ANALYZING HALSTEAD'S COUNTING RULES IN COBOL

by

MANA HUNG

B. S., National Chengchi University, 1970

A MASTER'S REPORT

submitted in partial fulfillment of the

requirement for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

David A. Gustafson

Major Professor

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

Chapter 1

INTRODUCTION

1.1 Software Complexity Measures

Significant and increasing costs of software development, testing and maintenance have provided motivation to study software characteristics which contribute to improved reliability, increased understanding, and ease of maintenance. Software measures/metrics have been defined, studied, and validated both experimentally and theoretically with the purpose of quantifying such characteristics as number of decisions, level of nesting, and the number of operators and operands.

Software measures are used to evaluate and predict various aspects of computer software and its development. One common use of software measures is to quantify the notion of complexity. The complexity of a program refers to the effort required to understand it. Complexity measures are often applied to the interaction between a program and a programmer working on some programming task. Measuring the complexity of computer programs can provide valuable information to aid in detecting potential program difficulties. Complexity measures can also assess programming techniques and the use of unstructed constructs. Either of these could compromise the final quality of a product. Measures can be helpful in avoiding unnecessary complexities and in achieving high quality software. A good measure should be algorithmic, direct and automatable. It also should be applicable to software written in a wide variety of program languages.

In the past years many complexity measures have been developed which assess different quality characteristics of software [Halstead 77, McCabe 76, and Henry 81]. Among these measures, one that has received much attention by researchers and has become a popular area of measures is Maurice Halstead's "software science." This theory is gaining acceptance in software engineering. Software science is the most comprehensive theory of the software development process yet attempted. It is claimed that software science may be used to compare different programming languages, to estimate the time required to develop computer program, and to make prediction about the errors that remain in a delivered computer program. Some of the attractiveness of software science is due to the simplicity of its instrumentation. An explanation of software science will be presented in the next section.

1.2 Theory of Software Science

Software science was developed during the 1970's by the late Professor Maurice Halstead at Purdue University. He proposed that as an experimental science, software science is concerned initially with those properties of programs that can be measured and with the relationship among those properties that remain invariant under translation from one language to another. He attempted to predict these measures at the most primitive level of programming when all that is known is the number of operators and operands.

Halstead proposed that complexity is closely related to program size. In this theory, software consists of an ordered string of operators and operands. They are mutually exclusive. When a program is translated from one language to another, e. g., from Pascal to machine language, the actual operators and operands may change but both versions still consist of a combination of operators and operands.

An operand is defined as a variable or constant. An operator is defined as any implicit or explicit symbol or group of symbols that can affect the values of operands or the order in which the values of an operand is changed. Examples of operands are keywords, delimiters, and arithmetic operators. From the identification of operators and operands a number of countable and measurable properties of any program can be defined. All software science measures are functions of the counts of operators and operands. Software science begins by defining four basic measures as the following:

n_1 = number of unique operators

n_2 = number of unique operands

N_1 = total occurrences of operators

N_2 = total occurrences of operands.

(actually, the Greek letter 'eta' is used in place of the 'n' symbol presented in this report.)

Based on these counts, the size of the vocabulary of a given program is the total number of unique operators and operands in that program. It is a measure of the repertoire of elements that a programmer must deal with to implement the program. Vocabulary is defined as:

$$n = n_1 + n_2.$$

The most fundamental and important relationship involves the length of the program. The length of the program, which is also related to the numbers of unique operators and unique operands, required for its implementation is defined as

$$N = N_1 + N_2.$$

Following the approach of information theory Halstead hypothesized that the length of the program can be estimated by the quantity $N_{est.}$ The estimated length is a function only of the number of unique operators and operands. The function, called the length equation, is denoted by N_{est} and is defined by

$$N_{est} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2.$$

He asssumed that the programs are well structured and better agreement between N and N_{est} should result when impurities are removed before operators and operands are counted. Halstead identified six classes of impurity. They are 1) complementary operations -- use of two complementary operators to the same operand, 2) ambigious operands -- use a given operand name to refer to different things at different places in the program, 3) synonymous operands -- the opposite of the above impurity, use two operands for the same thing, 4) common subexpressions -- fail to assign new name to the results of frequently used calculation, 5) unwarranted assignments -- an operand is assigned a value and used only once, and 6) unfactored express -- does not factor a factable expression. In the study of this report all impurities are ignored.

Based on the counts of operators and operands, it is possible to obtain quantitative measures for many useful properties of program, such as program volume, program level, program effort, programming time, and error rates.

1.3 Purpose for the Project

By surveying the published literature, one can see that almost every experiment on software science uses unique counting rules. Counting rules can change the magnitude of the measures. Hamer and Frewin stated [Hamer 82]:

> "The only limiting factor to widespread application of the measure is the unavailability of the basic counts of operators and operands. When these become accepted as standard output from compiler we can look forward to the general use of the measurement as a basic tool of analysts and programmers."

Lister also pointed out that since all software science measures are derived from counts of operators and operands, it is crucial that the counting rules be clearly defined and consistent across experiments [Lister 82]. At Purdue University, IBM, and General Motors, some research has been conducted on the effect of changing the counting rules [Christ 81, Elshoff 78, and Shen 81]. No universal agreement exists for exactly which tokens in a language are operators and which are operands. The differences of the counting rules make it difficult for researchers to compare the results of empirical studies conducted at different places or times. Researchers raised many questions concerning with the counting rules and length estimator. They suggest that program size may be a critical factor when considering the performance of the length estimator. Programs of different size seem to have different behavior. It is necessary to address the sensibility of the counting rule. Furthermore, it is interesting to see to what extent counting rules can be changed and how the changes affects the length estimator.

Although much research work in applying the methodology of software science to software measures has been done, most of it has concentrated on programs written in "scientifically oriented" or "procedure oriented" languages such as Fortran, PL/I, and Algol. With substantially different characteristics and application area, COBOL has received relatively little research attention with three notable exceptions [Zweben 79, Shen 81, and Debnath 84 & 85]. This paper primarily reports the investigation into different counting rules applied to a set of 45 professsionally produced COBOL programs.

1.4 Review of Current Research

Software science is a software complexity measure based upon a manageable number of major factors that affect programming. Experimental results provided by Halstead and other researchers have been very encouraging and have received considerable attention from the computer science community. With the rapidly growing interest in software science rules and counting tools, some researchers have raised serious questions about the underlying theory of software science. Meanwhile experimental evidence supporting some of the measures continue to be reported.

The original rules established by Halstead excluded the counting of declaration statements and input/output statements. Statement labels were considered a part of direct transfers. For example, in his experiment for Algol, GOTO statements such as GOTO label-1 and GOTO label-2 are considered as two different operators. Currently, most researchers tend to count the tokens in declaration and I/O statements. Meanwhile statement labels are mostly counted as operands whenever they appear. In such case GOTO label-1 and GOTO label-2 contain two occurrences of the one operator GOTO and one occurrence each of the two operands, label-1 and label-2. The classification of operators and operands is usually determined at the convenience of the programmer who is building the counting rules. As mentioned before there seems to be no agreement among researchers on what is the most meaningful way to classify and count these tokens.

In his study Elshoff developed 8 different counting rules to study the effect of variations of operator and operand counting methods on values calculated for the software science measures [Elshoff 78]. He intentionally perturbed the counting methods as much as possible in either direction. In one direction he expanded the count of unique operators as much as possible by splitting them very finely. On the other hand he reduced the number of unique operators to the greatest degree possible, combining all slightly similar operators into one. He then applied these rules to 34 PL/I programs, and found that when different counting methods were used some properties of the software such as length and volume remained stable, while others such as effort and level are not at all robust to slight variations of the rules in the classification of operators and operands. Although no methods was shown to be the best, the result implied the importance of the counting rule to the overall measure.

Conte et al. also concluded the same result that length and volume are quite robust for programs written in Fortran [Conte 82]. They used two analyzers which only had minor differences on the way the GOTO statement was counted. The number of unique operator n_1 decreased with the method which counted the GOTO statement as one operator and operand for the label. n_1 dropped dramatically if the program used a large number of GOTO statements and remained unchanged when no GOTO was used. In considering n_1 changes with program size they found that in a Fortran program of reasonable size, the number of unique operator n_1 is quite constant. The increase of n_1 as a program size grows is mainly due to the use of subroutine calls and function references.

Various studies of vocabulary relationships from software science have shown that for highly structured languages, the count of unique operator n_1 tends to remain fairly constant while the count of unique operand n_2 grows as the size of the program grows [Christ 82, Feuer 79, Fitso 80, and Lister 82]. Fitso has plotted n_1 and n_2 by program size for Assembler and PL/S languages. Although the fact that n_1 tends to be flat does not hold for Assembler language, it is true for the 490 PL/S programs he used. Because PL/S is a subset of PL/I, this may also be true for PL/I. Christensen et al. proposed that for structured languages, line of code, length and volume were linearly related and are equally valid as measures of program size. Since program size is a function of vocabulary n, and n_1 tends to be constant, program size is a function of the operands n_2. Fitso indicated that any one of the following factors will affect the number of unique operators in a language :

1) User-defined functions and procedures
2) The build-in-functions, procedures and operators
3) Number of branches, i.e. labels that are the target of a control transfer.

