D5-61

N87-24900

Technical Report TR-1228 November 1982
NSG-5123
AFOSR-F49620-80-C-001

METRIC ANALYSIS AND DATA VALIDATION ACROSS FORTRAN PROJECTS *

Victor R. Basili, Richard W. Selby, Jr.
and Tsai-Yun Phillips

Department of Computer Science
University of Maryland
College Park, MD 20742

*Research supported in part by the National Aeronautics and Space Administration Grant NSG-5123 and the Air Force Office of Scientific Research Contract AFOSR-F49620-80-C-001 to the University of Maryland. Computer support provided in part by the facilities of NASA/Goddard Space Flight Center and the Computer Science Center at the University of Maryland.

ABSTRACT

The desire to predict the effort in developing or explain the quality of software has led to the proposal of several metrics in the literature. As a step toward validating these metrics, the Software Engineering Laboratory has analyzed the Software Science metrics, cyclomatic complexity and various standard program measures for their relation to 1) effort (including design through acceptance testing), 2) development errors (both discrete and weighted according to the amount of time to locate and fix) and 3) one another. The data investigated are collected from a production FORTRAN environment and examined across several projects at once, within individual projects and by individual programmers across projects, with three effort reporting accuracy checks demonstrating the need to validate a database. When the data come from individual programmers or certain validated projects, the metrics' correlations with actual effort seem to be strongest. For modules developed entirely by individual programmers, the validity ratios induce a statistically significant ordering of several of the metrics' correlations. When comparing the strongest correlations, neither Software Science's E metric, cyclomatic complexity nor source lines of code appears to relate convincingly better with effort than the others.

I. Introduction

Several metrics based on characteristics of the software product have appeared in the literature. These metrics attempt to predict the effort in developing or explain the quality of that software [11], [17], [19], [23]. Studies have applied them to data from various organizations to determine their validity and appropriateness [1], [13], [15]. However, the question of how well the various metrics really measure or predict effort or quality is still an issue in need of confirmation. Since development environments and types of software vary, individual studies within organizations are confounded by variations in the predictive powers of the metrics. Studies across different environments will be needed before this question can be answered with any degree of confidence.

Among the most popular metrics have been the Software Science metrics of Halstead [19] and the cyclomatic complexity metric of McCabe [23]. The Software Science E metric attempts to quantify the complexity of understanding an algorithm. Cyclomatic complexity has been applied to establish quality thresholds for programs. Whether these metrics relate to the concepts of effort and quality depends on how these factors are defined and measured. The definition of effort employed in this paper is the amount of time required to produce the software product (the number of man-hours programmers and managers spent from the beginning of functional design to the end of acceptance testing). One aspect of software quality is the number of errors

reported during the product's development, and this is the measure associated with quality for this study.

Regarding a metric evaluation, there are several issues that need to be addressed. How well do the various metrics predict or explain these measures of effort and quality? Does the correspondence increase with greater accuracy of effort and error reporting? How do these metrics compare in predictive power to simpler and more standard metrics, such as lines of source code or the number of executable statements? These questions deal with the external validation of the metrics. More fundamental questions exist dealing with the internal validation or consistency of the metrics. How well do the estimators defined actually relate to the Software Science metrics? How do the Software Science metrics, the cyclomatic complexity metric and the more traditional metrics relate to one another? In this paper, both sets of issues are addressed. The analysis examines whether the given family of metrics is internally consistent and attempts to determine how well these metrics really measure the quantities that they theoretically describe.

One goal of the Software Engineering Laboratory [6], [7], [8], [10], a joint venture between the University of Maryland, NASA/Goddard Space Flight Center and Computer Sciences Corporation, has been to provide an experimental database for examining these relationships and providing insights into the answering of such questions.

The software comprising the database is ground support software for satellites. The systems analyzed consist of 51,000 to 112,000 lines of FORTRAN source code and took between 6900 and 22,300 man-hours to develop over a period of 9 to 21 months. There are from 200 to 600 modules (e.g., subroutines) in each system and the staff size ranges from 8 to 23 people, including the support personnel. While anywhere from 10 to 61 percent of the source code is modified from previous projects, this analysis focuses on just the newly developed modules.

The next section discusses the data collection process and some of the potential problems involved. The third section defines the metrics and interprets the counting procedure used in their calculation. In the fourth section, the Software Science metrics are correlated with their estimators and related to more primitive program measures. Finally, the fifth section determines how well this collection of volume and complexity metrics corresponds to actual effort and developmental errors.

II. The Data

The Software Engineering Laboratory collects data that deal with many aspects of the development process and product. Among these data are the effort to design, code and test the various modules of the systems as well as the errors committed during their development. The collected data are analyzed to provide insights into software development and to study the effect of various factors on the process and product. Unlike the typical

controlled experiments where the projects tend to be smaller and the data collection process dominates the development process, the major concern here is the software development process, and the data collectors must affect minimal interference to the developers.

This creates potential problems with the validity of the data. For example, suppose we are interested in the effort expended on a particular module and one programmer forgets to turn in his weekly effort report. This can cause erroneous data for all modules the programmer may have worked on that week. Another problem is how does a programmer report time on the integration testing of three modules? Does he charge the time to the parent module of all three, even though that module may be just a small driver? That is clearly easier to do than to proportion the effort between all three modules he has worked on. Another issue is how to count errors. An error that is limited to one module is easy to assign. What about an error that required the analysis of ten modules to determine that it affects changes in three modules? Does the programmer associate one error with all ten modules, an error with just the three modules or one third of an error with each of the three?~ The larger the system

---

~ Efforts [18], [21] have attempted to make this assignment scheme more precise by the explanation: a "fault" is a specific manifestation in the source code of a programmer "error"; due to a misconception or document discrepancy, a programmer commits an "error" that can result in several "faults" in the program. With this interpretation, what are referred to as errors in this study should probably be called faults. In the interest of consistency with previous work and clarity, however, the term error will be used throughout the paper.

the more complicated the association. All this assumes that all the errors are reported. It is common for programmers not to report clerical errors because the time to fill out the error report form might take longer than the time to fix the error. These subtleties exist in most observation processes and must be addressed in a fashion that is consistent and appropriate for the environment.

