Toward a Theoretical Basis for Estimating Programming Efforts

M. H. Halstead

Report Number:
75-143


This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
TOWARD A THEORETICAL BASIS FOR ESTIMATING PROGRAMMING EFFORTS

M. H. Halstead
Purdue University
May 1975

CSD-TR 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:
\[
\begin{align*}
\eta_1 &= \text{Unique Operators Used}, \\
\eta_2 &= \text{Unique Operands Used}, \\
N_1 &= \text{Total Operators Used}, \\
N_2 &= \text{Total Operands Used}
\end{align*}
\]
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:
\[
N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2
\]
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:
\[
\begin{align*}
V &= N \log_2 \eta \\
L &= \frac{\eta_1 \eta_2}{N_1 N_2}
\end{align*}
\]

(2)

(3)
where $n_1 = 1 + M$ \hfill (4)
and $M = \text{Number of modules have a product}$

$L \times V = V^*$ \hfill (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 vocab­
ularies, $n_i$, by means of a binary search. The number of comparisons required for
each binary search will be, on the average, $\log_2 n_i$. Consequently, the total
number of comparisons required to generate a program of length $N$ will be
$N \log_2 n_i$, 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:

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

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

$$E = \frac{V^2}{V^*}$$ \hfill (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:

$$5 \leq S \leq 20$$ \hfill (7)

Equation 6 then becomes:

$$T = \frac{V}{SL}$$ \hfill (8a)
or

$$T = \frac{V^2}{SV^*}$$ \hfill (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/l 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 uncertainty 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 \( \eta_1, \eta_2, N_1, N_2 \). His data, taken from Appendix D, iteration1, and Appendix F, \( \eta \) and \( N \) counts, are reproduced in Table 1.

<table>
<thead>
<tr>
<th>Algorithm (CACM Nr.)</th>
<th>Implementation Time (Minutes)</th>
<th>( \eta_1 )</th>
<th>( \eta_2 )</th>
<th>( N_1 )</th>
<th>( N_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>33</td>
<td>15</td>
<td>17</td>
<td>64</td>
<td>51</td>
</tr>
<tr>
<td>16</td>
<td>135</td>
<td>20</td>
<td>35</td>
<td>223</td>
<td>303</td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>15</td>
<td>15</td>
<td>78</td>
<td>81</td>
</tr>
<tr>
<td>19</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>59</td>
<td>38</td>
</tr>
<tr>
<td>21</td>
<td>43</td>
<td>23</td>
<td>25</td>
<td>106</td>
<td>97</td>
</tr>
<tr>
<td>23</td>
<td>21</td>
<td>17</td>
<td>13</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>81</td>
<td>82</td>
</tr>
<tr>
<td>25</td>
<td>62</td>
<td>26</td>
<td>34</td>
<td>179</td>
<td>163</td>
</tr>
<tr>
<td>29</td>
<td>25</td>
<td>7</td>
<td>11</td>
<td>72</td>
<td>67</td>
</tr>
<tr>
<td>31</td>
<td>20</td>
<td>17</td>
<td>25</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>33</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>15</td>
</tr>
</tbody>
</table>
For purposes of calculation, Equation 8 may be expanded, algebraically, to:

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

(9)

where, for \( T \) in minutes, \( S = 18 \times 60 = 1080/\text{Min} \).

Now \( \eta_1^* \), 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 \( \eta_1^* \). 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 implementations. 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

<table>
<thead>
<tr>
<th>Algorithm (CACM Nr.)</th>
<th>Number of Statements</th>
<th>T(obs) (Minutes)</th>
<th>T(Eq.9) (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>1</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>7</td>
<td>2.6</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>12</td>
<td>6.3</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
<td>21</td>
<td>14.9</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>16</td>
<td>32.2</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>33</td>
<td>29.3</td>
</tr>
<tr>
<td>21</td>
<td>12</td>
<td>43</td>
<td>46.8</td>
</tr>
<tr>
<td>29</td>
<td>14</td>
<td>25</td>
<td>11.4</td>
</tr>
<tr>
<td>31</td>
<td>15</td>
<td>20</td>
<td>9.8</td>
</tr>
<tr>
<td>14</td>
<td>22</td>
<td>33</td>
<td>12.0</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>135</td>
<td>121.9</td>
</tr>
<tr>
<td>25</td>
<td>57</td>
<td>62</td>
<td>77.7</td>
</tr>
<tr>
<td>EX</td>
<td>191</td>
<td>411</td>
<td>365.4</td>
</tr>
<tr>
<td>( \overline{X} )</td>
<td>15.92</td>
<td>34.25</td>
<td>30.45</td>
</tr>
<tr>
<td>EX^2</td>
<td>5779</td>
<td>28027</td>
<td>25261</td>
</tr>
<tr>
<td>nX^2</td>
<td>3041</td>
<td>14077</td>
<td>11126</td>
</tr>
<tr>
<td>( \sqrt{EX^2-nX^2} )</td>
<td>52.33</td>
<td>118.11</td>
<td>120.40</td>
</tr>
<tr>
<td>EX_\text{obs}_i \overline{X}</td>
<td>10834</td>
<td>26054</td>
<td></td>
</tr>
<tr>
<td>nX_\text{obs}_i \overline{X}</td>
<td>6543</td>
<td>12515</td>
<td></td>
</tr>
<tr>
<td>EX_\text{obs}<em>i \overline{X} \overline{X}</em>\text{obs}_i</td>
<td>4291</td>
<td>13539</td>
<td></td>
</tr>
<tr>
<td>( r = \frac{EX_\text{obs}<em>i \overline{X} \overline{X}</em>\text{obs}<em>i}{\sqrt{EX</em>\text{obs}<em>i \overline{X}^2} \sqrt{EX</em>\text{obs}<em>i \overline{X}</em>\text{obs}_i^2}} )</td>
<td>.694</td>
<td>.952</td>
<td></td>
</tr>
</tbody>
</table>

The analysis in Table 2 clearly indicates two things. First, while the coefficient 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 \( T \), 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 guarantee their reproducibility.
ucibility. All that can be said is that the results presented here appear to be of sufficient potential interest to warrant additional experimentation by others.