1976

Predicting the Number of Bugs Expected in a Program Module

Linda M. Ottenstein

Victor B. Schneider

Maurice H. Halstead

Report Number:
76-205

PREDICTING THE NUMBER OF BUGS EXPECTED IN A PROGRAM MODULE

Linda M. Ottenstein
Victor B. Schneider
Maurice H. Halstead

Computer Sciences Department
Purdue University
West Lafayette, Indiana 47907

CSD-TR 205
October 1976

Revised: January 1977

Abstract: Determining the amount of time to allot for debugging of programs is of prime importance in producing reliable software on schedule. An estimate for the expected number of bugs in a program module could be one aid in solving this problem. In this paper we present a relationship to give such an estimate and offer some preliminary verification of its accuracy.

PREDICTING THE NUMBER OF BUGS EXPECTED IN A PROGRAM MODULE

INTRODUCTION

With the current concern for reliable software, one major topic of discussion today is the occurrence of programming errors. Some researchers such as Gannon and Horning [GaH75] are interested in the impact of language design on the number of bugs. Others such as Litecky and Davis [LiD76] are more interested in error diagnosis by compilers and the effect this might have on programmers. Still others such as Bell and Sullivan [BeS74] use the number of bugs in a program module as a measurable quantity to correlate with theoretical calculations of the complexity of software. Because of the difficulty in collecting such data, good quantitative results about program bugs are limited. Although the term "bug" has intuitive meaning, it is hard to define rigorously. In this paper, the concept that we are dealing with is that of "delivered bugs" as used by Bell and Sullivan.

Akiyama [Aki71] showed that in the development of a particular large software system the number of bugs found in a module was more closely correlated to the number of

comparisons and subroutine calls in that module than to the module's length. An extension of this phenomenon was recently analyzed by Funami and Halstead [FuH76] utilizing the principles of software science. Using the data available from Akiyama, they calculated estimates of E, the number of effective mental discriminations needed to write a module, finding it highly correlated with the number of bugs.

In this paper we expand on this foundation by presenting a relationship to predict from E the average number of bugs expected in a module. Some preliminary verification of the hypothesis is presented based on two distinct sets of data. Although by no means conclusive, the results are interesting and therefore, we feel, worth reporting.

The first section of this paper is a reproduction of earlier software science results necessary for the development of the hypothesis. The second section presents the motivation for the hypothesis and some results using Akiyama's data. The third section shows the use of the formula to predict the actual results as found by Bell and Sullivan and also how the hypothesis could have been obtained from their data. The final section derives a formula for calculating the probability of bugs remaining in a programming project as a function of the number of bugs

already found.

SOFTWARE SCIENCE FUNDAMENTALS

It has been shown that several important program properties can be obtained from two simple measures of the expressed algorithm. These measures are the number of distinct operators and the number of distinct operands used to express the algorithm. The following tested hypotheses from software science [Hal77] are needed for the development of the model:

Estimated program length, $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$ (1)

Program volume, $V = N \log_2 \eta$ (2)

Minimum program volume, $V^* = (\eta_1^* + \eta_2^*) \log_2 (\eta_1^* + \eta_2^*)$ (3)

Program level, $L = V^*/V$ (maximum value of 1) (4)

Language level, $\lambda = LV^*$ (5)

Number of mental discriminations needed, $E = V/L$ (6)

where

η_1 = number of unique operators used in a program

η_2 = number of unique operands used in a program

η_1^* = minimum number of unique operators needed to express the algorithm in a potential language
= 2

η_2^* = minimum number of conceptually unique operands needed to express the algorithm in a potential

language

= the number of input-output parameters

$\eta = \eta_1 + \eta_2$

N_1 = total number of operator usages

N_2 = total number of operand usages

$N = N_1 + N_2$ = actual length

DEVELOPMENT OF HYPOTHESIS

It is intuitively seen that many factors affect the number of bugs in a program module at any particular point in the module's development. One of the most pronounced factors is the program length as measured by lines of code. Recent work has shown that along with this length, the complexity of the program needs to be accounted for. The correlation found by Akiyama between the length of the modules and the number of bugs was .83. He then showed that the correlation between the number of subroutine calls plus the number of decisions (one possible measure of complexity) and the number of bugs was higher, being .92.

