1980

A Software Science Analysis of COBOL Programs

V. Y. Shen

Herbert E. Dunsmore
Purdue University, dunsmore@cs.purdue.edu

Report Number:
80-348

A Software Science Analysis of COBOL Programs

V. Y. Shen
H. E. Dunsmore

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

CSD-TR-348

ABSTRACT

A Software Science COBOL analyzer has been developed by the Software Research Group at Purdue University. The analyzer, written in COBOL, counts operators and operands in the Data and Procedure Divisions. The results of some pilot studies with this analyzer are reported. Examining 16 programs written by experienced COBOL programmers, 237 programs written by students, and 11 professionally-produced programs, the analyzer produces operator and operand counts that satisfy Software Science requirements. In addition there is nearly perfect agreement between the Software Science time estimate and the actual time it took to construct the analyzer.

Keywords: Software Science, COBOL programming analysis, program length prediction, programming effort prediction, language level.

August 6, 1980

1. Introduction

Software Science is a research area concerned with measurable properties of computer programs [HALS77, FITZ78, HALS79]. The underlying hypothesis is that a computer program need not be considered solely as an art form or even as an example of logic; instead it can be treated as a structure which may be studied using the classical methods of natural science. This theory has been widely investigated by independent research groups and is gaining acceptance as a tool in Software Engineering (see, for example, [SMIT80] on the application of Software Science at IBM's Santa Teresa Laboratory). It is possible to use the formulas of Software Science to compare different programming languages, to estimate the time required to develop computer programs, and to make predictions about the number of errors that still remain in a delivered computer program. However, most of the applications reported to date used "procedure-oriented" languages such as FORTRAN or PL/1.

COBOL is perhaps the most widely used programming language in this country, but it has received little research attention [SAMM78]. This paper reports preliminary findings on programs written in COBOL. In section 2 we describe a Software Science COBOL analyzer that has been developed at Purdue University. We present some preliminary results in section 3 suggesting that COBOL programs may be subjected to Software Science analysis and that it will be

useful in explaining programming effort. In section 4 we discuss an important byproduct of this work and summarize our results to date.

2. Design of the COBOL Analyzer

In Software Science a computer program is considered to be a string of tokens, which are divided into the classes of operators and operands. Generally any symbol or keyword in a program that specifies an action of the computer is considered an operator, and any symbol used to represent data is considered an operand. Most punctuation marks are considered operators; most labels (e.g., paragraph names) are considered operands. An analyzer for Software Science metrics should generate the following counts after scanning a program:

η_1 = number of unique operators

η_2 = number of unique operands

N_1 = total occurrences of operators

N_2 = total occurrences of operands.

A one-pass lexical analysis is adequate to produce the above results since all variables in a COBOL program must be declared in the DATA division and all keywords are reserved words. We feel that it is reasonable to exclude certain COBOL keywords that are actually "noise" words since they do

not affect the function of the program in any way. A list of the rules used in counting is shown in the Appendix. One should note that Software Science metrics are "gross" measures, i.e., a minor variation of these rules normally has little effect on the final outcome [ELSH78; SMIT80].

The COBOL analyzer developed by the Software Research Group at Purdue University is written in ANSI standard COBOL. Although it is not the most efficient language for character manipulation and string comparisons, we have implemented our analyzer in COBOL with the hope that any installation that wishes to analyze its COBOL programs can compile and execute our analyzer. It ignores the IDENTIFICATION and ENVIRONMENT divisions since these parts are normally copied from program to program, thus requiring little programming effort. Our analyzer counts operators and operands in the DATA and PROCEDURE divisions separately. Finally, it combines the two divisions treating them as a single entity. For each program the analyzer generates three sets of η_1, η_2, N_1, and N_2: one for the DATA division, one for the PROCEDURE division, and one for the combination of both divisions.

Although there is an ANSI standard for COBOL ([ANSI74]), most installations still have local features and collating sequences. The analyzer has two parts: the "installation package" that sorts the list of keywords on the host machine, and the "analysis package" that uses the

sorted file of keywords to analyze COBOL programs. (Notice that our analyzer does not employ any installation-specific features, but needs to be aware of such verbs and terms in order to analyze programs that may use them.)

3. Preliminary Results

We have conducted some pilot studies in order to investigate the usefulness of our COBOL analyzer. We present below results from 16 programs written by experienced COBOL programmers, 237 programs constructed by students in a programming course, 11 production programs used by the U. S. Army, and even results concerning the construction of the analyzer itself. These results suggest that our COBOL analyzer produces operator and operand counts that satisfy Software Science requirements. Furthermore, we have one data point that shows nearly perfect agreement between the software science time estimate and the actual time to produce our analyzer.

The first set of programs we analyzed were some sample programs written by experienced programmers who were instructors in an introductory COBOL course at Purdue University. These programs were relatively short so that hand-checking of the results was possible. The lengths range from 142 to 1030 statements (including blank statements) with Software Science measure N ranging from 167 to 1800. The results of analyzing the 16 programs are shown

in Table 1.