The use of Halstead's length equation to predict program length has been investigated by many researchers [Cook 82, Harrison 84, and Waguespack 87]. Several empirical studies tending to confirm that estimated program length is a good estimator of the actual program length have been reported [Wood 85 and Debnath 85]. On the other hand, some studies have raised questions about the accuracy of the estimator. From his experiment with Fortran, Basili found that the relationship of observed length with estimated length seems to be program size dependent; the estimator N_{est} tends to overestimate N for small programs and tend to underestimate N for large programs [Basili 83]. These results are essentially the same as those reported by some other researchers who used several different languages, such as PL/S and Pascal. At IBM, they also made the observation that the range of program sizes for which the length equation works best is $2000 \leq N < 4000$. The Software Metrics Research Group at Purdue University examined the length equation for possible modification in light of these types of results.

The recent survey on software economics has listed software size estimation as the first major issue needing further research. Several revised length estimators to predict program size by program vocabulary have been suggested in the literature [Jensen 85, and Livitin 87]. These new expressions were found to be much better approximation than Halstead's for the data used in each specific study. Whether, in general, the revised estimator are more accurate than the N_{est} provided by Halstead remains to be answered by further studies.

The principal limitation of the length equation as a tool for estimating program size lies in the fact that Halstead's estimator can be evaluated only after the program has been written. Early assessment of software quality, particularly in the design phase of software development, would provide designers and managers confidence of a quality end product. It is highly desirable to "use measurement that can lead to the optimization of program organization while the progrem is being written or while it is being designed. Measurement is an inherent part of the optimization process in other engineering disciplines. Software engineering definitely needs this kind of measurement discipline that each programmer can understand and can relate to choices made while designing and coding a program" [Christ 81].

Gustafson and White [Gustafson 83] have studied the possibility of applying Halstead's software science measures to one of the program design techniques, Warnier-Orr diagrams. In this study Warnier-Orr process operators as well as logical and arithmetic operators are counted as Halstead's operators, while numbers and noun phrases are counted as operands. Because of the small size of the experiment, conclusive results for estimated length cannot be obtained. It is reasonable and useful to do further research in this area in an effort to test the possibility of applying software science measures to the design phase.

Although most software measures have historically focused on code quality despite the importance of early and continuous quality evaluation in a software development effort, Szulewski et al. have applied software science to assess the quality of software design [Szule 83]. In order to compute software metrics prior to coding, the operators and operands in the design medium need to be identified and counted. This application has produced evidence that such measures can provide designers with useful feedback during system development.

1.5 Outline of Contents

Halstead's software science is based on the number of operators and operands. Chapter 2 will define the 6 sets of counting rules for COBOL language and give an overview of the test programs chosen. As the counting rules are defined they can be used to count the measures of software science. How to implement the automatic counting tool is then stated. Chapter 3 gives a detailed description of the way the statistical analysis is carried out from the output of the counting tools described in Chapter 2. Several statistical methods and packages were used. Chapter 4 provides the conclusions that have been reached. Also described are ideas for future work related to this project. The entire source code of the counting tool is listed in the Appendix A.

Chapter 2

DESCRIPTION OF THE METHODOLOGY

2.1 Counting Strategies

Software science measures proposed by Halstead are appealing. Calculation of the measures depends on the existence of well-defined counting strategies. The strategies require precise definitions of the components of a program : operators and operands. These definitions may influence the values of the measures.

Intuitively, it seems the task of establishing rules for classifying operators and operands would be easy. However, COBOL is such a rich and complex language, we are not surprised to find actually it is not a trivial work. An attempt was made to maintain as much consistency with procedures used for obtaining counts in other languages as possible. Our approaches are not based solely on intuition, but are guided by the language syntax requirement. Since COBOL is so flexible, we proposed several different ways to count operators and operands.

For every set of counting rules we ignored tokens which appear in Identification and Environment divisions. These two divisions do not affect the implementation of a program, consequently they do not require much programming effort. Only tokens in Data and Procedure divisions are considered. Traditionally, Halstead's software science measures do not include declarative statements in the operator and operand count. In most programming languages declarative statements are a major portion of a program and to a certain extent they determine the structure and complexity of the program.

Since all variables in a COBOL program must be declared in the Data division it is appropriate to include the Data division as well. Comment is an internal document; its presence or absence does not affect the function of the program hence it is ignored.

In our strategies basically we define an operator as any of the following:

1) an arithmetic operator which includes "+", "-", "*", "/", and "**";

2) a logical operator which include "AND", "NOT", and "OR";

3) a relational operator which includes "<", "=", and ">";

4) a delimiter: ",", ";", ".", "(", ")", and the quote """;

5) a current symbol "$";

6) a reserved word with a few exceptions.

A parenthesis pair ( ) is counted as a single operator, as is a quote pair " ". When encountering a "picture clause" in the Data division every character is counted as an operator, except any digit number enclosed by parenthesis ( ). For example, PICTURE S9(9)V99, every picture character symbol: S, 9, and V is counted as one occurrence of the operator, only the 9 inside ( ) is counted as an operand. Moreover, the "+", or "-" is counted as an unique operator no matter if it is a binary or unary sign.

Defining an operand is not as difficult as defining an operator. It is defined as a numerical literal, nonnumerical literal, figurative constants, or programmer-supplied word. In COBOL, some reserved words also are figurative constants, in this case they are classified as operands. These reserved words are ZERO(S), ZEROES, SPACE(S), QUOTE(S), HIGH-VALUE(S), and LOW-VALUE(S). There are 17 types of programmer-supplied words, examples of which are: data-name, file-name, record-name, and condition-name. Although a programmer-supplied word is counted as an operand, one type of programmer-supplied word, procedure name, commonly referred to as paragraph name in the Procedure division, is classified in two different ways: one as an operator and the other as an operand.

COBOL is characterized by great flexibility in the form of options available to the programmer. Consequently we use a syntax requirement, the COBOL language statement format [Spence 85] as guidance for determining the way to count the occurrences of operators and operands in the Data division and Procedure division. In a COBOL program certain reserved words are required in a statement while some are optional. The omission of these 'noise' optional words does not affect the function of the program. Three approaches are then developed with two extremes. Following is the explanation of the strategies.

1) Count all -- every occurring token is counted either as an operator or an operand.
2) Minimum count -- according to the format of COBOL statement, words which are underlined or/and which are enclosed in braces { } are required in the statement. Words which are not underlined and used to improve the readability of the program as well as words which appear inside brackets [ ] indicate that the words are optional. Under this approach the count only includes one of those required words and excludes the optional words.
3) Maximum count -- Both required and optional words are counted except the words that are not underlined.

For verbs that have different formats, we combined and reformated them into one new form. Also, if in a statement there are several required words, we chose one appropriate word and counted only that word. These two rules apply to counting approaches 2 and 3 described earlier.

2.2 Test Programs

After we defined different counting rules, a COBOL program(s) is needed to study the effect of variations in the counting rules on values of software measures. Two sets of COBOL programs were obtained from COBOL programming shops. These two sets contain a total of several hundred commercial COBOL programs. Many of these programs are either the same program with different versions or different program with the same application. To avoid bias, we kept away from the programs which were similar or which perform the same kind of function. If a program has different versions, we picked the latest version. The programs used for measuring were a set of 45 programs. These programs have been written by different professional programmers in different circumstances, and modified by persons other than the original programmer. They range in size from 70 lines of code to 2000 lines of code approximately.

2.3 Strategies for Implementation

A research should have an accurate and efficient tool available to collect data on product measures. We have devised the counting rules and selected a set of 45 appropriate programs written in COBOL to be test objects. In order to investigate Halstead's measures, an automated method of scanning each of these 45 programs to determine the counts of operator and operand as well as other measures were developed. The counting programs written in Pascal were used to count paragraph name by both operator and operand in each of the three ways given in the previous section. For convenience, when paragraph-name is counted as operand, we referred to the counting strategy with count all, with minimum count, and with maximum count as strategy 1, strategy 2, and strategy 3, respectively. Similarly, when paragraph-name is counted as operator, the strategy with count all, with minimum count, and with maximum count were referred to as strategy 4, strategy 5, and strategy 6, respectively. The Pascal language was chosen because of its string manipulation and dynamic abstract data structure.