The data discussed in this paper are extracted from several sources. Effort data were obtained from a Component Status Report that is filled out weekly by each programmer on the project. They report the time they spend on each module in the system partitioned into the phases of design, code and test, as well as any other time they spend on work related to the project, e.g., documentation, meetings, etc. A module is defined as any named object in the system; that is, a module is either a main procedure, block data, subroutine or function. The Resource Summary Form, filled out weekly by the project management, represents accounting data and records all time charged to the project for the various personnel, but does not break effort down on a module basis. Both of these effort reports are utilized in Section V of this paper to validate the effort reporting on the modules. The errors are collected from the Change Report Forms that are completed by a programmer each time a change is made to the system. While the collection of effort and error data is a subjective process and done manually, the remainder of the software measures are objective and their calculation is

automated.

A static code analyzing program called SAP [25] automatically computes several of the metrics examined in this analysis. On a module basis, the SAP program determines the number of source and executable statements, the cyclomatic complexity, the primitive Software Science metrics and various other volume and complexity related measures. Computer Sciences Corporation developed SAP specifically for the Software Engineering Laboratory and the program has been recently updated [14] to incorporate a more consistent and thorough counting scheme of the Software Science parameters. In an earlier study, Basili and Phillips [3] employed the preliminary version of SAP in a related analysis. The next section explains the revised counting procedure and defines the various metrics.

III. Metric Definition

In the application of each of the metrics, there exist various ways to count each of the entities. This section interprets the counting procedure used by the updated version of SAP and defines each of the metrics examined in the analysis. These definitions are given relative to the FORTRAN language, since that is the language used in all the projects studied here. The counting scheme depends on the syntactic analysis performed by SAP and is, therefore, not necessarily chosen to coincide exactly with other definitions of the various counts.

Primitive Software Science metrics Software Science defines the vocabulary metric n as the sum of the number of unique operators n1 and the number of unique operands n2. The operators fall into three classes.

i) Basic operators include

```
+  -  *  /  **  =  ()  &  //  .NE.  .EQ.  .LE.  .LT.
.GE.  .GT.  .AND.  .OR.  .XOR.  .NOT.  .EQV.  .NEQV.
```

ii) Keyword operators include

```
IF() THEN                    /* logical if */
IF() THEN ELSE               /* logical if-then-else */
IF()  , ,                    /* arithmetic if */
IF() THEN ENDIF              /* block if */
IF() THEN ELSE ENDIF         /* block if-then-else */
IF() THEN
   ELSEIF() THEN
    ...   ENDIF              /* case if */
DO                           /* do loop */
DOWHILE                      /* while loop */
GOTO <target>                /* unconditional goto: distinct
                                targets imply different operators */
GOTO (T1...Tn) <expr>        /* computed goto: different number of
                                targets imply different operators */
GOTO <ident>, (T1...Tn)      /* assigned goto: distinct identifiers
                                imply different operators */
<subr>( , ,*<target>)        /* alternate return */
END=                         /* read/write option */
ERR=                         /* read/write option */
ASSIGNTO                     /* target assignment */
EOS                          /* implicit statement delimiter */
```

iii) Special operators consist of the names of subroutines, functions and entry points.

Operands consist of the all variable names and constants. Note that the major differences of this counting scheme from that used by Basili and Phillips [3] are in the way goto and if statements are counted.

The metric n* represents the potential vocabulary, and Software Science defines it as the sum of the minimum number of

operators n1* and the minimum number of operands n2*. The potential operator count n1* is equal to two; that is, n1* equals one grouping operator plus one subroutine/function designator. In this paper, the potential operand count n2* is equal to the sum of the number of variables referenced from common blocks, the number of formal parameters in the subroutine and the number of additional arguments in entry points.

Source lines This is the total number of source lines that appear in the module, including comments and any data statements while excluding blank lines.

Source lines - comments This is the difference between the number of source lines and the number of comment lines.

Executable statements This is the number of FORTRAN executable statements that appear in the program.

Cyclomatic complexity Cyclomatic complexity is defined as being the number of partitions of the space in a module's control-flow graph. For programs with unique entry and exit nodes, this metric is equivalent to one plus the number of decisions and in this work, is equal to the one plus sum of the following constructs: logical if's, if-then-else's, block-if's, block if-then-else's, do loops, while loops, AND's, OR's, XOR's, EQV's, NEQV's, twice the number of arithmetic if's, n - 1 decision counts for a computed goto with n statement labels and n

decision counts for a case if with n predicates.

A variation on this definition excludes the counts of AND's, OR's, XOR's, EQV's and NEQV's (later referred to as Cyclo_cmplx_2).

Calls This is the number of subroutine and function invocations in the module.

Calls and jumps This is the total number of calls and decisions as they are defined above.

Revisions This is the number of versions of the module that are generated in the program library.

Changes This is the total number of changes to the system that affected this module. Changes are classified into the following types (a single change can be of more than one type):

a. error correction
b. planned enhancement
c. implement requirements change
d. improve clarity
e. improve user service
f. debug statement insertion/deletion
g. optimization
h. adapt to environment change
i. other

Weighted changes This is a measure of the total amount of effort spent making changes to the module. A programmer reports the amount of effort to actually implement a given change by

indicating either

a. less than one hour,
b. one hour to a day,
c. one day to three days or
d. over three days.

The respective means of these durations, 0.5, 4.5, 16 and 32 hours, are divided equally among all modules affected by the change. The sum of these effort portions over all changes involving a given module defines the weighted changes for the module.

Errors This is the total number of errors reported by programmers; i.e., the number of system changes that listed this module as involved in an error correction. (See the footnote at the bottom of page 4 regarding the usage of the term "error".)

Weighted errors This is a measure of the total amount of effort spent isolating and fixing errors in a module. For error corrections, a programmer also reports the amount of effort spent isolating the error by indicating either

a. less than one hour,
b. one hour to one day,
c. more than one day or
d. never found.

The representative amounts of time for these durations, 0.5, 4.5, 16 and 32 hours, are combined with the effort to implement the correction (as calculated earlier) and divided equally among the modules changed. The sum of these effort portions over all error corrections involving a given module defines the weighted errors for the module.

IV. Internal Validation of the Software Science Metrics

The purpose of this section is to briefly define the Software Science metrics, to see how these metrics relate to standard program measures and to determine if the metrics are internally consistent. That is, Software Science hypothesizes that certain estimators of the basic parameters, such as program length N and program level L, can be approximated by formulas written totally in terms of the number of unique operators and operands. Initially, an attempt is made to find correlations between various definitions of these quantities based on the interpretations of operators and operands given in the previous section. Then, the family of metrics that Software Science proposes is correlated with traditional measures of software.

