Approximations for the Halstead Software Science Software Error Rate and Project Effort Estimators

Victor Schneider
Computer Science Department
Framingham State College
Framingham, MA 01701

ABSTRACT

Experimental estimators are presented relating the expected number of software errors (B) in a software development project to

* the overall reported months of programmer effort for the project (E)

* the number of subprograms (n)

* the count of thousands of coded source statements (S)

These estimators are

$$B \approx 7.6\, E^{0.667}\, S^{0.333}$$

and

$$B \approx n \left(\frac{S/n}{.047} \right)^{1.667}$$

These estimators are shown to be consistent with data obtained from the air force Rome air development center, the naval research laboratory, and Fujitsu corporation. It is suggested here that more data is needed to refine these estimators further.

INTRODUCTION

This paper presents several promising estimators for software problem reports (denoted by B) collected during a software development project. These estimators are extensions of the Halstead Software Science estimators [1], in that they use software metrics more easy to obtain than those in the original Halstead version.

As a first example of this recasting process, Halstead found that his measure of program length (N) can be approximated by the empirically derived conversion:

$$N = 7500S \qquad (1)$$

with S in thousands of source statements, minus comment lines. Furthermore, the value of η, the Halstead count of distinct operators and operands in a program, can be obtained from the approximation [2]:

$$\log_2 \eta = N^{0.32} \qquad (2)$$

which holds true for $50 \leq N \leq 500$, the range of subprogram sizes. Since the values of N and $\log_2 \eta$ appear in many of the basic Halstead estimators, approximations (1) and (2) can be substituted into the Halstead estimators, leading to:

$$E = 98 \sum_{i=1}^{n} S_i^2 \qquad (3)$$

where the S are the (fractional) counts of thousands of source language statements. Reference [2] reports a study of 21 large software projects, in which estimator (3) yielded significant correlations between estimated and reported efforts. An approximation to (3), that usually yields estimates within 25% of more careful estimates, assumes that variance of subprogram sizes is "small", and is given by:

$$E \approx \frac{98S^2}{n} \qquad (4)$$

with

$$S = \sum_{i=1}^{n} S_i \qquad (5)$$

The basic Halstead software error rate estimator is given by:

$$B \propto E^{2/3} \qquad (6)$$

with B the expected count of software problem reports collected during the coding, integration, and final testing phases of a software development project whose reported effort is E. Halstead calibrated (6) using data from Fujitsu Corporation [3] and obtained good agreement between his estimates and that data.

I subsequently wrote to F. Akiyama of Fujitsu asking if he had saved data on effort associated with submodules of the software system reported in his paper [3], and was sent the data presented in Table 1 below [4]. On reviewing this data, I discovered that the estimator:

$$B = 7.6\, E^{0.667} S^{0.333} \qquad (7)$$

gave better agreement with the data than (6). Solving for E in (7), I obtained:

$$E = \frac{B^{1.5}}{21S^{0.5}} \qquad (8)$$

which provided effort estimates for the modules comprising the most recently received Fujitsu data.

Table 1 below shows Akiyama's reported module efforts together with the estimates for each module and for the entire software project. Estimator (8) is additive with this data: Combining module data in pairs and triples, etc., still yields accurate effort estimates. (Note that the module A information differs from the Akiyama paper [3], as well as the value reported in [1], [11], and [12]. Substituting these module A values into the results of [11] improves the agreement between the estimators in [11] and the data. I have recently written to Akiyama about this discrepancy between [3] and [4], but received no reply.)

TABLE 1: ESTIMATOR 8 APPLIED TO THE FUJITSU SOFTWARE DATA

Module	S	B	E	$B^{1.5} / (21S^{0.5})$	
A	6.132	102	14	20	
B	1.329	18	7	3	
C	5.453	93	17	19	
D	1.674	26	8	5	
E	2.051	71	17	20	
F	2.513	37	9	9	
TOTAL	19.152	347	72	70	$\Sigma E_i = 74$

OTHER SOURCES OF SOFTWARE ERROR-RATE DATA

In addition to the Fujitsu data, there is a software error data base obtainable from the Rome air development center (RADC) [5], an individual project reported by Weiss [7] and an earlier study by Bell and Sullivan [8]. However, only the Weiss study lists individual subprogram size [9], and the RADC data lists only overall source statement counts and errors. In what follows, estimators (7) and (8) are recast to show their consistency with these sources of data.