The Pascal counting program (Appendix A) scanned the COBOL source code one token at a time. At a single pass of the source code, different values of measures were calculated. A table of token with its classification, whether it is counted, and occurrences in the Data division and Procedure division were produced. In addition, the number of distinct operators, the number of distinct operands, total occurrence of operators, and total occurrence of operands were calculated separately for the Data division, the Procedure division, and the whole program. The observed length of the program, N (the total occurrence of operators and operands in the program), and the estimated length obtained by using Halstead's length equation were also produced.

After having the above results for each of the 45 COBOL programs, a small C program was written to retrieve the values of n_1, n_2, N_1, N_2, N, and N_{est} from the results and output them into six separate files. For each of the six Pascal counting programs we repeated the same steps described above. A total of 270 files were created by running these 6 Pascal counting programs and a total of 36 files were created by running the C program. These 36 files were then copied from Unix[1] system to a Macintosh[2] PC where the empirical data can be processed and analyzed. Next these 36 files were combined into 6, one for each strategy, as shown in Tables 1-6. Table 7 shows the correspondence between the column heading in these tables and the software science measures.

1. Unix is a trademark of Bell Laboratories.
2. Macintosh is a trademark of Apple Computer, Inc.

Table 1. Result for Strategy 1

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S1	n2-S1	N1-S1	N2-S1	N-S1	EstN-S1	|RE|-S1	n1/n2-S1	n-S1	Est |RE|-S1	|RE| of |RE|-S1
2	87	166	1111	566	1677	1630.685	0.028	0.404	233	0.230	7.3140
3	53	70	312	199	511	732.630	0.434	0.757	123	0.375	0.1363
4	56	48	202	117	319	593.290	0.860	1.167	104	0.543	0.3690
5	70	77	440	223	663	911.592	0.375	0.909	147	0.437	0.1653
6	59	46	206	104	310	601.160	0.939	1.283	105	0.590	0.3717
7	55	138	599	349	948	1298.951	0.370	0.399	193	0.228	0.3854
8	43	25	116	59	175	349.426	0.997	1.720	68	0.770	0.2279
9	74	153	983	532	1515	1569.880	0.036	0.484	227	0.262	6.2450
10	57	272	1933	1502	3435	2532.265	0.263	0.210	329	0.150	0.4292
11	53	174	853	820	1473	1598.852	0.085	0.305	227	0.189	1.2156
12	38	164	572	461	1033	1406.060	0.361	0.232	202	0.159	0.5594
13	103	350	2702	1521	4223	3646.633	0.136	0.294	453	0.185	0.3538
14	83	160	792	414	1206	1700.637	0.410	0.519	243	0.277	0.3250
15	61	95	431	228	659	985.911	0.496	0.642	156	0.327	0.3399
16	87	124	756	400	1156	1422.856	0.231	0.702	211	0.352	0.5242
17	34	236	1587	691	2278	2033.277	0.107	0.144	270	0.123	0.1464
18	35	90	573	266	839	763.792	0.090	0.389	125	0.224	1.4941
19	52	39	197	114	311	502.554	0.816	1.333	91	0.611	0.0081
20	69	191	903	559	1462	1868.777	0.278	0.361	260	0.212	0.2372
21	38	23	119	52	171	303.463	0.775	1.652	81	0.742	0.0425
22	79	337	3026	1761	4787	3327.654	0.305	0.234	416	0.160	0.4744
23	94	716	4723	3121	7844	7406.543	0.056	0.131	810	0.118	1.1143
24	65	267	1519	809	2328	2543.660	0.093	0.243	332	0.164	0.7695
25	73	251	2188	1179	3367	2452.715	0.272	0.291	324	0.183	0.3248
26	64	432	1867	1295	3162	4168.111	0.318	0.148	496	0.125	0.6069
27	56	378	1100	891	1991	3561.740	0.789	0.148	434	0.125	0.8418
28	37	124	511	405	916	1055.070	0.152	0.298	161	0.186	0.2281
29	83	116	650	349	999	1324.654	0.326	0.716	199	0.358	0.0968
30	73	82	431	237	668	973.176	0.457	0.890	155	0.429	0.0605
31	58	196	998	591	1589	1832.246	0.153	0.296	254	0.185	0.2114
32	42	72	374	188	562	870.712	0.193	0.583	114	0.303	0.5681
33	33	13	79	31	110	214.571	0.951	2.538	46	1.105	0.1626
34	55	92	510	291	801	918.142	0.146	0.598	147	0.309	1.1148
35	86	345	2266	1296	3562	3461.165	0.028	0.249	431	0.166	4.8749
36	67	184	903	558	1461	1790.763	0.226	0.364	251	0.213	0.0545
37	79	252	1560	961	2521	2508.273	0.005	0.313	331	0.193	37.1607
38	93	202	1139	659	1798	2155.100	0.199	0.460	295	0.253	0.2734
39	44	27	165	73	238	388.597	0.549	1.630	71	0.732	0.3349
40	97	390	2580	1229	3809	3997.050	0.049	0.249	487	0.166	2.3640
41	69	102	641	299	940	1102.076	0.172	0.676	171	0.342	0.9808
42	70	100	598	363	961	1093.435	0.138	0.700	170	0.351	1.5483
43	41	89	514	273	787	796.000	0.011	0.461	130	0.253	21.1251
44	85	200	1415	729	2144	2073.569	0.033	0.425	285	0.238	6.2568
45	44	38	179	92	271	439.636	0.622	1.158	82	0.539	0.1338
46	101	858	4735	3018	7753	9033.547	0.165	0.118	959	0.112	0.3198
47											
48						MRE =	0.318				2.2865

Table 2. Result for Strategy 2

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S2	n2-S2	N1-S2	N2-S2	N-S2	EstN-S2	|RE|-S2	n1/n2-S2	n-S2	Est |RE|-S2	|RE| of |RE|-S2
2	51	188	932	588	1498	1513.550	0.010	0.307	217	0.2481	22.8983
3	42	70	271	199	470	855.527	0.395	0.800	112	0.3703	0.0620
4	41	48	185	117	282	487.738	0.730	0.854	88	0.4783	0.3471
5	55	77	314	223	537	800.517	0.491	0.714	132	0.4180	0.1483
6	47	46	179	104	283	515.150	0.820	1.022	83	0.5463	0.3341
7	41	138	511	349	860	1200.838	0.398	0.297	179	0.2438	0.3844
8	34	25	98	59	155	289.070	0.885	1.360	58	0.8874	0.2052
9	52	153	799	532	1331	1408.803	0.057	0.340	205	0.2817	3.5851
10	41	272	1705	1502	3207	2419.450	0.248	0.151	313	0.1828	0.2558
11	38	174	753	820	1373	1494.493	0.088	0.218	212	0.2110	1.3845
12	30	184	510	481	871	1353.845	0.394	0.183	194	0.1882	0.5024
13	75	350	2138	1521	3859	3425.085	0.084	0.214	425	0.2093	2.2738
14	80	180	581	414	995	1525.922	0.534	0.375	220	0.2784	0.4821
15	46	95	353	228	581	878.220	0.512	0.484	141	0.3219	0.3707
16	60	124	582	400	982	1218.734	0.239	0.484	184	0.3218	0.3462
17	22	236	1354	891	2045	1958.411	0.042	0.093	258	0.1588	2.7495
18	23	90	488	288	752	688.309	0.085	0.258	113	0.2285	1.6744
19	39	39	180	114	274	412.261	0.505	1.000	78	0.5372	0.0846
20	50	181	897	559	1258	1729.482	0.377	0.282	241	0.2281	0.3922
21	29	23	95	52	147	244.923	0.888	1.281	52	0.8481	0.0301
22	58	337	2643	1761	4404	3169.419	0.280	0.172	395	0.1917	0.3182
23	87	718	3837	3121	7058	7198.840	0.020	0.094	783	0.1589	7.0782
24	48	287	1349	809	2158	2420.284	0.122	0.180	315	0.1949	0.8035
25	55	251	1887	1179	3088	2318.832	0.244	0.219	308	0.2113	0.1329
26	48	432	1651	1295	2948	4050.190	0.375	0.111	480	0.1882	0.5585
27	38	378	998	891	1888	3435.849	0.819	0.101	418	0.1818	0.8024
28	31	124	484	405	889	1015.900	0.189	0.250	155	0.2242	0.3282
29	82	118	503	349	852	1164.886	0.387	0.534	178	0.3428	0.0856
30	55	82	317	237	554	839.294	0.515	0.871	137	0.3898	0.2237
31	45	198	813	591	1504	1739.817	0.157	0.230	241	0.2157	0.3787
32	33	72	305	188	493	810.700	0.239	0.458	105	0.3111	0.3032
33	25	13	83	31	94	184.202	0.747	1.923	38	0.9224	0.2351
34	43	92	421	291	712	833.487	0.171	0.487	135	0.3148	0.8455
35	84	345	1908	1298	3204	3292.508	0.028	0.188	409	0.1973	6.1418
36	52	184	748	558	1306	1880.758	0.287	0.283	238	0.2378	0.1713
37	81	252	1338	961	2287	2372.050	0.033	0.242	313	0.2209	5.7803
38	85	202	905	858	1584	1838.413	0.239	0.322	267	0.2541	0.0818
39	34	27	134	73	207	301.358	0.456	1.259	61	0.8454	0.4159
40	88	390	2045	1229	3274	3770.806	0.152	0.174	458	0.1928	0.2694
41	49	102	487	299	786	955.708	0.248	0.480	151	0.3203	0.2935
42	49	100	491	383	854	939.508	0.100	0.480	148	0.3244	2.2395
43	32	89	480	273	733	738.340	0.005	0.380	121	0.2899	58.2353
44	59	200	1092	729	1821	1875.847	0.030	0.295	259	0.2430	7.0670
45	35	38	148	92	240	378.848	0.579	0.821	73	0.5043	0.1290
46	70	858	4008	3018	7027	8790.117	0.251	0.082	928	0.1538	0.3866
47											
48						MRE =	0.314				2.9231