Program length Program length N is defined as the sum of the total number of operators N1 and the total number of operands N2; i.e., N = N1 + N2. Software Science hypothesizes that this can be approximated by an estimator $\hat{N}$ that is a function of the vocabulary, defined as

$$\hat{N} = n1\log2(n1) + n2\log2(n2).$$

The scatter plot appearing in Figure 1 and Pearson correlation coefficient of .899 ($p < .001$; 1794 modules)~ show the relationship between N and $\hat{N}$ (polynomial regression rejects including a second degree term at $p = .05$). Several sources [12], [16], [26], [27] have observed that the length estimator tends to be

~ The symbol p will be used to stand for significance level.

high for small programs and low for large programs. The correlations and significance levels for the pairwise Wilcoxon statistic [20], broken down by executable statements and length, are displayed in Table 1. In our environment, either measure of size demonstrates that N^ significantly overestimates N in the first and second quartiles and underestimates it (most significantly) in the fourth quartile. Feuer and Fowlkes [15] assert that the accuracy of the relation between the natural logarithms of estimated and observed length changes less with program size. The scatter plot appearing in Figure 2 and correlation coefficient for ln N vs. ln N^ of .927 ($p < .001$; 1794 modules) show moderate improvement.

<< Figure 1 >>

Table 1. Observed vs. estimated length broken down by program size.

a. N vs. N^ broken down by executable statments.

XQT STMTS	MODS	R-	ESTIMATION	WILCOXON SIGNIF
0 - 19	446	.601	over	<<.0001
20 - 40	442	.511	over	<<.0001
41 - 78	457	.478	under	.0367
79 <=	449	.751	under	<<.0001

b. N vs. N^ broken down by N.

Length N	MODS	R-	ESTIMATION	WILCOXON SIGNIF
0 - 114	449	.750	over	<<.0001
115 - 243	445	.447	over	<<.0001
244 - 512	453	.348	under	.0010
513 <=	447	.731	under	<<.0001

- ($p < .001$)

<< Figure 2 >>

Program volume A program volume metric V defined as N log2 n represents the size of an implementation, which can be thought of as the number of bits necessary to express it. The potential volume V* of an algorithm reflects the minimum representation of that algorithm in a language where the required operation is already defined or implemented. The parameter V* is a function of the number of input and output arguments of the algorithm and is meant to be a measure of its specification. The metric V* is defined as

$$V^* = (2 + n2^*) \log2 (2 + n2^*).$$

The correlation coefficient for V vs. V* of .670 ($p < .001$; 1794 modules) shows a reasonable relationship between a program's necessary volume and its specification.

Program level The program level L for an algorithm is defined as the ratio of its potential volume to the size of its implementation, expressed as

$$L = V^*/V.$$

Thus, the highest level for an algorithm is its program specification and there L has value unity. The larger the size of the required implementation V, the lower the program level of the implementation. Since L requires the calculation of V*, which is not always readily obtainable, Software Science hypothesizes that L can be approximated by

$$\hat{L} = \frac{2}{n1} \cdot \frac{n2}{N2}.$$

The correlation for L vs. L^ of .531 ($p < .001$; 1794 modules) is disappointingly below that of .90 given in [19]. Hoping for an increase in the correlations, the modules are partitioned by the number of executable statements in Table 2. Although the upper quartiles show measured improvement over the correlation of the whole sample, a more interesting relationship surfaces. The level estimator significantly underestimates the program level in the second, third and fourth quartiles, with the hypothesis being rejected in the first quartile. The increase in magnitude of the n2* parameter does not appear to be totally captured by the definition of L^.

Table 2. Relationship of observed vs. estimated program level broken down by program size.

XQT STMTS	MODS	R~	ESTIMATION	WILCOXON SIGNIF
0 - 19	446	.484	--	--
20 - 40	442	.672	under	<<.0001
41 - 78	457	.597	under	<<.0001
79 <=	449	.615	under	<<.0001
all	1794	.531	under	<<.0001

~ ($p < .001$)

Program difficulty The program difficulty D is defined as the difficulty of coding an algorithm. The metric D and the program level L have an inverse relationship; D is expressed

$$D = 1/L .$$

An alternate interpretation of difficulty defines it as the inverse of L^, given by

$$D2 = \frac{1}{L\hat{}} = \frac{n1N2}{2\ n2} \quad .$$

Christensen, Fitsos and Smith [12] demonstrate that the unique operator count n1 tends to remain relatively constant with respect to length for 490 PL/S programs. They propose that the average operand usage N2/n2 is the main contributor to the program difficulty D2. The scatter plot appearing in Figure 3 and Pearson correlation coefficient of .729 ($p < .001$; 1794 modules) display the relationship between N2/n2 and D2 for our FORTRAN modules. The application of polynomial regression brings in a second degree term ($p < .001$) and results in a correlation of .738.

<< Figure 3 >>

However, after observing in Figure 4 that n1 varies with program size, it seems as if the n1's inflation might possibly better explain D2. The scatter plot appearing in Figure 5 and the correlation of .865 ($p < .001$; 1794 modules) show the relationship of D2 vs. n1. Step-wise polynomial regression brings in a second degree term initially, followed by a linear term ($p < .001$), and results in a correlation of .879. In our environment, the unique operator count n1 explains a greater proportion of the variance of the difficulty D2 than the average operand usage N2/n2.

<< Figure 4 >>

<< Figure 5 >>

Program effort The Software Science effort metric E attempts to quantify the effort required to comprehend the implementation of an algorithm. It is defined as the ratio of the volume of an implementation to its level, expressed as

$$E = \frac{V}{L} = \frac{(V)^{**}2}{V^*} .$$

The E metric increases for programs implemented with large volumes or written at low program levels; that is, it varies with the square of the volume. An approximation to E can be obtained without the knowledge of the potential volume by substituting $L^\wedge$ for L in the above equation. The metric

$$E^\wedge = \frac{V}{L^\wedge} = \frac{n1\ N2\ V}{2\ n2} = \frac{n1\ N2\ N\ \log2\ n}{2\ n2}$$

defines the product of one half the number of unique operators, the average operand usage and the volume. In an attempt to remove the effect of possible program impurities [9], [19], $N^\wedge$ is substituted for N in the above equation, yielding

$$E^{\wedge\wedge} = \frac{N^\wedge \log2\ n}{L^\wedge} = \frac{n1\ N2\ (n1\log2n1 + n2\log2n2)\ \log2\ n}{2\ n2} .$$