Table 1. Analysis of Sample Programs

Division	N		$\hat{N}$		$(N-\hat{N})/N$		λ	
	mean	σ	mean	σ	mean	σ	mean	σ
DATA	303	244	372	283	-.21	.24	34.13	24.59
PROCEDURE	443	289	708	524	-.55	.41	3.66	1.41
COMBINATION	746	474	840	607	-.09	.22	2.07	.90

The column $\hat{N}$ represents the predicted length using the length equation of Software Science:

$$\hat{N}=\eta_1\log_2\eta_1 + \eta_2\log_2\eta_2$$

The "error" of the length equation, which is defined as $(N-\hat{N})/N$, is frequently used to indicate the appropriateness of other Software Science formulas to a program -- the lower the error, the more confident we are that Software Science estimates are valid. The sign of the error indicates whether $\hat{N}$ is an over- or under- estimate. Notice in Table 1 that $\hat{N}$ is consistently an over-estimate of the actual program lengths. Neither the DATA nor PROCEDURE Division alone yield a very acceptable length estimate. In this case the best estimate of program length is attained when both the DATA and PROCEDURE Divisions are employed. In all three cases the correlations between N and $\hat{N}$ are higher than 0.90.

Software Science postulates a "language level" (λ) which may be used to compare languages. It is stated that language level is constant for all programs and all programmers for a given language. However, our experience has suggested that language level is <u>not</u> constant. So we consider λ as a metric that characterizes a language, although we make no claim for its constancy. Mean λ's reported in [HALS77] include 2.16 for English prose, 1.53 for PL/1, 1.21 for Algol, and 1.14 for Fortran. The COBOL language level 2.07 based on the combination of DATA and PROCEDURE Divisions, thus, seems reasonable. The extremely high (34.13) λ for the DATA Division alone reflects the fact that COBOL provides for a compact representation of a good deal of information about type, size, structure, and initial values of individual and group data items.

Another COBOL program which is readily available as data to the analyzer is the analyzer itself. The results of the analysis are shown in Table 2.

Table 2. Analysis of the Analyzer

Division	N	$\hat{N}$	$(N-\hat{N})/N$	λ
DATA	1706	2333	-.37	75.43
PROCEDURE	3661	3518	.04	.77
COMBINATION	5367	4107	.23	.61

In this case both the PROCEDURE division and the COMBINATION give reasonable values for the error and λ. The DATA

division analysis gives a λ value that is extremely large as explained above. Since the DATA division is a significant part in any COBOL program and may require major programming effort, we think that it is reasonable to include it in Software Science studies. Thus the remainder of this paper includes results only of the analysis of the combination of DATA and PROCEDURE Divisions.

Although the COBOL course offered by the Department of Computer Sciences of Purdue University is an introductory course, its students all have experience in at least one other programming language (Fortran or Pascal). In the fall of 1979 we asked the students in this class to save source listings for us of selected programming assignments. There were 106 students in the class, but not every one was successful in completing each of the four selected assignments. Analysis of the results are shown in Table 3.

Table 3. Analysis of Student Programs

ASN#	#progs	N		$\hat{N}$		$(N-\hat{N})/N$		λ	
		mean	σ	mean	σ	mean	σ	mean	σ
2	67	157	50	195	60	-.26	.18	1.19	.38
4	59	657	125	898	132	-.38	.16	.91	.27
5	53	1089	157	1075	125	.01	.07	.73	.17
6	58	937	237	841	183	.09	.12	1.82	.73

In Table 3 ASN# refers to the assignment number. We were able to collect copies of assignments 2, 4, 5, and 6. The

column #progs represents the number of programs for each assignment that were completed successfully and made available to us. In general Table 3 illustrates that our Software Science analyzer worked well for these 237 student programs, especially for the more lengthy and more difficult assignments 5 and 6. For these two assignments the mean error is less than ten percent. Furthermore, the mean λ's seem reasonable (if not constant).

The COBOL analyzer has been installed at the Army Institute for Research in Management Information and Computer Science (AIRMICS) in Atlanta, Georgia. They have analyzed 11 production programs used by the Army with the results shown in Table 4.

Table 4. Analysis of AIRMICS Programs

PROG#	N	$\hat{N}$	$(N-\hat{N})/N$	λ
1	294	330	-.12	1.81
2	726	790	-.09	.79
3	1521	1701	-.12	1.25
4	3420	3787	-.11	.47
5	3643	3277	.10	.63
6	6410	5587	.13	.42
7	8732	7159	.18	.66
8	11078	7039	.36	.33
9	12358	9389	.24	.73
10	12772	10876	.15	.72
11	17846	10019	.44	.49

The AIRMICS production programs show that the length equation produces negative errors for small programs but positive errors for large programs. Thus, the Software Science length predictor equation seems to be overestimating N for small programs and underestimating N for larger programs. The same observation has been made on IBM production programs [SMIT80] which were written in PL/S. They made the observation that the range of program sizes for which the length equation works best is $2000 \leq N < 4000$. It is of interest to note that the program length predictions for the two AIRMICS programs in that range are quite good. At Purdue University the Software Research Group is examining the length equation for possible modification in light of these types of results.