Table 3. Result for Strategy 3

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S3	n2-S3	N1-S3	N2-S3	N-S3	EstN-S3	|RE|-S3	n1/n2-S3	n-S3	Est |RE|-S3	|RE| of |RE|-S3
2	61	166	1087	566	1653	1586.032	0.041	0.367	227	0.2200	4.4304
3	49	70	304	199	503	704.171	0.400	0.700	119	0.3550	0.1123
4	51	48	194	117	311	557.372	0.792	1.063	99	0.5025	0.3657
5	65	77	418	223	641	873.996	0.363	0.844	142	0.4136	0.1379
6	53	46	194	104	298	557.664	0.871	1.152	99	0.5387	0.3817
7	48	138	585	349	934	1249.055	0.337	0.348	186	0.2120	0.3716
8	39	25	107	59	166	322.227	0.941	1.560	64	0.7044	0.2515
9	67	153	962	532	1494	1516.808	0.015	0.438	220	0.2486	15.2825
10	53	272	1923	1502	3425	2503.370	0.269	0.195	325	0.1498	0.4432
11	49	174	847	620	1467	1570.193	0.070	0.282	223	0.1851	1.6311
12	36	164	569	461	1030	1392.756	0.352	0.220	200	0.1599	0.5461
13	94	350	2647	1521	4168	3574.055	0.143	0.269	444	0.1798	0.2616
14	72	180	749	414	1163	1615.743	0.389	0.450	232	0.2535	0.3488
15	54	95	420	228	648	934.900	0.443	0.568	149	0.3016	0.3188
16	78	124	719	400	1119	1352.582	0.209	0.629	202	0.3262	0.5628
17	28	236	1580	691	2271	1994.910	0.122	0.119	264	0.1189	0.0221
18	29	90	564	266	830	725.148	0.126	0.322	119	0.2016	0.5957
19	46	39	183	114	297	460.215	0.550	1.179	85	0.5498	0.0005
20	65	191	887	559	1446	1838.743	0.272	0.340	256	0.2089	0.2308
21	33	23	107	52	159	270.507	0.701	1.435	56	0.6535	0.0681
22	72	337	3012	1761	4773	3273.890	0.314	0.214	409	0.1575	0.4986
23	84	716	4671	3121	7792	7327.367	0.060	0.117	800	0.1183	0.9846
24	58	267	1500	809	2309	2491.969	0.079	0.217	325	0.1589	1.0056
25	86	251	2176	1179	3355	2399.787	0.285	0.263	317	0.1775	0.3766
26	58	432	1854	1295	3149	4121.874	0.309	0.134	490	0.1252	0.5947
27	50	378	1091	891	1982	3518.720	0.775	0.132	428	0.1244	0.8395
28	36	124	509	405	914	1048.438	0.147	0.290	160	0.1886	0.2824
29	75	118	620	349	969	1262.887	0.303	0.847	191	0.3333	0.0998
30	67	82	413	237	650	927.747	0.427	0.817	149	0.4028	0.0578
31	52	196	986	591	1577	1788.906	0.134	0.265	248	0.1785	0.3281
32	37	72	362	188	550	636.984	0.158	0.514	109	0.2794	0.7669
33	28	13	70	31	101	182.712	0.809	2.154	41	0.9456	0.1689
34	50	92	493	291	784	882.361	0.125	0.543	142	0.2915	1.3231
35	80	345	2244	1296	3540	3414.260	0.036	0.232	425	0.1649	3.6420
36	61	184	891	558	1449	1748.110	0.205	0.332	245	0.2054	0.0015
37	71	252	1526	961	2487	2448.907	0.016	0.282	323	0.1851	10.4842
38	86	202	1113	859	1772	2099.617	0.185	0.426	288	0.2436	0.3178
39	40	27	158	73	231	341.259	0.477	1.481	67	0.6725	0.4089
40	89	390	2528	1229	3757	3933.199	0.047	0.228	479	0.1634	2.4838
41	64	102	627	299	926	1064.587	0.150	0.627	166	0.3256	1.1754
42	62	100	581	363	944	1033.546	0.095	0.620	162	0.3225	2.4003
43	37	89	507	273	780	769.090	0.014	0.416	126	0.2396	16.1275
44	77	200	1353	729	2082	2011.314	0.034	0.385	277	0.2271	5.6885
45	39	38	168	92	260	405.552	0.560	1.026	77	0.4876	0.1290
46	90	858	4658	3018	7676	8945.334	0.165	0.105	948	0.1133	0.3149
47											
48						MRE =	0.296				1.7081

Table 4. Result for Strategy 4

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S4	n2-S4	N1-S4	N2-S4	N-S4	EstN-S4	|RE|-S4	n1/n2-S4	n-S4	Est |RE|-S4	|RE| of |RE|-S4
2	64	149	1154	523	1677	1612.611	0.038	0.564	233	0.242	5.309
3	58	65	333	178	511	731.217	0.431	0.892	123	0.358	0.170
4	60	44	212	107	319	594.628	0.864	1.364	104	0.523	0.394
5	72	75	445	218	663	911.396	0.375	0.960	147	0.381	0.018
6	64	41	220	90	310	603.680	0.947	1.581	105	0.593	0.374
7	62	131	617	331	948	1290.539	0.361	0.473	193	0.210	0.418
8	48	20	126	49	175	354.517	1.026	2.400	66	0.887	0.135
9	87	140	1017	498	1515	1558.636	0.029	0.621	227	0.263	8.114
10	111	218	2099	1336	3435	2447.644	0.267	0.509	329	0.223	0.224
11	60	147	925	548	1473	1564.106	0.062	0.544	227	0.235	2.806
12	55	147	623	410	1033	1376.327	0.332	0.374	202	0.176	0.472
13	139	314	2796	1427	4223	3594.044	0.149	0.443	453	0.200	0.341
14	97	146	824	382	1206	1669.906	0.401	0.664	243	0.278	0.308
15	69	67	452	207	659	982.024	0.490	0.793	156	0.323	0.341
16	93	116	776	380	1156	1420.294	0.229	0.768	211	0.321	0.404
17	36	234	1590	688	2278	2027.783	0.110	0.154	270	0.098	0.106
18	38	87	579	260	839	759.957	0.094	0.437	125	0.198	1.098
19	55	36	201	110	311	504.092	0.621	1.526	91	0.581	0.064
20	71	189	909	553	1462	1865.896	0.276	0.376	260	0.176	0.362
21	40	21	124	47	171	305.116	0.784	1.905	61	0.713	0.090
22	108	308	3101	1686	4787	3275.698	0.316	0.351	416	0.167	0.470
23	168	642	4995	2849	7844	7229.477	0.078	0.262	610	0.136	0.737
24	79	253	1564	764	2328	2517.696	0.061	0.312	332	0.154	0.888
25	92	232	2230	1137	3367	2423.219	0.280	0.397	324	0.183	0.345
26	92	404	1967	1195	3162	4098.085	0.296	0.228	496	0.124	0.561
27	82	352	1191	800	1991	3499.039	0.757	0.233	434	0.126	0.634
28	53	106	557	359	916	1033.108	0.128	0.491	161	0.217	0.694
29	92	107	675	324	999	1321.505	0.323	0.660	199	0.346	0.073
30	77	76	443	225	668	972.804	0.456	0.987	155	0.391	0.143
31	67	167	1022	567	1569	1617.697	0.144	0.358	254	0.170	0.182
32	44	70	378	184	562	669.265	0.191	0.629	114	0.265	0.389
33	35	11	83	27	110	217.579	0.978	3.182	46	1.162	0.188
34	67	80	532	269	801	912.182	0.139	0.638	147	0.338	1.438
35	116	315	2340	1222	3562	3409.776	0.043	0.368	431	0.174	3.081
36	76	175	923	538	1461	1778.604	0.218	0.434	151	0.197	0.095
37	106	225	1635	886	2521	2471.260	0.020	0.471	331	0.210	9.628
38	108	187	1169	629	1798	2140.797	0.191	0.578	295	0.247	0.296
39	46	25	169	69	236	370.160	0.555	1.840	71	0.691	0.244
40	106	381	2600	1209	3809	3979.719	0.045	0.278	487	0.142	2.166
41	72	99	646	294	940	1100.541	0.171	0.727	171	0.300	0.755
42	86	64	631	330	961	1089.613	0.134	1.024	170	0.404	2.018
43	55	75	563	224	767	765.136	0.002	0.733	130	0.302	126.439
44	92	193	1440	704	2144	2065.512	0.037	0.477	285	0.212	4.782
45	47	35	188	83	271	440.591	0.626	1.343	82	0.516	0.175
46	150	809	4936	2817	7753	6899.259	0.146	0.185	959	0.109	0.261
47											
48						MRE =	0.317				3.965