The correlation coefficients for E vs. $E^\wedge$, E vs. $E^{\wedge\wedge}$, ln E vs. ln $E^\wedge$ and ln E vs. ln $E^{\wedge\wedge}$ are given in Table 3a. A fit of a least squares regression line to the log-log plot of E vs. $E^\wedge$ produces

the equation

$$\ln E = .830 * \ln E\hat{} + 1.357 .$$

Equivalently,

$$E = \exp(1.357) * (E\hat{})^{**}0.830 .$$

Due to this non-linear relationship and the improved correlation of ln E vs. ln E^, the modules are partitioned by executable statements in Table 3b. The application of polynomial regression confirms this non-linearity by bringing in a second degree term ($p < .001$), resulting in a correlation of .698. In Table 3b, notice that the correlations seem substantially better for modules below median size. The significant overestimation in the upper three quartiles attributes to the relationship of L and L^ described earlier.

Table 3. Observed vs. estimated Software Science E metric.

a. Pearson Correlation ($p < .001$; 1794 modules).

	R
E vs. E^	.663
ln E vs. ln E^	.931
E vs. E^^	.603
ln E vs. ln E^^	.890

b. E vs. E^ broken down by executable statements.

XQT STMTS	MODS	R~	ESTIMATION	WILCOXON SIGNIF
0 - 19	446	.708	under	.0050
20 - 40	442	.709	over	<<.0001
41 - 78	457	.411	over	<<.0001
79 <=	449	.550	over	<<.0001

~ ($p < .001$)

Program bugs Software Science defines the bugs metric B as the total number of "delivered" bugs in a given implementation. Not to be confused with user acceptance testing, the metric B is

the number of inherent errors in a system component at the completion of a distinct phase in its development. Bugs B is expressed by

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

where Eo is theoretically equivalent to the mean number of elementary discriminations between potential errors in programming. Through a calculation that employs the definitions of E, L and lambda (lambda = LV* is referred to as the language level), this equation becomes

$$B = \frac{(\text{lambda})^{**}1/3 \ (E)^{**}2/3}{Eo} .$$

The derivation determines an Eo value of 3000, assumes (lambda)**1/3 ~= 1 and obtains

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

The correlation for B vs. B^ is .789 ($p < .001$; 1794 modules).

In summary, the relationship of some of the Software Science metrics with their estimators seems to be program size dependent. Several observations lead to the result that the metric N^ significantly overestimates N for modules below the median size and underestimates for those above the median size. The level estimator L^ seems to have a moderate correlation with L, and its sig-

nificant underestimation of L in the upper three quartiles reflects its failure to capture the magnitude of n2* in the larger modules. With respect to the E metric, the effort estimator E^ correlates better over the whole sample than E^^, and their strongest correlations are for modules below median size. The estimator E^ shows a non-linear relationship to the effort metric E. The correlation of ln E vs. ln E^ significantly improves over that of E vs. E^, with the E^ metric's overestimation of E for larger modules attributing to the role of L^ in its definition. With the above family of metrics, Software Science attempts to quantify size and complexity related concepts that have traditionally been described by a more fundamental set of measures.

Table 4 displays the correlations of the Software Science metrics with the classical program measures of source lines of code, cyclomatic complexity, etc. There are several observations worth noting. Length N and volume V have remarkably similar correlations and correspond quite well with most of the program measures. Several of the metrics correlate well with the number of executable statements, especially the program "size" metrics of N1, N2, N and V (also B). The level estimator L^ and its inverse D2 seem to be much more related to the standard size and complexity measures than their counterparts L and D1. The language level lambda does not seem to show a significant relationship to the standard size and complexity measures, as expected. The E^^ metric relates best with the number of execut-

able statements and the modified cyclomatic complexity, while correlating with all the measures better than the E metric and slightly better than E^. None of the Software Science measures correlate especially well with the number of revisions or the sum

Table 4. Comparison of Software Science metrics against more traditional software measures.

Key: ? not significant at .05 level
* significant at .05 level
a significant at .01 level
otherwise significant at .001 level

	Source_Lines	Source-Cmmts	Cyclo_cmplx_2		Calls_&_Jumps	
	Execut_Stmts	Cyclo_cmplx	Revisions		Calls	
n1	.776	.854	.778	.796	.818	.361	.802	.542
n2	.852	.867	.853	.767	.774	.430	.809	.614
N1	.824	.964	.868	.881	.889	.328	.869	.552
N2	.826	.949	.871	.858	.870	.355	.870	.597
n2*	.792	.691	.754	.635	.629	.501	.683	.541
N	.829	.961	.873	.874	.884	.343	.874	.577
N^	.864	.897	.864	.800	.811	.420	.836	.621
V -	.837	.962	.875	.873	.883	.343	.876	.584
V*	.776	.677	.734	.618	.611	.485	.664	.525
L	-.098	-.179	-.112	-.170	-.173	?	-.158	-.083
L^	-.383	-.411	-.394	-.389	-.396	-.216	-.386	-.250
D1=1/L	.067a	.244	.113	.178	.196	-.093	.134	?
D2=1/L^	.696	.872	.745	.816	.839	.269	.791	.478
N2/n2	.365	.544	.437	.508	.517	.106	.470	.241
Lambda	.136	?	.108	?	?	.134	?	.051*
E	.439	.629	.500	.535	.556	.106	.506	.282
E^	.663	.831	.711	.771	.797	.224	.748	.452
E^^	.738	.871	.760	.799	.829	.268	.788	.501
B -	.837	.962	.875	.873	.883	.343	.876	.584
B^	.546	.749	.610	.650	.670	.149	.620	.355

- B and V will have identical correlations since they are linear functions of one another.

of procedure and function calls. The primary measures of unique operators n1 and unique operands n2 correspond reasonably well overall with n2 being stronger with source lines and n1 stronger with the cyclomatic complexities. In the next section, an analysis attempts to determine the relationship that these parameters really have with the quantities that they theoretically describe.

V. External Validation of the Software Science and Related Metrics

The purpose of this section is to determine how well the Software Science metrics and various complexity measures relate to actual effort and errors encountered during the development of software in a commercial environment. These objective product metrics are compared against more primitive volume metrics, such as lines of source code. The reservoir of development data includes the monitoring of several projects and the analysis examines several projects at once, individual projects and individual programmers across projects. To remove the dependency of the distribution of the correlation coefficient on the actual measures of effort and errors, the nonparametric Spearman rank order correlation coefficients are examined in this section [22]. (The ability of a few data points to artificially inflate or deflate the Pearson product-moment correlation coefficient is well recognized.) The analysis first examines how well these measures correspond to the total effort spent in the development of software.

