1975

Using the Methodology of Natural Science to Understand Software

M. H. Halstead

Report Number:
76-190

USING THE METHODOLOGY OF NATURAL SCIENCE TO UNDERSTAND SOFTWARE

M. H. Halstead
Purdue University
CSD TR 190

Prepared as a Panelist's Position Paper for Dr. Roger Firestone's Session at NCC-76.

In the next ten minutes I would like to interest you in an area of scientific investigation which a growing number of us find fascinating. Perhaps I should warn you that some have described it as "addictive". It has been called by various names: Algorithm Dynamics, Software Physics, Software Theory and even Software Science, but in general terms it is nothing but the application of the experimental methodology of the natural sciences to the study of the properties and structure of computer programs.

Rather than dwelling upon those discoveries which have been made in this field since 1972, I prefer to spend the available time in explaining and illustrating the method itself.

For this purpose I have selected two examples which, I hope, will be interesting in themselves. The first involves the use of a single language on a number of different algorithms, where the objective is the investigation of the effect of the number of transfers of control (GO TO's) upon the Language Level. The second deals with an attempt to extend the effort equation, just discussed by Gordon, into the area of predicting program bugs.

Before starting on the first we must add one relationship to those used by Gordon:

$$\lambda = L^2V$$

Where λ is the Language Level, and both program level, L, and volume, V, are identical to those just explained in the previous paper. Now studies have shown that unlike L, λ does not vary widely over a single language. It has been noticed, however, that the variance in observed values of λ tends to increase as its mean value increases. Using the parameter counts of Professor Zweben for a few computer languages, and of Professor Kulm for English passages gave the following results.

λ FOR SMALL SAMPLES

	Mean	Variance
English prose	2.16	.74
PL/I	1.53	.92
Algol 58	1.21	.74
Fortran	1.11	.83
Pilot	.92	.43
Assembly (CDC)	.88	.42

While it is intuitively satisfying to note, perhaps, that we should expect that a higher level language must provide more alternative ways of expressing a solution, and consequently exhibit a larger variance, this is intuitive still.

If we are to actually use the scientific method, we must not be content with intuition. Instead, let us test the hypothesis that frequent use of the GO TO reduces the level of a language, and that variations in its use may contribute to the variance. For this experiment we define ν as the number of transfers of control within a program, and the block size, β, becomes:

$$\beta = \frac{N}{\nu + 1}$$

Zweben's Pilot programs then give the following results.

ZWEBEN'S PILOT PROGRAMS

ACM#	λ	β
14	2.15	62
1	1.31	24
6	1.30	41
8	.92	18
3	.90	27
4	.90	11
10	.90	10
12	.84	12
11	.78	11
13	.72	12
9	.67	22
5	.56	16
7	.46	7
2	.41	18

Correlation: $r = 0.86$

From these observations, it seems clear that frequent use of the GO TO does indeed contribute to a lowering of the language level, and that variations in its use contribute to the observed variance.

Before leaving this example, perhaps we should point out that the effort equation, just expressed by Gordon as:

$$E = \frac{V}{L}$$

can be transformed algebraically to:

$$E = \frac{(L \times V)^3}{\lambda^2}$$

Since the product LV is constant for a given problem, programming effort must vary as the inverse square of λ. The effect of the block size thus becomes apparent.

The second example, as I indicated, will deal with an attempt to understand the frequency, but not the nature, of programming bugs. For this experiment we will rely upon observed values published in three papers, and test the extent to which they are consistent. The first of these is a report by Bell and Sullivan of MITRE, in which they separated certified algorithms in CACM into two classes. Their first class consisted of programs which the certifier found correct as published, and the second consisted of those for which the certifier had found and corrected a bug.

Using this definition of a "Delivered Bug", Bell and Sullivan then calculated the mean value of N for each class, reporting:

$$\overline{N}_o = 161.9$$

$$\overline{N}_x = 515.4$$

The second paper we will need is Akiyama's IFIPS-71 paper given at the Yugoslavia meeting. In that paper he gave both the number of bugs, and adequate program characteristic data for each of the nine modules of a large Japanese system. Funami then used Akiyama's program characteristic data to calculate E for each module, in a paper given at the Polytechnic Software Engineering Symposium here six weeks ago.

Now it would seem reasonable, as pointed out in the Funami paper, that since E represents the count of elementary mental discriminations required, it should be correlated with the number of discriminations made incorrectly. Indeed it was, and he found a correlation coefficient of 0.982 between E and B.

In view of this high correlation, it might at first appear that relation

$$B = E/Ecrit$$

would apply, where Ecrit is the average number of discriminations between errors. A bit of reflection, however, suggests that while the relation may be simple, it is not quite that simple. For example, in almost any program, there will be some amount of redundancy. Consequently, even while writing a single program, there will be some learning taking place. Following experience with "Learning Curves" in industry, we would therefore expect

$$B = E^{\ell}/Ecrit$$

where the exponent ℓ is somewhat less than 1.

