of sufficient potential interest to warrent additional experimentation by others.