A. Metrics' Relation to Actual Effort

Initially, a correlation across seven projects of the Software Science E metric vs. actual effort, on a module by module basis using only those that are newly developed, produces the results in Table 5. The table also displays the correlations of some of the more standard volume metrics with actual effort. These disappointingly low correlations create a fear that there

Table 5. Spearman rank order correlations Rs with effort for all modules (731) from all projects.

Key:		
	?	not significant at .05 level
	*	significant at .05 level
	a	significant at .01 level
otherwise		significant at .001 level

E	.345
E^	.445
E^^	.488
Cyclo_cmplx	.463
Cyclo_cmplx_2	.467
Calls	.414
Calls_&_Jumps	.494
D1=1/L	.126
D2=1/L^	.417
Source_Lines	.522
Execut_Stmts	.456
Source-Cmmts	.460
V	.448
N	.434
eta1	.485
eta2	.461
B	.448
B^	.345
Revisions	.531
Changes	.469
Weighted_Chg	.468
Errors	.220
Weighted_Err	.226

may be some modules with poor effort reporting skewing the analysis. Since there is partial redundancy built into the effort data collection process, there exists hope of validating the effort data.

Validation of effort data The partial redundancy in the development monitoring process is that both managers and programmers submit effort data. Individual programmers record time spent on each module, partitioned by design, code, test and support phases, on a weekly basis with a Component Status Report (CSR). Managers record the amount of time every programmer spends working each week on the project they are supervising with a Resource Summary Form (RSF). Since the latter form possesses the enforcement associated with the distribution of financial resources, it is considered more accurate [24]. However, the Resource Summary Form does not break effort down by module, and thus a combination of the two forms has to be used.

Three different possible effort reporting validity checks are proposed. All employ the idea of selecting programmers that tend to be good effort reporters, and then using just the modules that only they worked on in the metric analysis. The three proposed effort reporting validity checks are:

a. $$Vm = \frac{\text{number of weekly CSR's submitted by programmer}}{\text{number of weeks programmer appears on RSF's}}$$

b. $Vt = \frac{\text{sum of all man-hours reported by programmer on all CSR's}}{\text{sum of all man-hours reported for programmer on all RSF's}}$

c. $Vi = 1 - \frac{\text{number of weeks programmer's CSR effort > RSF effort}}{\text{total number of weeks programmer active in project}}$

The first validity proposal attempts to capture the frequency of the programmer's effort reporting. It checks for missing data by ranking the programmers according to the ratio Vm of the number of Component Status Reports submitted over the number of weeks that the programmer appears on Resource Summary Forms. The second validity proposal attempts to capture the total percentage of effort reported by the programmer. This proposal ranks the programmers according to the ratio Vt formed by the sum of all the man-hours reported on Component Status Reports over the sum of all hours delegated to him on Resource Summary Forms.

Note that for a given week, the amount of time reported on a Component Status Report should be always less than or equal to the amount of time reported on the corresponding Resource Summary Form. This is not because the programmer fails to "cover" himself, but a consequence of the management's encouragement for programmers to realisticly allocate their time rather than to guess in an ad hoc manner. This observation defines a third validity proposal to attempt to capture the frequency of a programmer's reporting of inflated effort. This data check ranks

the programmers according to the quantity Vi equal to one minus the ratio of the number of weeks that CSR effort reported exceeded RSF effort over the total number of weeks that the programmer is active in the project.

Metrics' relation to validated effort data Of the given proposals, the systems development head of the institution where the software is being developed suggests that the first proposal, the missing data check, would be a good initial attempt to select modules with accurate effort reporting [24]. The missing data ratios Vm are defined for programmers on a project by project basis. Table 6 displays the effort correlations of the newly developed modules worked on by only programmers with Vm >= 90% from all projects, those with Vm >= 80% and for all newly developed modules. Most of the correlations of the modules included in the Vm >= 90% level seem to show improvement over those at the Vm >= 80% level. Although this is the desired effect and several of the Vm >= 90% correlations increase over the original values, a majority of the correlations with modules at the Vm >= 80% level are actually lower than their original coefficients. Since the effect of the ratio's screening of the data is inconsistent and the overall magnitudes of the correlations are low, the analysis now examines modules from different projects separately.

Table 6. Spearman rank order correlations Rs with effort for modules across seven projects with various validity levels.

Key: ? not significant at .05 level
* significant at .05 level
a significant at .01 level
otherwise significant at .001 level

	Validity ratio Vm (#mods)		
	all(731)	80%(398)	90%(215)
E	.345	.307	.357
E^	.445	.422	.467
E^^	.488	.480	.513
Cyclo_cmplx	.463	.457	.479
Cyclo_cmplx_2	.467	.454	.506
Calls	.414	.360	.402
Calls_&_Jumps	.494	.475	.479
D1=1/L	.126	.088*	?
D2=1/L^	.417	.371	.421
Source_Lines	.522	.519	.501
Execut_Stmts	.456	.429	.475
Source-Cmmts	.460	.420	.439
V	.448	.434	.475
N	.434	.416	.460
eta1	.485	.462	.493
eta2	.461	.467	.503
B	.448	.434	.475
B^	.345	.307	.357
Revisions	.531	.580	.565
Changes	.469	.495	.385
Weighted_Chg	.468	.521	.462
Errors	.220	.381	.205
Weighted_Err	.226	.382	.247

The Spearman correlations of the various metrics with effort for three of the individual projects appear in Table 7.

Table 7. Spearman rank order correlations Rs with effort for various validity rankings of modules from individual projects S1, S3 and S7.