Based on Akiyama's data, Funami and Halstead were able to estimate E, the number of mental discriminations, for each module. The correlation coefficient reported between E and the number of bugs was .982. One would not expect this correlation to always be this high. Many other factors,

such as programmer experience, method of programming, and amount of available machine time, must also have an effect on the number of bugs. It appears, however, that many of the complicating factors were not important in Akiyama's experiment. The data, therefore, should be useful in discovering basic relationships.

In one simple model, using Funami and Halstead's calculations to predict the number of module bugs from the number of mental discriminations, a simple hypothesis might be that the number of bugs is a function of the number of mental discriminations and some constant, E_0, which represents the average amount of work a programmer can do without introducing an error. Letting $\hat{B}$ be the predicted number of module bugs, a working approximation to this function can be stated as[1]

$$\hat{B} = E/E_0$$

Working with this equation, however, did not lead to satisfactory results. Depending on the amount of repetition in the code, it was hypothesized that a learning factor might need to be taken into account. To a large extent, the level of the language dictates the amount of repetition in

---

[1]Since bugs happen in discrete units, what we really mean is $\hat{B} = \text{round}(E/E_0)$. For simplicity, the round is assumed and does not appear in any of the equations.

the code; therefore it too should have an effect on the number of bugs. The level of a program, L, as defined in software science is inversely related to the amount of repetition. That is, a program of the highest level, 1, would have no repetition. Based on this, a second approximation is

$$\hat{B} = LE/E_0 \qquad (7a)$$

$$\text{or } \hat{B} = V/E_0 \qquad (7b)$$

In order to test this hypothesis on readily available data, an alternate formulation is needed. Using the basic software science relationships:

$E = VV^*/\lambda$	solving (5) for L and substituting into (6)
$E = V^2L/\lambda$	solving (4) for V^* and substituting
$\lambda E = V^2L$	using basic algebra
$\lambda EV = V^3L$	
$\lambda EV/L = V^3$	
$\lambda E^2 = V^3$	substituting (6)

$$V = \lambda^{1/3}E^{2/3} \qquad (8)$$

For all common programming languages λ has been found to be near 1 [Hal76]. Since the cube root of a number near 1 is even nearer to 1, $\lambda^{1/3}$ is ignored, at least in this first approximation to the hypothesis. Thus,

$$V \approx E^{2/3}.$$

Substituting for V in (7b):

$$\hat{B} \approx E^{2/3}/E_0. \qquad (9)$$

Running a least squares fit on Akiyama's data to determine coefficients for an equation of the form $B = E^{\lambda}/E_0$ resulted in $\lambda = .61$ and $E_0 = 3200$. The value for λ is very close to 2/3 and thus provides support for (9). The obvious next step is to determine if there is any evidence, perhaps based on psychological experiments, to warrant the assumption that programmers have an opportunity to make an error after making 3200 mental discriminations.

Approximately 20 years ago, Miller [Mil56] conceptualized the idea of a basic unit of information that the short term memory of the human brain could hold for immediate recall. He called these basic units 'chunks' and concluded that the human short term memory can hold approximately seven of them. More recently, however, Simon [Sim74] has shown the short term memory chunk capacity to be closer to five.

One can deduce that if a person holds 5 chunks of information in his short term memory for immediate recall, he can also operate on these same five chunks of information at any one time. Each time an operation is performed on the available information in the short term memory a result is

obtained. Thus the number of input and output operands, or η_2^M, for each of these operations should be 6. From this and the basic equations, the amount of work done and the volume of information processed in each of these operations can be determined.

Solving (4) for V and substituting into (6):

$$E = V^M/L^2.$$

Solving (5) for L and substituting:

$$E = (V^M)^3/\lambda^2. \tag{10}$$

Knowing η_2^M, we can determine V^M. That is

$$\begin{aligned} V^M &= (2+\eta_2^M)\log_2(2+\eta_2^M) \\ &= 8\log_2(8) \\ &= 24. \end{aligned}$$

Given the value of λ, one can determine the number of elementary mental discriminations in each operation from (10). The value of λ for English has been found to be approximately 2.16 [Hal76], and it is assumed that natural language is the language of the brain. Now substituting into (10), we get

$$E = 24^3/2.16^2 \approx 3000.$$

If the assumptions are correct, this implies that after every 3000 mental discriminations a result is produced. This result, whether correct or incorrect, is more than likely either used as an input for the next operation or is

output to the environment. If incorrect the error should become apparent. Thus, an opportunity for error occurs every 3000 mental discriminations.