Now, from the two data points from Bell and Sullivan, it is possible to solve for both ℓ and Ecrit, and then to test them on Akiyama's data.

First, however, we must convert from observed values of N to the corresponding values of E. This is only possible for average values, of course, and then only if we have a mean value of λ. In the Bell and Sullivan sample, which was in Algol, we may use the value 1.21 shown earlier. Since several steps are required, each requiring one of the software relations, we may show them in the following way:

1. From Bell and Sullivan:

   $\bar{N}_o = 161.9$ $\qquad\qquad$ $\bar{N}_x = 515.4$

2. From $N = \eta \log_2 (\eta/2)$

   $\eta_o \doteq 38.1$ $\qquad\qquad$ $\eta_x = 93.0$

3. From $V = N \log_2 \eta$

   $V_o = 850$ $\qquad\qquad$ $V_x = 3370$

4. From $E = \sqrt{V^3/\lambda}$

   $E_o = 22529$ $\qquad\qquad$ $E_x = 177894$

Now for their sample with one error, the mean number of errors can be conviently taken as one. But for their sample with zero errors, the true mean can not be zero. In going from the discrete to the continuous, we see that everthing below one half will round to zero. Assuming a uniform distribution between .5 and 0, the mean would be one fourth, rather than zero.

We then have:

$$\ell = \frac{\log (1/.25)}{\log (177849/22529)} = .6710$$

$$\text{Ecrit} = 177849^{\ell} = 3331$$

Or

$$\hat{B} = E^{.6710}/3331$$

Testing this equation, we find the following:

CALCULATING "DELIVERED" BUGS

Module	B	E (Millions)	$\hat{B} = E^{.671}/3331$
MA	102	170.3	100
MB	18	15.3	20
MC	146	322.6	154
MD	26	28.4	30
ME	71	100.2	70
MF	37	65.5	53
MG	16	6.5	11
MH	50	58.5	49
MX	80	135.9	86
Sum	546		573

Correlation: $r = 0.990$

From the foregoing, it seems to some of us that the traditional methods of the natural sciences have much to recommend them as we try to understand our own. We might even suggest that in the field of computer programming, the long day of "Expert Opinion", or worse, the "Opinionated Expert", may be reaching its welcome sunset.

References

Funami, Yasuo and M.H. Halstead, "Software Physics Analysis of Akiyama's Debugging Data", Proc. MRI XXIV International Symposium: Software Engineering, Polytechnic Press, New York, to appear.

Gordon, R.D. and M.H. Halstead, "An Experiment Comparing Fortran Programming Times with the Software Physics Hypothesis", AFIPS Conf. Proc., V 45, 1976 National Computer Conference.

REFERENCES

[1] Rose, Ida, "Programming Productivity", (With the proper reference lost in antiquity, this information passed from one computer center manager to another during the 1950's).

[2] Dijkstra, Edsger W. "The Humble Programmer", Comm. ACM V 15, N 10, (Oct. 1972) pps 859-866.

[3] Halstead, M. H., "Natural Laws Controlling Algorithmic Structure?", ACM SIGPLAN Notices, V 7, N 2 (Feb 1972).

[4] Halstead, M. H. and Rudolf Bayer, "Algorithm Dynamics", Proc ACM Annual Conference, 1973. Atlanta.

[5] Bulut, Necdet, M. H. Halstead and Rudolf Bayer, "The Experimental Verification of a Structural Property of Fortran Programs", Proc. ACM Annual Conference, 1974.

[6] Bulut, Necdet, "An Inariant Property of Algorithms", Ph.D. Thesis, Purdue, August 1973.

[7] Halstead, M. H., "Software Physics Comparison of a Sample Program in DSL ALPHA and COBOL", IBM Research Report R. J. 1460, October, 1974.

[8] Zislis, Paul, "An Experiment in Algorithm Implementation", CSD Tech. Rpt. No. 96, Purdue, June 1973.

[9] Stroud, John M., "The Fine Structure of Psychological Time", Annals of the New York Academy of Sciences, 1966, pps 623-631.

[10] Halstead, M. H., "A Theoretical Relationship Between Mental Work and Machine Language Programming", CSD Tech. Rpt. No. 67, Purdue, May 1972.

[11] Beam, A., "Algorithm 14, Complex Exponential Integral", Comm. of the ACM, V 3, N 7, pp 406 (July 1960) and following.

[12] Bohrer, Robert, "Halstead's Criterion and Statistical Algorithms", Proc 8th Computer Science/Statistics Interface Symposium, Feb 1975, Los Angeles.