Table 5. Result for Strategy 5

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S5	n2-S5	N1-S5	N2-S5	N-S5	EstN-S5	|RE|-S5	n1/n2-S5	n-S5	Est |RE|-S5	|RE| of |RE|-S5
2	68	149	975	523	1498	1489.604	0.006	0.456	217	0.2554	44.5724
3	47	65	292	178	470	652.520	0.388	0.723	112	0.3484	0.1028
4	45	44	175	107	282	487.348	0.728	1.023	89	0.4529	0.3780
5	57	75	319	218	537	788.636	0.489	0.760	132	0.3613	0.2612
6	52	41	193	90	283	516.082	0.824	1.268	93	0.5386	0.3461
7	48	131	529	331	860	1189.457	0.383	0.366	179	0.2241	0.4151
8	39	20	106	49	155	292.569	0.888	1.950	59	0.7763	0.1253
9	65	140	833	498	1331	1389.554	0.044	0.464	205	0.2582	4.8688
10	95	218	1871	1336	3207	2317.600	0.277	0.436	313	0.2482	0.1049
11	65	147	825	548	1373	1449.806	0.056	0.442	212	0.2505	3.4775
12	47	147	561	410	971	1319.418	0.359	0.320	194	0.2078	0.4210
13	111	314	2232	1427	3659	3358.691	0.082	0.354	425	0.2195	1.6750
14	74	146	613	382	995	1509.214	0.517	0.507	220	0.2730	0.4717
15	54	87	374	207	581	871.300	0.500	0.621	141	0.3127	0.3741
16	66	118	602	380	882	1211.082	0.233	0.558	184	0.2813	0.2488
17	24	234	1357	688	2045	1951.704	0.046	0.103	258	0.1320	1.8942
18	26	87	492	260	752	682.748	0.092	0.298	113	0.2005	1.1771
19	42	36	164	110	274	412.595	0.506	1.187	78	0.5031	0.0053
20	52	189	703	553	1256	1725.687	0.374	0.275	241	0.1922	0.4860
21	31	21	100	47	147	245.819	0.672	1.476	52	0.6111	0.0810
22	87	308	2718	1686	4404	3106.706	0.295	0.282	395	0.1848	0.3388
23	141	642	4209	2849	7058	6994.244	0.009	0.220	783	0.1729	18.1363
24	62	253	1394	764	2158	2388.858	0.107	0.245	315	0.1817	0.6988
25	74	232	1929	1137	3066	2282.551	0.256	0.319	306	0.2075	0.1879
26	76	404	1751	1185	2946	3972.760	0.349	0.188	480	0.1618	0.5356
27	64	352	1089	800	1889	3361.720	0.780	0.182	416	0.1597	0.7952
28	47	108	510	359	869	990.594	0.140	0.435	155	0.2480	0.7726
29	71	107	528	324	852	1157.968	0.359	0.684	178	0.3277	0.0876
30	59	78	329	225	554	837.337	0.511	0.756	137	0.3601	0.2960
31	54	187	837	567	1504	1722.033	0.145	0.289	241	0.1870	0.3587
32	35	70	309	184	493	608.575	0.234	0.500	105	0.2706	0.1544
33	27	11	67	27	84	166.436	0.771	2.455	38	0.8523	0.2357
34	55	80	443	269	712	823.729	0.157	0.688	135	0.3360	1.1414
35	94	315	1982	1222	3204	3230.382	0.008	0.298	409	0.2003	23.3302
36	61	175	768	538	1306	1665.737	0.275	0.349	236	0.2178	0.2092
37	88	225	1411	886	2297	2326.531	0.013	0.391	313	0.2327	17.0972
38	80	187	935	629	1564	1917.024	0.226	0.428	267	0.2455	0.0875
39	36	25	138	69	207	302.214	0.460	1.440	61	0.5985	0.3011
40	77	381	2065	1209	3274	3749.102	0.145	0.202	458	0.1667	0.1491
41	52	99	472	294	766	952.729	0.244	0.525	151	0.2794	0.1463
42	65	84	524	330	854	928.409	0.087	0.774	149	0.3661	3.2021
43	46	75	509	224	733	721.245	0.016	0.613	121	0.3102	18.3406
44	66	193	1117	704	1821	1864.274	0.024	0.342	258	0.2155	8.0695
45	38	35	157	83	240	378.946	0.579	1.086	73	0.4749	0.1797
46	119	809	4210	2817	7027	8635.420	0.229	0.147	928	0.1476	0.3553
47											
48						MRE =	0.308				3.4823

Table 6. Result for Strategy 6

	A	B	C	D	E	F	G	H	I	J	K
1	n1-S6	n2-S6	N1-S6	N2-S6	N-S6	EstN-S6	|RE|-S6	n1/n2-S6	n-S6	Est |RE|-S6	|RE| of |RE|-S6
2	78	149	1130	523	1653	1565.917	0.053	0.523	227	0.2318	3.3997
3	54	65	325	178	503	702.218	0.396	0.831	119	0.3386	0.1451
4	55	44	204	107	311	558.190	0.795	1.250	99	0.4843	0.3907
5	67	75	423	218	641	673.589	0.363	0.893	142	0.3603	0.0070
6	58	41	208	90	298	559.423	0.677	1.415	99	0.5415	0.3827
7	55	131	603	331	934	1239.353	0.327	0.420	186	0.1958	0.4012
8	44	20	117	49	166	326.654	0.968	2.200	64	0.8144	0.1585
9	80	140	996	498	1494	1503.854	0.007	0.571	220	0.2484	36.6677
10	107	218	2089	1336	3425	2414.801	0.295	0.491	325	0.2204	0.2526
11	76	147	919	548	1467	1533.194	0.045	0.517	223	0.2295	4.0869
12	53	147	620	410	1030	1361.932	0.322	0.361	200	0.1752	0.4565
13	130	314	2741	1427	4168	3517.419	0.156	0.414	444	0.1937	0.2412
14	86	146	781	382	1163	1602.373	0.378	0.589	232	0.2546	0.3262
15	62	67	441	207	648	929.696	0.435	0.713	149	0.2975	0.3156
16	84	118	739	380	1119	1349.107	0.206	0.712	202	0.2973	0.4455
17	30	234	1583	688	2271	1988.872	0.124	0.128	264	0.0944	0.2401
18	32	87	570	260	830	720.536	0.132	0.368	119	0.1777	0.3473
19	49	36	187	110	297	461.238	0.553	1.381	85	0.5229	0.0544
20	67	189	893	553	1446	1835.692	0.269	0.354	256	0.1731	0.3579
21	35	21	112	47	159	271.764	0.709	1.667	56	0.6291	0.1130
22	101	308	3087	1686	4773	3218.650	0.326	0.328	409	0.1638	0.4970
23	158	642	4943	2849	7792	7141.565	0.083	0.246	800	0.1354	0.6218
24	72	253	1545	764	2309	2463.932	0.067	0.285	325	0.1488	1.2170
25	85	232	2218	1137	3355	2367.850	0.294	0.366	317	0.1772	0.3978
28	86	404	1954	1195	3149	4050.576	0.286	0.213	490	0.1238	0.5675
27	76	352	1182	600	1982	3452.562	0.742	0.216	428	0.1249	0.8317
28	52	108	555	359	914	1025.951	0.122	0.481	160	0.2172	0.7732
29	84	107	645	324	969	1256.292	0.299	0.785	191	0.3227	0.0809
30	71	78	425	225	650	926.693	0.426	0.910	149	0.3662	0.1403
31	61	187	1010	567	1577	1773.044	0.124	0.326	248	0.1632	0.3130
32	39	70	366	184	550	635.180	0.155	0.557	109	0.2435	0.5721
33	30	11	74	27	101	185.260	0.834	2.727	41	0.9977	0.1959
34	62	80	515	269	784	874.914	0.116	0.775	142	0.3192	1.7526
35	110	315	2318	1222	3540	3360.200	0.051	0.349	425	0.1712	2.3710
36	70	175	911	538	1449	1733.012	0.196	0.400	245	0.1889	0.0364
37	98	225	1601	686	2487	2406.342	0.032	0.436	323	0.2012	5.2045
38	101	187	1143	629	1772	2083.749	0.176	0.540	288	0.2376	0.3503
39	42	25	162	69	231	342.574	0.483	1.680	67	0.6337	0.3120
40	98	381	2548	1209	3757	3914.801	0.042	0.257	479	0.1392	2.3152
41	67	99	632	294	926	1062.734	0.148	0.677	166	0.2851	0.9305
42	78	84	614	330	944	1027.216	0.088	0.929	162	0.3726	3.2264
43	51	75	556	224	780	756.455	0.030	0.680	126	0.2862	8.4805
44	64	193	1378	704	2082	2002.299	0.038	0.435	277	0.2011	4.2536
45	42	35	177	83	260	406.002	0.562	1.200	77	0.4669	0.1685
48	139	809	4859	2817	7676	8804.469	0.147	0.172	948	0.1096	0.2547
47											
48						MRE =	0.295				1.8812