Key: ? not significant at .05 level
* significant at .05 level
a significant at .01 level
otherwise significant at .001 level
z unavailable data

	Project						
	S1			S3-		S7--	
Validity ratio Vm	all	80%	90%	80%	90%	all	80%
#modules	79	29	20	132	81	127	49
E	.613	.647	.726	.469	.419	.285	.409a
E^	.665	.713	.746	.602	.585	.389	.569
E^^	.700	.747	.798	.638	.640	.430	.567
Cyclo_cmplx	.757	.774	.792	.583	.608	.463	.523
Cyclo_cmplx_2	.764	.785	.787	.609	.664	.491	.523
Calls	.681	.698	.818	.442	.492	.404	.485
Calls_&_Jumps	.776	.813	.822	.594	.619	.488	.569
D1=1/L	.262a	?	?	.156*	?	?	?
D2=1/L^	.625	.681	.745	.507	.442	.377	.499
Source_Lines	.686	.672	.729	.743	.734	.486	.499
Execut_Stmts	.688	.709	.781	.609	.594	.408	.515
Source-Cmmts	.670	.710	.778	.671	.654	.416	.471
V	.657	.692	.774	.627	.637	.377	.497
N	.653	.680	.755	.613	.619	.360	.484
eta1	.683	.740	.848	.553	.533	.439	.431
eta2	.667	.701	.747	.643	.698	.365	.445
B	.657	.692	.774	.627	.637	.377	.497
B^	.613	.643	.726	.469	.419	.285	.409a
Revisions	.677	.717	.804	.655	.632	.449	.510
Changes	.687	.645	.760	.672	.639	.238a	.380a
Weighted_Chg	.685	.629	.749	.673	.649	.238a	.256*
Errors	z	z	z	.644	.611	.253a	.438
Weighted_Err	z	z	z	.615	.605	.245a	.276*

- All modules in project S3 were developed by programmers with Vm >= 80%.

-- There exist fewer than a significant number of modules developed by programmers with Vm >= 90%.

Although the correlation coefficients vary considerably between and among the projects, the overall improvement in projects S1 and S3 is apparent. Almost every metric's correlation with development effort increases with the more reliable data in projects S1 and S7. When comparing the strongest correlations from the seven individual projects, neither Software Science's E metrics, cyclomatic complexity nor source lines of code relates convincingly better with effort than the others. Note that the estimators of the Software Science E metric, E^ and E^^, appear to show a stronger relationship to actual effort than E.

The validity screening process substantially improves the correlations for some projects, but not all. This observation points toward the existence of project dependent factors and interactions. In an attempt to minimize these intraproject effects, the analysis focuses on individual programmers across projects. Note that Basili and Hutchens [2] also suggest that programmer differences have a large effect on the results when many individuals contribute to a project.

The use of modules developed solely by individual programmers significantly reduces the number of available data points because of the team nature of commercial work. Fortunately, however, there are five programmers who totally developed at least fifteen modules each. The correlations for all modules developed by them and their values of the three proposed validity ratios are given in Table 8. The order of increasing correlation coefficients for a particular metric can be related to the order of

Table 8. Spearman rank order correlations Rs with effort for modules totally developed by five individual programmers.

Key: ? not significant at .05 level
* significant at .05 level
a significant at .01 level
otherwise significant at .001 level

	Programmer (#mods)				
	P1(31)	P2(17)	P3(21)	P4(24)	P5(15)
E	.593	?	?	.561a	?
E^	.718	.526*	.375*	.555a	.507*
E^^	.789	.570a	?	.539a	.511*
Cyclo_cmplx	.592	.469*	.521a	.565a	?
Cyclo_cmplx_2	.684	.583a	.481*	.546a	?
Calls	.622	.787	?	.669	?
Calls_&_Jumps	.701	.604a	.451*	.579a	?
D1=1/L	.314*	?	?	?	?
D2=1/L^	.713	.460*	?	.497a	.467*
Source_Lines	.863	.682	.605a	.624	?
Execut_Stmts	.747	.540*	.436*	.631	.534*
Source-Cmmts	.826	.576a	.530a	.612	.509*
V	.718	.540*	.453*	.579a	.451*
N	.676	.526*	.461*	.556a	.471*
eta1	.811	.575a	?	.536a	?
eta2	.765	.701	.527a	.597	?
B	.718	.540*	.453*	.579a	.451*
B^	.593	?	?	.561a	?
Revisions	.675	.523*	.777	.468*	?
Changes	.412*	.468*	.600a	?	?
Weighted_Chg	.428a	.527*	.502a	?	?
Errors	.386*	?	.668	?	.596a
Weighted_Err	.342*	?	.624	?	.545*
VALIDITY RATIOS (%)					
Vm	92.5	96.0	87.7	83.9	74.1
Vt	97.9	91.8	98.8	82.1	74.1
Vi	78.6	69.5	77.6	80.0	87.5
Ave. Vm,Vt	95.2	93.9	93.25	83.0	74.1
Ave. Vm,Vi	85.5	82.75	82.65	81.95	80.8

increasing values for a given validity ratio using the Spearman rank order correlation. The significance levels of these rank order correlations for several of the metrics appear in Table 9. The statistically significant correspondence between the programmers' validity ratios Vm and the correlation coefficients justifies the use of the ratio Vm in the earlier analysis; possible improvement is suggested if Vm were combined with either of the other two ratios.

Table 9. Significance levels for the Spearman rank order correlation between the programmer's validity ratios and the correlati coefficients for several of the metrics.

	Ratio					
Metric	Vm	Vt	Vi	Ave(Vm,Vt)	Ave(Vm,Vi)	Ave(Vt
E^^				.09	.09	
Cyclo_cmplx						.05
Cyclo_cmplx_2	.05			.02	.02	
Calls_&_Jumps	.05			.02	.02	
Source_Lines	.05			.02	.02	
Source-Cmmts				.09	.09	
V (B)				.09	.09	
eta2	.05			.02	.02	
Revisions		.001	.09⁻	.09	.09	

⁻ Negative correlation.

In summary, the strongest sets of correlations occur between the metrics and actual effort for certain validated projects and for modules totally developed by individual programmers. While relationships across all projects using both all modules and only validated modules produce only fair coefficients, the validation process shows patterns of improvement. Applying the validity

ratio screening to individual projects seems to filter out some of the project specific interactions while not affecting others, with the correlations improving accordingly. Two averages of the validity ratios (Vm with Vt and Vm with Vi) impose a ranking on the individual programmers that statistically agrees with an ordering of the improvement of several of the correlations. In all sectors of the analysis, the inclusion of L^ in the Software Science E metric in its estimators E^ and E^^ seems to improve the metric correlations with actual effort. The analysis now attempts to see how well these metrics relate to the number of errors encountered during the development of software.