Using $E_0 \approx 3000$, (9) becomes

$$\hat{B} = E^{2/3}/3000. \qquad (11)$$

Predictions for the number of bugs found from (11) are presented in Table 1 along with Akiyama's original data. The correlation between the predictions and the actual data is .99.

SMALLEST BUG-FREE MODULE

The next step is to test the hypothesis on another set of data. A technical report by Bell and Sullivan [BeS74] provides the needed data to try the model in a slightly different situation. Their research was concerned with discovering complexity measures of programs. One of their experiments consisted of dividing a set of algorithms from the Communications of the ACM into two groups: one group containing algorithms which had been found to contain an error, the other consisting of correct algorithms. They found that all the correct algorithms had a length as defined in software science of less than 237 and all the incorrect programs, with one exception, had a length greater than 284.

Module	No. Mental Discriminations (in millions)	No. Actual Bugs Found	Predicted No. Bugs
MA	170.3	102	102
MB	15.3	18	20
MC	322.6	146	156
MD	28.2	26	30
ME	100.2	71	71
MF	65.5	37	54
MG	6.5	16	11
MH	58.5	50	50
MX	135.9	80	88
Totals	903.0	546	582

TABLE 1

It would be interesting if the hypothesis could explain this phenomenon. Since equation (7b) requires rounding, by setting the right hand side of it equal to 1/2, the largest number which would round to zero, an estimate of V for the largest program which can be written with no errors is obtained. By substituting into (7b):

$$1/2 = V/3000$$

$$V = 3000 * 1/2$$

$$V = 1500$$

Now assuming $\eta_1 \approx \eta_2$,[2] (1) becomes

$$\hat{N} = \eta \log_2(\eta/2)$$

Using this, (2), and V=1500, an estimate for $\hat{N}$ can be obtained. Thus,

$$\hat{N} \approx 260.$$

This 260 is the same value that Bell and Sullivan had suggested as a possible cutoff point. Their recommendation was that perhaps after this point algorithms should be subjected to more scrupulous checking before acceptance for publication.

---

[2]The maximum error would occur if either $\eta_1 = 0$ or $\eta_2 = 0$. In this case, $\tilde{N} = \eta \log_2 \eta$. Assuming $\eta_1 \approx \eta_2$, the estimate becomes $\hat{N} = \eta \log_2 \eta - \eta$ giving a relative error of $1/\log_2 \eta$. Since the difference between η_1 and η_2 is never this large, the relative error introduced by this assumption is always less than $1/\log_2 \eta$.

Bell and Sullivan also presented data giving the means of the measurements they took on the various modules. They found a mean N of 161.9 for the correct programs and 515.4 for the programs with one error. Assuming the number of bugs in the programs with an error are uniformly distributed between 1/2 and 1 1/2, the mean number of bugs in the incorrect programs is 1. Likewise, if the number of bugs for the correct programs is uniformly distributed between 0 and 1/2, the mean is 1/4. Using these two data points, and the basic equations to obtain E from N, one should be able to solve the equation

$$B = E^{\lambda}/E_0$$

for the two unknowns λ and E_0.

Given N(correct) = 161.9 and again N(incorrect) = 515.4 and assuming $\eta_1 \approx \eta_2$, one gets η(correct) $\approx$ 38.1 and η(incorrect) $\approx$ 93.0. Solving for V, one gets V(correct) = 850 and V(incorrect) = 3370.

By solving (8) for E and using $\lambda = 1.21$ for Algol [Hal76], E(correct) = 22,529 and E(incorrect) = 177,849 is found. Now solving the two equations

$$1/4 = (22529)^{\lambda}/E_0 \text{ and}$$

$$1 = (177849)^{\lambda}/E_0,$$

one gets

$$\lambda = .871 \text{ and}$$

$$E_0 = 3331.$$

These values agree within 10% of the estimates for the same parameters obtained from Akiyama's data. Thus approximately the same basic equation can be obtained from a completely different approach.