Table 7

Explanation of Column Headings

Column heading	Measured value
n_1-S*	n_1
n_2-S*	n_2
N_1-S*	N_1
N_2-S*	N_2
N-S*	Actual length N
EstN-S*	Estimated length N
|RE|-S*	Absolute relative value

Note: * stand for strategy number

Chapter 3

ANALYSIS

3.1 Statistical Procedure Used

The usefulness of software measures in the management of software development not only should be justified theoretically, but also should be supported by empirical results. To investigate the relationship between the counting strategies with the Halstead length equation, several statistical procedures and packages were used to analyze the 6 sets of empirical data obtained from running 6 versions of the Pascal counting programs.

In order to assess the strength of the relationship between observed length N and estimated length N_{est} Pearson's correlation coefficients were computed for each set of data. Pearson's correlation coefficient is a measure indicating the degree of linear relationship between two variables. It may not imply a cause-effect relationship. The symbol for the correlation coefficient is r. When there is a perfect linear relationship, positive or negative, between these two variables, then r = 1 or -1, respectively. When there is little or no linear relationship between them, r has a value close to 0. The absolute value of the correlation is always a number between 0 and 1. Since the correlation coefficient cannot be a complete test of validity of the length equation, other methods of analysis are needed.

There are several criteria which have been successfully used in software metric research [Conte 86]. The most natural and important measure of the accuracy of an estimator of M is the mean absolute relative error, $\overline{MRE}$, defined as

$$\overline{MRE} = \frac{1}{j} \sum_{i=1}^{j} MRE_i$$

$$\text{where } MRE_i = |RE_i| = \left| \frac{M_{(observed)} - M_{(estimated)}}{M_{(observed)}} \right|$$

Here, j is the number of M in the data set being investigated, RE_i is the relative error of the estimate, MRE is its absolute value. An estimator which consistently yields small values of $\overline{MRE}$ is desirable. The smaller the $\overline{MRE}$, the more accurate the estimation.

The scatter diagrams of different variables were plotted to see the relationship between the variables. Regression analysis, ANOVA, and LSD were also used.

In the analysis, several tools were used. A spreadsheet package, Excel, and a statistical analysis package, MacSS, both for the Macintosh were used to manipulate numerical data, to calculate correlation coefficient, $|RE_i|$, $\overline{MRE}$, and to plot the 2-dimensional graph, the scatter diagram.

3.2 Analysis

The correlation coefficient calculated for observed program length N versus estimated length N_{est}, which were previously obtained by using our six counting strategies, are shown in Table 8. As seen from the table these correlation coefficient valued between 0.959 and 0.961, thus indicating that the relationship between N and N_{est} are strongly correlated as found in previous studies. These six values of r are very similar and all have the significant level of 0.00. It seems that there are only minor differences of r between counting strategies.

Table 8

Averages and Correlation Coefficient of N vs N_{est}

for 6 Counting Strategies

Strategy	Avg n_1	Avg n_2	Avg N_1	Avg N_2	Avg N	Avg N_{est}	Corr. Coeff.
1	63.9	189.0	1112.4	659.4	1771.8	1904.9	0.9606
2	47.2	189.0	930.6	659.4	1590.0	1782.4	0.9602
3	57.8	189.0	1092.4	659.4	1751.8	1859.4	0.9600
4	78.6	174.3	1155.6	616.3	1771.8	1882.5	0.9599
5	61.8	174.3	973.7	616.3	1590.0	1754.6	0.9591
6	72.4	174.3	1135.5	616.3	1751.8	1835.3	0.9592

In order to measure the discrepancy between N and N_{est} of the different counting strategies, $|RE_i|$ for each of 45 programs in 6 sets of data and $\overline{MRE}$ were obtained by using the Excel spreadsheet package. Entries in the 7th column of Tables 1-6 show the values of |RE| and $\overline{MRE}$. The $\overline{MRE}$ values, 0.318, 0.314, 0.296, 0.317, 0.308, and 0.295 show the overall performance of N_{est} is not bad. As with the correlation coefficients, the $\overline{MRE}$ values were not significantly different between the six counting strategies. Among them, strategies 3 and 6 both with max count of tokens show to have the best value of $\overline{MRE}$. Even though the $\overline{MRE}$ is not large, there may be one or more individual predictions that could be very bad.

For easier observation, six scatter diagrams of |RE| vs N_{est} were plotted (Figures 1-6). By looking at the figures it can be seen that most of the points

```
           214.60   1684.50   3154.40   4624.30   6094.20   7564.10   9034.

             +-----+----+----+----+----+----+----+----+----+----+----+
          1.00| *
           .93|*  *
           .87|   *
           .81|                    *
           .75| *
           .69|
           .62|  *
           .56| *
|RE|-S1    .50|    *
           .44|   **   *
           .38|    * **
           .31|      *            *     *
           .25|      *  *   **
           .19|   *  *    *                                        *
           .13|    **    **        *
           .07|    *    *     *       *                 *
          0.00|    *     *  *  *      *
             +-----+----+----+----+----+----+----+----+----+----+----+
           214.60   1684.50   3154.40   4624.30   6094.20   7564.10   9034.

                              Estimated Length - S1
```

Figure 1. Scatter Diagram of Estimated Length vs |RE| for Strategy 1

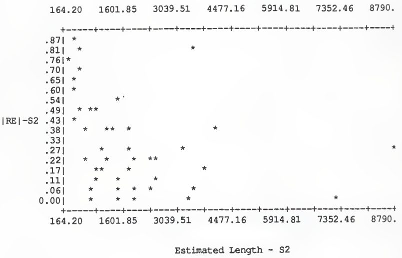

Figure 2. Scatter Diagram of Estimated Length vs |RE| for Strategy 2

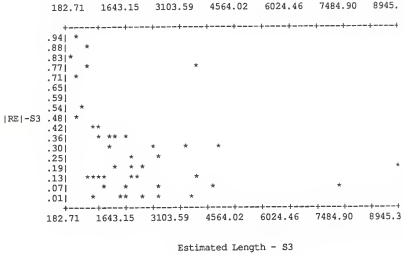

Figure 3. Scatter Diagram of Estimated Length vs |RE| for Strategy 3

```
              217.58    1664.53   3111.47   4558.42   6005.37   7452.31   8899.

              +----+----+----+----+----+----+----+----+----+----+----+----+----+
         1.03|  *
          .96|*   *
          .90|
          .83|    *
          .77|  *                     *
          .71|
          .64|   *
          .58|  *
|RE|-S4   .51|      *
          .45|     **
          .39|      * *  *
          .32|         *            *     *
          .26|         *  *   *
          .19|    *  *    * *
          .13|      **    * *         *                                    *
          .07|     *    ** *  *      *   *                     *
         0.00|     *    *      *
              +----+----+----+----+----+----+----+----+----+----+----+----+----+
              217.58    1664.53   3111.47   4558.42   6005.37   7452.31   8899.

                                  Estimated Length - S4
```