B. Metric's Relation to Errors

This section attempts to determine the correspondence of the Software Science and related metrics both to the number of development errors and to the weighted sum of effort required to isolate and fix the errors. A correlation across all projects of the Software Science bugs metric B and some of the standard volume and complexity metrics with errors and weighted errors, using only newly developed modules, produces the results in Table 10. Most of the correlations are very weak, with the exception of system changes. These disappointingly low correlations attribute to the discrete nature of error reporting and that 340 of the 652 modules (52%) have zero reported errors. Even though these correlations show little or no correspondence, the following observations indicate potential improvement.

Table 10. Spearman rank order correlations Rs with errors and weighted-errors for all modules (652) from six projects.

Key: ? not significant at .05 level
* significant at .05 level
a significant at .01 level
otherwise significant at .001 level

	Errors	Weighted_err
E	.083*	.101a
E^	.151	.171
E^^	.163	.186
Cyclo_cmplx	.196	.205
Cyclo_cmplx_2	.189	.200
Calls	.220	.236
Calls_&_Jumps	.235	.248
D1=1/L	?	?
D2=1/L^	.124	.140
Source_Lines	.255	.265
Execut_Stmts	.177	.198
Source-Cmmts	.288	.298
V	.168	.186
N	.162	.180
eta1	.102a	.132
eta2	.181	.199
B	.168	.186
B^	.083*	.101a
Revisions	.375	.375
Changes	.677	.636
Weighted_Chg	.627	.677
Design_Eff	.219	.185
Code_Eff	.285	.316
Test_Eff	.149	.164
Tot_Effort	.324	.332

- Project S1 has no data to distinguish errors from changes.

Weiss [4], [5] conducted an extensive error analysis that involved three of the projects and employed enforcement of error reporting through programmer interviews and hand-checks. For two

of the more recent projects, independent validation and verification was performed. In addition, the on-site systems development head asserts that due to the maturity of the collection environment, the accuracy of the error reporting is more reliable for the more recent projects [24]. These developmental differences provide the motivation for an examination of the relationships on an individual project basis.

Table 11 displays the attributes of the projects and the correlations of all the metrics vs. errors and weighted errors for three of the individual projects. The correlations in S7, a project involved in the Weiss study, are fair but better than those of project S5 (not shown) that was developed at about the same time. Project S4 and S6 (also not shown) have very poor overall correlations and unreasonably low relationships of revisions with errors, which point to the effect of being early projects in the collection effort. The trend that the attributes produce is not very apparent, although chronology and error reporting enforcement do seem to have some effect. In another attempt to improve the correlations, the analysis applies the

Table 11. Spearman rank order correlations Rs with errors and weighted-errors for modules from three individual projects.

Key:	?	not significant at .05 level
	*	significant at .05 level
	a	significant at .01 level
	otherwise	significant at .001 level
	Err	errors
	W_err	weighted-errors

	Project (#mods)					
	S3(132)		S4(35)		S7(127)	
	Err	W_err	Err	W_err	Err	W_err
E	.401	.378	?	?	.397	.391
E^	.536	.482	?	?	.507	.503
E^^	.579	.522	?	?	.492	.505
Cyclo_cmplx	.542	.481	?	?	.393	.368
Cyclo_cmplx_2	.553	.489	?	?	.405	.400
Calls	.445	.432	.300*	.316*	.423	.419
Calls_&_Jumps	.566	.518	?	?	.432	.412
D1=1/L	?	?	?	?	.168*	.178*
D2=1/L^	.491	.426	?	?	.563	.559
Source_Lines	.648	.622	.339*	?	.490	.487
Execut_Stmts	.538	.505	?	?	.478	.465
Source-Cmmts	.599	.568	?	?	.501	.483
V	.541	.495	?	?	.461	.456
N	.526	.480	?	?	.457	.449
eta1	.550	.500	?	?	.488	.522
eta2	.541	.500	?	?	.348	.367
B	.541	.495	?	?	.461	.456
B^	.401	.378	?	?	.396	.390
Revisions	.784	.694	.686	.630	.567	.500
Changes	.939	.864	.770	.761	.727	.670
Weighted_Chg	.840	.885	.661	.757	.624	.714
Design_Eff	?	?	?	?	?	?
Code_Eff	.620	.632	.413a	.398a	.274	.264
Test_Eff	.473	.481	.312*	?	?	?
Tot_Effort	.644	.615	.455a	.447a	.253a	.245a
PROJECT ATTRIBUTES						
Weiss study			X		X	
IV & V	X					
Chronology	recent		early		middle	

previous section's hypothesis of focusing on individual programmers. Table 12 gives the correlations of the metrics with errors and weighted errors for modules that two of the individual programmers totally developed. Even though it is encouraging to see

Table 12. Spearman rank order correlations Rs with errors and weighted-errors for modules totally developed by two individual programmers.

Key:	
?	not significant at .05 level
*	significant at .05 level
a	significant at .01 level
otherwise	significant at .001 level

Err	errors
W_err	weighted-errors

	Programmer (#mods)			
	P2(17)		P3(21)	
	Err	W_err	Err	W_err
E	.514*	.447*	.368*	?
E^	.527*	.493*	.600a	.563a
E^^	.515*	.473*	.666	.649
Cyclo_cmplx	.575a	.558a	.463*	.428*
Cyclo_cmplx_2	.661a	.616a	.484*	.449*
Calls	?	.498*	.506a	.469*
Calls_&_Jumps	.545*	.560a	.598a	.557a
D1=1/L	?	?	?	?
D2=1/L^	.558a	.526*	.459*	.429*
Source_Lines	?	?	.662	.646
Execut_Stmts	.624a	.577a	.579a	.533a
Source-Cmmts	?	.436*	.635	.594a
V	.491*	.472*	.679	.655
N	.494*	.479*	.641	.610a
eta1	.497*	.448*	.611a	.589a
eta2	?	?	.715	.717
B	.491*	.472*	.679	.655
B^	.514*	.447*	.368*	?
Revisions	?	?	.830	.811
Changes	.716	.662a	.855	.828
Weighted_Chg	?	.510*	.863	.861
Design_Eff	?	?	.460*	.392*
Code_Eff	?	.450*	.699	.667
Test_Eff	?	?	.668	.644
Tot_Effort	?	?	.668	.624

the correspondences of the metrics B, E^^ and eta2 with errors as among the best for programmer P3, the same metrics do not relate as well for other programmers.