EXTENDING THE HYPOTHESIS TO MODULARIZED PROGRAMS

From our earlier discussion, it follows that any large program can be written as a collection of M modules each of length ≈ 260, the length shown above to minimize the number of bugs in a module. Each module then has a probability of 1/4 of containing one delivered bug and of 3/4 of containing zero bugs. This gives an estimate for B of

$$\hat{B} = M/4,$$

and the probability that this program contains precisely i delivered bugs is

$$p(i) = \text{binomial}(i,M,1/4) = \binom{M}{i}\left(\frac{3}{4}\right)^{M-i}\left(\frac{1}{4}\right)^{i}.$$

Consequently, the a priori probability that k or fewer bugs exist, P_0, is

$$P_0 = p(i)$$

and P_k, the probability that, after k bugs have been found,

no more exist, is

$$P_k = Pr[x=k \mid x \geq k]$$

$$= \frac{Pr[x=k \wedge x \geq k]}{Pr[x \geq k]}$$

$$= \frac{Pr[x=k]}{Pr[x \geq k]}$$

$$= \frac{p(k)}{\sum_{i=k}^{\infty} p(i)}$$

Hence P_K is computable for all values of k, and therefore, the number of bugs needed to be found can be obtained for any level of confidence desired.

Using the model, it is also possible to determine the expected number of remaining bugs, B_K, when k bugs have already been found. That is

$$B_k = Pr[x=i \mid x \geq k](i-k)$$

$$= \frac{Pr[x=i \wedge x \geq k](i-k)}{Pr[x \geq k]}$$

$$= \frac{p(i)(i-k)}{\sum_{i=k}^{\infty} p(i)}$$

Table 2 below gives sample results obtained from this model for a program consisting of 40 modules. The expected number of bugs is $\hat{B}=M/4=40/4=10$.

Table 2: Sample calculations for a program consisting of 40 modules ($\hat{B}$=10.)

No. of Errors	A priori Pr. of k or Fewer Errors	Pr. of No Errors after k Have Been Found	Expected No. of Remaining Errors
k	P_0	P_K	$\hat{B}_K$
0	0.00001	0.00001	10.0
1	0.0001	0.0001	9.0001
2	0.0010	0.0009	8.0013
3	0.0047	0.0037	7.0083
4	0.0160	0.0114	6.0342
5	0.0433	0.0277	5.1038
6	0.0962	0.0553	4.2491
7	0.1820	0.0949	3.4980
8	0.2998	0.1441	2.8646
9	0.4395	0.1995	2.3469
10	0.5839	0.2576	1.9319
11	0.7151	0.3154	1.6021
12	0.8210	0.3711	1.3403
13	0.8968	0.4237	1.1313
14	0.9456	0.4727	0.9630
15	0.9738	0.5179	0.8262
16	0.9884	0.5595	0.7138
17	0.9953	0.5977	0.6203
18	0.9983	0.6327	0.5418
19	0.9994	0.6649	0.4752
20	0.9998	0.6945	0.4181
21	0.99995	0.7218	0.3688
22	0.99998	0.7470	0.3259
23	1.00000	0.7703	0.2882
24	1.00000	0.7919	0.2549
25	1.00000	0.8120	0.2254
26	1.00000	0.8307	0.1989
27	1.00000	0.8482	0.1752
28	1.00000	0.8645	0.1537
29	1.00000	0.8797	0.1343
30	1.00000	0.8941	0.1166
31	1.00000	0.9075	0.1004
32	1.00000	0.9202	0.0855
33	1.00000	0.9322	0.0718
34	1.00000	0.9435	0.0592
35	1.00000	0.9542	0.0475
36	1.00000	0.9643	0.0366
37	1.00000	0.9739	0.0265
38	1.00000	0.9832	0.0168
39	1.00000	1.0000	0.0
40	1.00000	1.0000	0.0

CONCLUSION

In this paper, we presented a simple model to predict the number of delivered bugs in a program module. Because of the difficulty in obtaining good, published data, only a limited amount was available on which to test the hypothesis. It does appear certain that the number of bugs in a program module is very strongly correlated with the number of mental discriminations needed to write the module as defined in software science. To a limited extent, we were able to extend this to show that the number of bugs in a module is predictable. Obviously, the hypothesis presented is simple and as such may be good only as a first approximation. Future research may indicate that more factors need to be accounted for in the model; however, because of its current simplicity, refinements should not be difficult to make. We feel enough evidence has been presented to warrant further research into this area.