Figure 4. Scatter Diagram of Estimated Length vs |RE| for Strategy 4

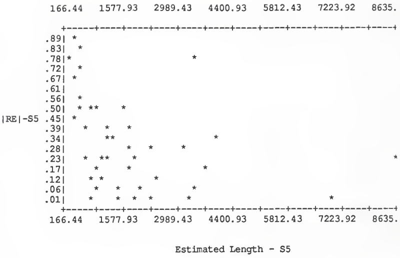

Figure 5. Scatter Diagram of Estimated Length vs |RE| for Strategy 5

```
            185.26   1621.79   3058.33   4494.86   5931.40   7367.93   8804.

              +-----+----+-----+-----+----+----+-----+-----+-----+-----+----+----+
           .97|  *
           .91|
           .85|*   *
           .79|    *
           .73|  *                   *
           .67|
           .61|
           .55|   *
|RE|-S6    .49|  *
           .43|      *
           .37|     **     *
           .31|        **       **     *      *
           .25|            *
           .19|          *  * *
           .13|    ****    * *        *                                    *
           .07|        *  **  *  *   *   *                      *
           .01|      *    *     *
              +-----+----+-----+-----+----+-----+-----+-----+-----+-----+-----+
            185.26   1621.79   3058.33   4494.86   5931.40   7367.93   8804.4

                                 Estimated Length - S6
```

Figure 6. Scatter Diagram of Estimated Length vs |RE| for Strategy 6

are scattered toward the lower left and seems N_{est} and |RE| have a negative relationship. |RE| tends to be high for a small value of N_{est} and tends to be low for a large value N_{est}. Moreover, in the scatter plot six components with unusually high |RE| values, termed "outlier," were found. A close look at the six programs with high |RE| values shows that five of them are among the smallest in terms of N in the set of 45 COBOL programs, while the other program although medium in size used a MOVE TO statement to initialize a large number of variables. The observation of Zweben and Fungs' study [Zweben 79], which measured 25 small COBOL programs, showed that the relative error of N and N_{est} are much higher than in similar comparisons, obtained from Fortran and PL/I programs. Our results agree with this in that small programs have a high value of |RE| (greater than 0.5). COBOL language has a large repertoire of verbs which also are reserved words and are counted as operators. The high discrepancy between the values of N and N_{est} is probably caused by the existence of a large number of infrequently referenced operators which come from these verbs. These verbs contributed little to N but largely to n_t. For the outlier which has a large amount of data items initialized by a single MOVE statement, the infrequently referenced operands caused n_2 to be large, hence also a large value of N_{est}. Therefore, these outliers had high |RE| values.

The above observation implies that N_{est} does not work well for small programs and it appears that $n_2 = 50$ or $n_1/n_2 = 1$ may be a good separator between where N_{est} works and does not work well. As we calculated the correlation coefficients (Table 9) and plotted the scatter diagrams (Figures 7-12) between n_t/n_2 and |RE| for six sets of data, the results indicate there exists a positive linear relationship between n_t/n_2 and |RE|. Every one of the six set data were then separated into two sets: one set of $n_1/n_2 < 1$ and another set of $n_1/n_2 \geq 1$. There were 8, 6, 8, 9, 8, and 8 programs in set 2 respectively. Correlation coefficients of N versus N_{est} and $\overline{MRE}$ for each of these separated sets were again calculated. As shown in Tables 10 and 11 the correlation coefficients are all above 0.95 and remain very high. Compared with the $\overline{MRE}$ for all 45 programs, the $\overline{MRE}$ for the set of $n_1/n_2 < 1$ decreased little to 0.219, 0.259, 0.206, 0.215, 0.228, and 0.203, respectively. On the other hand, for the set of $n_1/n_2 \geq 1$, the $\overline{MRE}$ are 0.789, 0.676, 0.713, 0.726, 0.678 and 0.723, respectively, which increased dramatically and are more than double those for the 45 programs. Further, we would like to know if n_1/n_2 can be a possible linear predictor of |RE|. The slope and intercept obtained from regression analysis of n_1/n_2 and |RE| were used to calculate the estimated |RE| (see Tables 1-6). The correlation coefficients of |RE| against estimated |RE| for six strategies are 0.7637, 0.681, 0.7263, 0.774, 0.6814, and 0.7407, respectively. From the high $\overline{MRE}$ of |RE| and estimated |RE| (Table 12) we can say that n_1/n_2 is not a good linear prdictor of |RE|.

Table 9

Correlation Coefficient of n_1/n_2 vs |RE|
for 6 Counting Strategies

Strategy	Correlation Coefficient
1	0.7636
2	0.6811
3	0.7264
4	0.7740
5	0.6814
6	0.7407

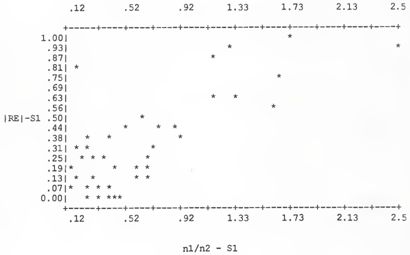

Figure 7. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 1

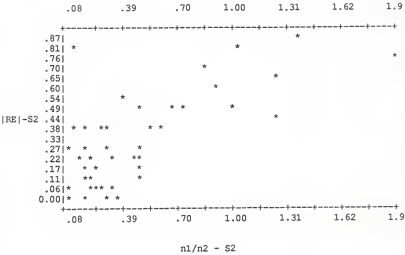

Figure 8. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 2

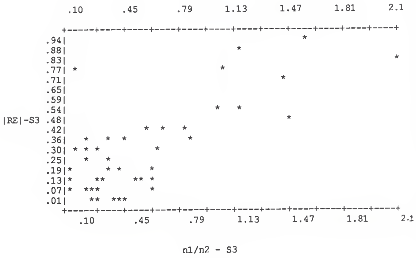

Figure 9. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 3

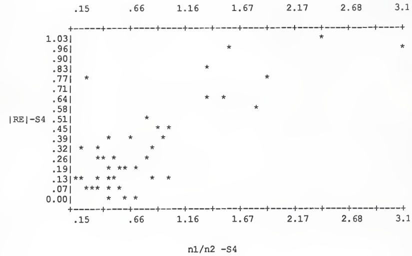

Figure 10. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 4

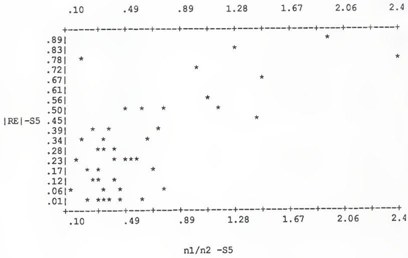

Figure 11. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 5

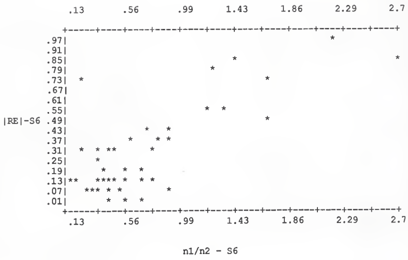

Figure 12. Scatter Diagram of n_1/n_2 vs |RE| for Strategy 6

Table 10

Correlation Coefficient of N vs N_{est} for the Programs with $n_1/n_2 < 1$

Strategy	Correlation Coefficient
1	0.9531
2	0.9549
3	0.9523
4	0.9515
5	0.9513
6	0.9513

Table 11

Correlation Coefficient of N vs N_{est} for the Programs with $n_1/n_2 \geq 1$

Strategy	Correlation Coefficient
1	0.9545
2	0.9567
3	0.9552
4	0.9667
5	0.9615
6	0.9534

Table 12

$\overline{MRE}$ Values of |RE| and Predicted |RE|
for Six Strategies

Strategy	$\overline{MRE}$
1	2.287
2	2.923
3	1.708
4	3.965
5	3.482
6	1.881

In the previous section of our statistical analysis showed that by applying 6 different counting strategies the differences of correlation coefficients for N and N_{est} are very small between different strategies for 45 sample programs. The same phenomenon occurred with the analysis of MRE differences.

To check the sensitivity of N_{est} to the different counting strategies the statistical techniques of the ANOVA and Multiple Comparison procedure were used. The ANOVA procedure using significant level of 0.05 was performed on six set of n_1, n_2, n, N_{est}, and |RE|. The results showed that for n_1 there is a significant difference in the six strategies. Multiple comparison of means using LSD (Least Significant Difference) were also done to see where the significant differences between the means of n_1 existed. We found that for n_1 strategies 4 and 6; 6 and 1; and 1, 5, and 3 are not significantly different. The comparisons of means of n_2, n, N_{est}, and |RE| by the LSD showed that there is no significant differences at all between the six counting strategies.

Chapter 4

CONCLUSIONS

4.1 Summary of Findings

In this report we devised six different counting strategies to count the operators and operands in programs. Several statistical techniques were used to analyze 6 sets of empirical data obtained from running the counting program on 45 COBOL sample programs. The high correlation coefficient calculated for N and N_{est} agreed with the high levels of correlation found in published studies. Although N_{est} appears to overestimate the observed length N for small programs, and to underestimate N for large programs, the overall performance of N_{est} as a predictor of the actual length N was good. The relative errors show N_{est} does not work well for small programs. The value of n_1/n_2 seems a good separator of when N_{est} works well ($n_1/n_2 < 1$) and when it does not work well ($n_1/n_2 > 1$). However, n_1/n_2 is not a good predictor of $|RE|$.