As a first example, Weiss's data contains reported software problem counts (B), overall count of source statements (S), and the number of subprograms (denoted by n). Substituting approximation (4) into estimator (7) gives us:

$$S \approx \frac{162S^{1.667}}{n^{0.667}} = n\left(\frac{S/n}{0.047}\right)^{1.667} \qquad (9)$$

From Weiss's data, we have

$$B \approx 253\left(\frac{0.036}{0.047}\right)^{1.667} = 162$$

which compares to a reported value of 125.

Weiss's data also lists overall project effort as "192 man weeks" [9] which, substituted into (7), yields

$$B \approx (7.6)(48)^{0.667}(9.108)^{0.333} = 210.$$

As a second example, the RADC data [5] has an upper bound of approximately

$$B_{ub} \approx 70\ S^{1.07} \qquad (10)$$

By substituting the RADC upper bound on software project effort [5]:

$$E_{ub} \approx 40\ S \qquad (11)$$

from (10) into (7), we obtain an estimate of the upper bound on software problem reports in the RADC data base:

$$B_{ub} \approx 7.6(40\ S)^{0.667}\ S^{0.333} = 87S \qquad (12)$$

There are 30 points in the RADC software error data base, and (10) is delineated by four of these points [5]. However, the RADC software effort data base has 400 points, and (11) is clearly its upper bound, which makes (12) an interesting result.

Furthermore, there is an industry average of approximately 20 software problem reports per thousand statements, and, substituting the RADC regression line for the software effort data

$$E \approx 4.9\ S^{0.976} \qquad (13)$$

$$B_{\text{average}} \approx 7.6(4.9\ S^{0.976})^{0.667}\ S^{0.333} = 21.9S^{0.98} \quad (14)$$

The estimate of (14) is respectably close to this industrial average and to the regression error rates obtained by RADC.

DELIVERED ERRORS IN TESTED PROGRAMS

Bell and Sullivan's study of "public algorithms" that were published in the Communications of the ACM [8] indicates that Halstead program length works as a discriminator for delivered errors in small algorithms. "Halstead length alone appears to be an excellent predictor of correctness. No correct program had (N) greater than 237, and only one erroneous program...had (N) less than 284...." [p. 273] They further state that N = 267 is the length above which a program is almost certain to contain an error [p. 76], N = 162 is the average length of all correct programs in their survey [p. 72], and N = 515 is the average length of all programs with a delivered error.

For comparison, setting B = 1/2 in (8) yields:

$$B = 1/2 = 162S^{1.667}$$

or

$$S = 0.31 \text{ (or 31 source statements)}$$

and

$$N = 7500S = 234$$

Thus, (8) predicts a Halstead length of 234 for a single subprogram with an expected error count of 1/2.

Furthermore, the length of a subprogram with B = 1/4 is given by:

$$1/4 = 162S^{1.667}$$

or

$$S = 0.021 \text{ (or 21 statements)}$$

and

$$N = 7500S = 154.$$

One interpretation of this consistency test is that, for a subprogram of 21 statements,

$P \quad = 1/4 =$ the probability that a problem is found during

$P_{B=0} = 3/4$ = the probability that a problem is not found

For a program consisting of n subprograms of 21 statements, the probability of that program containing a total of i software problems is given by:

$$p(i) = (3/4)^{n-i} (1/4)^{i} \left(\frac{n!}{(n-i)!} \right) \qquad (15)$$

and the expected number of problems remaining after k problems are found during testing is given by:

$$B_k = \frac{\sum_{i=k}^{\infty} (i-k)\, p(i)}{1 - \sum_{i=0}^{k-1} p(i)} \qquad (16)$$

Suppose we ask what is the smallest program for which

k is the expected value of B and $B_k = 1$?