Also, the correlation between N and λ is -.542 (significant at the .05 level) suggesting that "language level" is not constant (as already discussed above) and affected by the size of the program. In this case, larger N's are accompanied by smaller λ's. We are considering possible modification to the computation of λ so that it might be more of a constant "language level".

The first version of our COBOL analyzer was developed by a graduate student at Purdue. He was asked to keep a log of time spent working on the project. This first version has been input to the analyzer itself yielding the following counts:

$\eta_1 = 92$

$\eta_2 = 405$

$N_1 = 2901$

$N_2 = 2466$

(i.e., 92 separate operands and 2901 total occurrences of them, and 405 separate operators and 2466 separate occurrences of them). The programming time for a program may be determined using the following Software Science formulas [HALS77]:

$$V = (N_1+N_2) \log_2(eta_1+\eta_2)$$

$$\hat{L} = \frac{2}{\eta_1} * \frac{\eta_2}{N_2}$$

$$E = V / \hat{L}$$

$$T = E / (18*3600) \text{ (hours)}$$

Using these formulas the estimated programming time is 207.8 hours. Summing the times in our programmer's log, we found that he actually spent a total of 201.5 hours in analyzer development. The error in this case is only -0.03.

4. Discussion

A byproduct of the COBOL analysis effort is the decision to include counts on data declarations and input/output statements. It is appropriate to include them since they are a major portion of any COBOL program. We went back to our existing FORTRAN analyzer and added statements to count the data declarations and input/output functions. The new analyzer did not produce results that are significantly different from those produced by the older version. This is expected because there are not very many data declarations and input/output statements in FORTRAN programs. Thus for consistency we have decided to include declarations and input/output statements in all Software Science analyses. Researchers at IBM have independently made the same decision for their PL/S analyzer [SMIT80].

Our preliminary results suggest that COBOL programs may be subjected to Software Science analysis, and that the major formulas such as the length equation and language level will be useful in explaining programming effort. We have seen that predictions of $\hat{N}$ and $\hat{V}$ are reasonable, although more work remains to be done to make them useful for predicting effort in any language. Our one piece of anecdotal evidence concerning time was extraordinarily good. It remains to be seen if this can be replicated with other COBOL programs. We conclude that it appears reasonable to conduct more experiments using our COBOL analyzer to gain

additional confidence, particularly in the areas of programming time prediction and error prediction.

5. Acknowledgements

The analyzer discussed in this paper was written by Dan Reed and later revised by Dennis Cok. Their proficiency and dedication to this project are greatly appreciated by the authors. Gary Gabrielle, Steve Booth, and Dennis Cok were instrumental in gathering data to be analyzed. We express our appreciation to Ken Christensen, George Fitzsos, and Chuck Smith for several useful discussions concerning Software Science. Finally, we thank the other members of the Purdue University Software Research Group (especially Sam Conte, Steve Thebaut, and Scott Woodfield) for their many helpful comments during the performance of this research and the preparation of this paper.

6. References

[ANSI74] American National Standard programming language COBOL, ANSI X3.23, 1974.

[ELSH78] Elshoff, J. L. A study of the structural composition of PL/I programs. ACM SIGPLAN Notices 13, 6 (June 1978), 29-37.

[FITZ78] Fitzsimmons, A. and Love, T. A review and evaluation of software science. Computing Surveys 10, 1 (March 1978), 3-18.

[HALS77] Halstead, M. H. Elements of Software Science, Elsevier North-Holland, New York, N.Y., 1977.

[HALS79] Halstead, M. H. Advances in software science, Advances in Computers, 18, M. Yovits, ed. Academic Press, New York, N.Y., 1979.

[SAMM78] Sammet, J. The early history of COBOL. ACM SIGPLAN Notices 13, 8 (August 1978), 121-160.

[SMIT80] Smith, C. P. A software science analysis of IBM programming products, TR 03.081 IBM Santa Teresa Laboratory, San Jose, Calif. Jan. 1980.

7. Appendix

Rules for Software Science counts in COBOL

OPERATORS:

1. FD, BLOCK, VALUE, REDEFINES, JUSTIFIED, PICTURE, SIGN, and THRU (Data Division).

2. End-of-statement for every statement in the Data and Procedure Divisions.

3. All procedure division verbs such as ACCEPT, ADD, ALTER, CALL, CLOSE, COMPUTE, DISPLAY, DIVIDE, EXAMINE, GO TO, IF, THEN, ELSE, MOVE, MULTIPLY, OPEN, PERFORM, READ, STOP, SUBTRACT, USE, WRITE, FROM, ROUNDED, USING, REMAINDER, TALLYING, REPLACING, DEPENDING, THRU, UNTIL, and AT END.

4. All conditional operators used in IF statements.

5. STOP RUN (i.e., the end-of-program operator).

OPERANDS:

1. All filenames, datanames, and condition names referenced in the Procedure Division.

2. FILLER.

3. Constants, literals, and character strings if they appear more than once.

4. All paragraph and section names.

5. All constants used in the Procedure Division.

(Specifically, do not count as operands: level numbers, data names that are never referenced in the Procedure Division, and unique Picture and Value items.)