In summary, partitioning an error analysis by individual project or programmer shows improved correlations with the various metrics. Strong relationships seem to depend on the individual programmer, while few high correlations show up on a project wide basis. The correlations for the projects reflect the positive effects of reporting enforcement and collection process maturity. Overall, the correlations with total errors are slightly higher than those with weighted errors, while the number of revisions appears to relate the best.

VI. Conclusions

In the Software Engineering Laboratory, the Software Science metrics, cyclomatic complexity and various traditional program measures have been analyzed for their relation to effort, development errors and one another. The major results of this investigation are the following: 1) None of the metrics examined seem to manifest a satisfactory explanation of effort spent developing software or the errors incurred during that process; 2) neither Software Science's E metric, cyclomatic complexity nor source lines of code relates convincingly better with effort than the others; 3) the strongest effort correlations are derived when modules obtained from individual programmers or certain validated projects are considered; 4) the majority of the effort correla-

tions increase with the more reliable data; 5) the number of revisions appears to correlate with development errors better than either Software Science's B metric, E metric, cyclomatic complexity or source lines of code; and 6) although some of the Software Science metrics have size dependent properties with their estimators, the metric family seems to possess reasonable internal consistency. These and the other results of this study contribute to the validation of software metrics proposed in the literature. The validation process must continue before metrics can be effectively used in the characterization and evaluation of software and in the prediction of its attributes.

Acknowledgment

The authors are grateful to F. McGarry and B. Curtis for their valuable comments on this analysis. We would also like to thank B. Decker, W. Taylor and E. Edwards for their assistance with the SAP program and the S.E.L. database.

Bibliography

[1] V. R. Basili, Tutorial on Models and Metrics for Software Management and Engineering, IEEE Comput. Society, IEEE Catalog No. EHO-167-7, 1980.

[2] V. R. Basili and D. H. Hutchens, "Analyzing a Syntactic Family of Complexity Metrics," Dept. Comput. Sci., Univ. of Maryland, College Park, MD 20742, Tech. Rep. TR-1053, Dec. 1981 (to appear in T.S.E.).

[3] V. R. Basili and T. Phillips, "Evaluating and Comparing the Software Metrics in the Software Engineering Laboratory," ACM Sigmetrics (1981 ACM Workshop/Symp. Measurement and Evaluation of Software Quality), Vol. 10, pp. 95-106, Mar. 1981.

[4] V. R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data*," Dept. Comput. Sci., Univ. of Maryland, College Park, MD 20742, Tech. Rep. TR-1235, Dec. 1982.

[5] V. R. Basili and D. M. Weiss, "Evaluating Software Development by Analysis of Changes: The Data from the Software Engineering Laboratory*," Dept. Comput. Sci., Univ. of Maryland, College Park, MD 20742, Tech. Rep. TR-1236, Dec. 1982.

[6] V. R. Basili and M. V. Zelkowitz, "Analyzing Medium Scale Software Developments," Proc. 3rd Int. Conf. Software Eng., Atlanta, GA, May 1978, pp. 116-123.

[7] V. R. Basili and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, Vol. 10, pp. 39-43, 1979.

[8] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, R. W. Reiter, Jr., W. F. Truszkowski and D. L. Weiss, "The Software Engineering Laboratory," Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD 20771, Rep. SEL-77-001, May 1977.

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

[10] D. N. Card, F. E. McGarry, J. Page, S. Eslinger and V. R. Basili, "The Software Engineering Laboratory," Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD 20771, Rep. SEL-81-104, Feb. 1982.

[11] E. T. Chen, "Program Complexity and Programmer Productivity," IEEE Trans. Software Eng., Vol. SE-4, pp. 187-194, May 1978.

[12] K. Christensen, G. P. Fitsos and C. P. Smith, "A Perspective on Software Science," IBM Syst. J., Vol. 20, pp. 372-387, 1981.

[13] B. Curtis, S. B. Sheppard and P. M. Milliman, "Third Time Charm: Stronger Replication of the Ability of Software Complexity Metrics to Predict Programmer Performance," Proc. 4th Int. Conf. Software Eng., Sept 1979, pp. 356-360.

[14] W. J. Decker and W. A. Taylor, "FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1)," Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD 20771, Rep. SEL-78-102, May 1982.

[15] A. R. Feuer and E. B. Fowlkes, "Some Results from an Empirical Study of Computer Software," Proc. 4th Int. Conf.

Software Eng., Sept. 1979, pp. 351-355.

[16] G. P. Fitsos, "Vocabulary Effects in Software Science," IBM Santa Teresa Lab., San Jose, CA 95150, Tech. Rep. TR 03.082, Jan. 1980.

[17] J. E. Gaffney and G. L. Heller, "Macro Variable Software Models for Application to Improved Software Development Management," Proc. of Workshop on Quantitative Software Models for Reliability, Complexity and Cost, IEEE Comput. Society, 1980.

[18] S. A. Gloss-Soler, The DACS Glossary: A Bibliography of Software Engineering Terms, Data & Analysis Center for Software, Griffiss Air Force Base, NY 13441, Rep. GLOS-1, Oct. 1979.

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

[20] R. V. Hogg and E. A. Tanis, Probability and Statistical Inference, MacMillian, New York, 1977, pp. 265-271.

[21] IEEE Standard Glossary of Software Engineering Terminology, IEEE, 342 E. 47th St., New York, Rep. IEEE-STD-729-1983, 1983.

[22] M. Kendall and A. Stuart, The Advanced Theory of Statistics, Vol. 2, 4th Ed., MacMillian, New York, 1979, pp. 503-508.

[23] T. J. McCabe, "A Complexity Measure," IEEE Trans. Software Eng., Vol. SE-2, pp. 308-320, Dec. 1976.

[24] F. E. McGarry, Systems Development Head, Code 582.1, NASA/ Goddard Space Flight Center, Greenbelt, MD 20771, personal consultation, Jan.-July 1982.

[25] E. M. O'Neill, S. R. Waligora and C. E. Goorevich, "FORTRAN Static Source Code Analyzer (SAP) User's Guide," Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD 20771, Rep. SEL-78-002, Feb. 1978.

[26] V. Y. Shen and H. E. Dunsmore, "A Software Science Analysis of COBOL Programs," Dept. Comput. Sci., Purdue Univ., West Lafayette, IN 47907, Tech. Rep. CSD-TR-348, August 1980.

[27] C. P. Smith, "A Software Science Analysis of IBM Programming Products," IBM Santa Teresa Lab., San Jose, CA 95150, Tech. Rep. TR 03.081, Jan. 1980.