Put another way, what is the length of the smallest program likely to have one error remaining after system test and delivery? By iterating with (15) and (16), we discover that, for n = 3,

$B = (3)(0.25) = 0.75$

and

$B_1 = 1.4$

Thus, the number of problems predicted during coding, integration, and system test of a program with three subprograms of 63 statements is 0.75, and the expected number of delivered errors is 1.4.

The Halstead length corresponding to 63 statements is

$N = (7500)(0.063) = 473$

which is respectably close to the Bell and Sullivan average of N = 515 for erroneous programs.

AVERAGE COST TO REPAIR A SOFTWARE ERROR AFTER DELIVERY

If estimator (8) is correct, it can be differentiated to obtain (dE/dB), the marginal effort to repair a software error. However, this derivative needs to be interpreted. For example, E is the overall software project effort, including testing, documentation, and system validation. After delivery of a software product to the customer, fixing software errors generally involves partial recoding of individual subprograms and unit testing. Such activities correspond to only 30% of overall software project costs. So, it seems reasonable to use 30% of (dE/dB) in estimating the cost to repair an error discovered after delivery. We thus obtain:

$$0.3\ (dE/dB) = 0.022(B/S)^{0.5} = 0.28(S/n)^{0.333} \qquad (17)$$

Here, (dE/dB) is evaluated at the time of delivery, with S and B the overall project values during development.

The data presently available to me for comparison is Weiss's data [7, 9], in which a total of 18 errors were discovered in the year after delivery, implying an estimate from (17) of 1.66 months of post delivery debugging effort. The actual reported debugging effort after delivery was approximately 1.3 months [9], which is in good agreement with the estimate of (17). Anyone with data that can be used to validate estimator (17) is invited to write to me at the Framingham State College Computer Science Department, Framingham, MA 01701.

As a last point, typical software projects discover a total of 15% of the count of development errors after delivery. Then,

$$B_{rem} \approx 0.15B = 0.15n\left(\frac{S/n}{0.047}\right)^{1.667}$$

and the corresponding effort to repair these discovered errors is approximately

$$E_{rem} \approx (0.15n)\left(\frac{S/n}{0.047}\right)^{1.667}(0.28)(S/n)^{0.333} = \frac{6.98}{n}^{2}$$

or 7% of the charged project effort.

ACKNOWLEDGMENT

The estimators for errors remaining in delivered programs were borrowed from a report by M. H. Halstead and L. Cornell [10].

REFERENCES

1. M. H. Halstead, Elements of Software Science. Elsevier Publishing Company, 1977.

2. V. B. Schneider, New Data Supporting the Software Project Costing Formulas, 1979 IEEE Computer Software and Applications Conference, November, 1979, pp. 708-711.

3. F. Akiyama, An Example of Software System Debugging, Proceedings of the International Federation of Information Processing Societies Congress, 1971, pp.353-359.

4. F. Akiyama, Personal letter, March, 1978.

5. R. Nelson, Software Data Collection and Analysis (Draft-Partial Report), Rome air development center Griffiss air force base, New York, 13441, September, 1978.

6. A. N. Sukert, A Four-Project Empirical Study of Software Error Prediction Models, 1978 IEEE Computer Software and Applications Conference, November,1978, pp. 577-582.

7. D. M. Weiss, Evaluating Software Development by Error Analysis: The Data from the Architecture Research Facility. Naval research laboratory report 8268, December, 1978.

8. D. E. Bell and J. E. Sullivan, Further Investigations into the Complexity of Software, MITRE corporation technical report MTR-2874, Volume II, June, 1074.

9. D. M. Weiss, Personal letters, August, 1979, and April, 1980.

10. L. Cornell and M. H. Halstead, Predicting the Number of Bugs Expected in a Program Module, Purdue University Computer Sciences Dept. Tech. Rep. CSD-TR-205, 1976.

11. J. E. Gaffney, Estimating the Number of Faults in Code, IEEE Transactions on Software Engineering, SE-10, 4, July, 1984, pp. 459-464.

12. S. D. Conte, H. E. Dunsmore, and V. Y. Shen, Software Engineering Metrics and Models, Benjamin/Cummings, 1986.