The theory presented here gives both an estimate of the number of bugs most likely to be found during the debugging of a program module and a method of predicting the presence and expected number of undiscovered bugs in a 'debugged' system. The benefits of such predictions when designing software are obvious. Knowing an approximation to the number of bugs should give a better approximation to needed

debugging time thus aiding in setting up more realistic software development time cycles. Likewise this information should be useful in determining confidence in the reliability of the developed system. In addition, the indicated size for the maximal bug-free module might be a highly useful metric for structuring programs.

REFERENCES

[Aki71] Akiyama, F., "An Example of Software System Debugging," Proc. of IFIP Congress, 1971.

[BeS74] Bell, D. E. and J. E. Sullivan, "Further Investigations into the Complexity of Software," MITRE MTR-2874, Vol. II, June 30, 1974.

[Bul73] Bulut, Necdet, "Invariant Properties of Algorithms," Ph.D. Thesis, Purdue Universiy, Aug. 1973.

[BuH74] Bulut, Necdet, and M. H. Halstead, "Impurities Found in Algorithm Implementations," ACM SIGPLAN Notices, 9, 3, March 1974.

[BHB74] Bulut, Necdet, M. H. Halstead, and Rudolf Bayer, "The Experimental Verification of a Structural Property of Fortran Programs," Proc. ACM Annual Conference, San Diego, 1974.

[FuH76] Funami, Yasao, and M. H. Halstead, "A Software Physics Analysis of Akiyama's Debugging Data," Technical Report CSD-TR 144, Purdue University, May 1975. An expanded version appears in Proceedings MRI XXIV International Symposium: Software Engineering, Polytechnic Press, New York, 1976.

[GaH75] Gannon, J. D. and J. J. Horning, "The Impact of Language Design on the Production of Reliable Software," Proc. of 1975 International Conf. on Reliable Software, April 1975.

[Hal72a] Halstead, M. H., "Natural Laws Controlling Algorithmic Structure," ACM SIGPLAN Notices, 7, 2, Feb. 1972.

[Hal72b] Halstead, M. H., "A Theoretical Relationship Between Mental Work and Machine Language

Programming," CSD-TR 67, Purdue University, May 1972.

[Hal73a] Halstead, M. H., "An Experimental Determination of the 'Purity' of a Trivial Algorithm," ACM SIGME Performance Evaluation Review, 2, 1, March 1973.

[Hal73b] Halstead, M. H., "Language Level, A Missing Concept in Information Theory," ACM SIGME Performance Evaluation Review, 2, 1, March 1973.

[HaB73] Halstead, M. H. and Rudolf Bayer, "Algorithmic Dynamics," Proc. ACM Annual Conference, Atlanta, 1973.

[HaZ73] Halstead, M. H. and P. M. Zislis, "Experimental Verification of Two Theorems of Software Physics," CSD-TR 97, Purdue University, June 1973.

[Hal75] Halstead, M. H., "Toward a Theoretical Basis for Estimating Programming Efforts," Proc. ACM Annual Conference, Minneapolis, 1975.

[Hal76] Halstead, M. H, "The Essential Design Criterion for Computer Languages: Software Science," CSD-TR 191, Purdue University, 1976.

[Hal77] Halstead, M. H., Elements of Software Science, American Elsevier, 1977 (forthcoming).

[LiD76] Litecky, Charles R. and Gordon B. Davis, "A Study of Errors, Error-proneness and Error Diagnosis in Cobol," CACM, Vol. 19, No. 1, Jan. 1976.

[Mil56] Miller, G. A., "The Magical Number Seven, Plus or Minus Two," Psychological Review, (1956), 311-329.

[Sim74] Simon, Herbert A., "How Big is a Chunk?," Science, Feb. 1974, Vol. 183 (4124), pp. 482-488.