The statistically significant differences between several pairs of correlation coefficient, $\overline{MRE}$, and n_1 are very few. There is no significant differences between pair of means of n_2, n, N_{est} and $|RE|$. In summary, the length equation is insensitive to the counting strategies and does not favor any one of the strategies.

4.2 Future Work

Software science is much like actuarial studies of populations. The results make it possible to predict gross properties of the whole. The software science measures become more accurate when applied to large numbers of programs then when applied to individual programs. We think more data collection and technology evaluation efforts are needed to establish a sufficient empirical basis for the formulation of software development standards. This report presents a very preliminary exploration of the applicability of software science measures to languages other than Fortran, Algol, and PL/I. It is worth trying to expand to other languages such as Ada, concurrent C, etc.

BIBLIOGRAPHY

[Basili 83] Basili, V. R., Selby, R. W., and Phillips, T. Y., "Metric Analysis and Data Validation across Fortran Projects," IEEE Trans. on Software Engineering , SE-9(6), March 1983, pp. 652-663

[Christ 83] Christensen, K., Fitso, G. P., and Smith, C.P., "A Perspective on Software Science," IBM System Journal, 20(4), 1981, pp. 372-387

[Conte 82] Conte, S. D., Shen, V. Y., and Dickey, K., "On the Effect of Different Counting Rules for Control Flow Operators on Software Science Metrics in Fortran," ACM Performance Evaluation Review, Summer 1982, pp. 118-126

[Conte 86] Conte, S. D., Dunsmore, H. E., and Shen, V. Y., Software Engineering Metrics and Models. The Benjamin/Cummings Publishing Co., Menlo Park, CA., 1986

[Cook 82] Cook, M. L., "Software Metrics: An Introduction and Annotated Bibliograph," Software Engineering Notes, 7(2), April 1982, pp. 41-60

[Debnath 84] Debnath, N. C., and Zweben, S. H., "A Study of the Application of Software Metrics to COBOL," Tech. Rep. OSU-CISRC-TR-84-3, Computer & Information Science Research Center, Ohio State University, 1984

[Debnath 85] Debnath, N. C., "On Halstead's Language Level," Engineering Software IV, Proceedings of the 4th International Conference, London, England, June 1985, pp. 2.21-2.29

[Elshoff 78] Elshoff, J. L., "An Investigation into the Effects of the Counting Methods Used on Software Science Measurement," ACM SIGPLAN NOTICES, 13(2), February 1978, pp. 30-45

[Feuer 79] Feuer, A. R., and Fowlkes, E. B., "Some Results from Empirical Study of Computer Software," 4th International Conference on Software Engineering, September 1979, pp. 351-355.

[Fitsos 80] Fitsos, G. P., "Vocabulary Effects of Software Science," Proceedings of COMPSAC 80, October 1980, pp. 751-756

[Fitz 78] Fitzsimmons, A., and Love, T., "A Review and Evaluation of Software Science," ACM Computing Survey, 10(1), March 1978, pp. 3-18

[Gustafson 83] Gustafson, D. A., and White, B., "Application of Halstead's Complexity Measures to Programs Designed with Wanier-Orr Diagrams," 1983 Symposium on Application and Assessment of Automated Tools for Software Development, pp. 147-155.

[Halstead 77] Halstead, M. H., Elements of Software Science, Elsevier North-Holland, NY., 1977

[Harrison 74] Harrison, W., "Bibliography on Software Complexity Metrics" ACM SIGPLAN NOTICES, 19(2), February 1984, pp. 17-27

[Hamer 82] Hamer, P. G., and Frewin, G. D., "M. H. Halstead's Software Science - A Critical Examination," Proceedings of the 6th International Conference on Software Engineering, Tokyo, September 1982, pp. 197-206

[Henry 81] Henry, S., and Kafura, D., "Software Structure Metrics Based on Information Flow," Trans. on Software Engineering, SE-7(5), September 1981, pp. 510-518

[Ivan 87] Ivan, I., Arhire, R., and Macesanu, M., "Program Complexity Comparative Analysis, Hierarchy, Classification," ACM SIGPLAN NOTICES, 22(4), April 1987, pp. 94-102

[Jensen 85] Jensen, H. A., and Vairavan, K., "An Experimental Study of Software Metrics for Real-Time Software," IEEE Trans. on Software Engineering, SE-11(2), February 1985, pp. 231-234

[Lassez 81] Lassez, J., et. al., "A Critical Examination of Software Science, " Journal of Systems and Software, 2(2), December 1981, pp. 105-112

[Levitin 85] Levitin, A. V., "On Predicting Program Size by Program Vocabulary," Proceedings of COMPSAC 85, October 1985, pp. 98-103

[Levitin 86] Levitin, A. V., "How to Measure Software Size and How Not to," Proceedings of COMPSAC 86, October 1986, pp. 314-318

[Levitin 87] Levitin, A. V., "Investigating Predictability of Program Size," Proceedings of COMPSAC 87, October 1987, pp. 231-235

[Lister 82] Lister, A. M., "Software Science - the Emperor's New Clothes?" The Australian Computer Journal, 14(2), May 1982, pp. 66-70

[McCabe 76] McCabe, T. J., "A Complexity Measure," IEEE Trans. on Software Engineering, SE-2(4), December 1976, pp. 308-320

[Salt 82] Salt, N. F., "Defining Software Science Counting Strategies," ACM SIGPLAN NOTICES, 17(3), March 1982, pp. 58-67

[Shen 81] Shen, V. Y., and Dunsmore, H. E., "Analyzing COBOL Programs via Software Science," Tech. Report CSD-TR-348, Department of Computer Science, Purdue University, September 1981

[Shen 83] Shen, V. Y., Conte, S. D., and Dunsmore, H. E., "Software Science Revisited : A Critical Analysis of the Theory and its Empirical Support," IEEE Trans. on Software Engineering, SE-9(2), March 1983, pp. 156-165

[Smith 80] Smith, C. P., "A Software Science Analyses of Programming Size," Proceedings ACM Annual Conference, October 1980, pp. 179-185

[Spence 85] Spence, J. M., COBOL for the 80's, West Publishing Co., 1985

[Wague 87] Waguespack, L. J., and Badlani, S., "Software Complexity Assessment : An Introduction and Annotated Bibliography," ACM SIGSOFT Software Engineering Notes, 12(4), October 1987, pp. 52-71

[Wang 85] Wang, A. S., and Dunsmore, H. E., "Early Software Size Estimation : A Critical Analysis of the Software Science Length Equation and a Data-Structure-Oriented Size Estimation Approach," Proceedings of the 3rd Symposium Empirical Foundations of Information and Software Science, Denmark, October 1985, pp. 211-222

[Wood 85] Wood, D. E., and Konstan, A. H., "Software Science Applied to APL," IEEE Trans. on Software Engineering, SE-11(10), October 1985, pp. 994-1000

[Zweben 79] Zweben, S. H., and Fung, K., "Exploring Software Science Relations in COBOL and APL," Proceedings of the 3rd International Conference on Software Engineering, Chicago, IL, 1979, pp. 702-707

Appendix A

Source Code of Counting Program

```
program counting1 (f1, fchange, input, output);
const
   maxchar = 30;

type
   alfa = packed array[1..maxchar] of char;
   ptr = ^noderecord;
   noderecord =
      record
         token : alfa;            (* value of the token read in *)
         CntOrNot:char;           (* Y = counted, N = not counted *)
         category,                (* 1 = operator, 2 = operand   *)
                                  (* 3 = paragraph name, 0 = undecide *)
         FirstInData,             (* token # where first used in data division *)
         FirstInProc,             (* token # where first used in procedure div *)
         UseInData,               (* token occurrence in data division *)
         UseInProc : integer;     (* token occurrence in procedure division *)
         left, right : ptr;       (* point to left/right subtree *)
   end;

var
   fchange,          (* external file contains figurative constant *)
   f1, : text;       (* external file contains reserved words & most of character set *)
   word  : alfa;     (* the read in token *)
   flag,             (* indicates whether to start counting *)
   Dcnt,             (* number of tokens in data division *)
   Pcnt,             (* number of tokens in procedure division *)
   belongs,          (* is operator, operand or paragraph name *)
   n1Data,           (* number of distinct operators in data division *)
   n1Proc,           (* number of distinct operators in procedure division *)
   n1,               (* number of distinct operators in the program *)
   n2Data,           (* number of distinct operands in data division *)
   n2Proc,           (* number of distinct operands in procedure division *)
   n2,               (* number of distinct operands in the program *)
   N1Data,           (* operator occurrences in data division *)
   N1Proc,           (* operator occurrences in procedure division *)
   N1,               (* operator occurrences in the program *)
   N2Data,           (* operand occurrences in data division *)
   N2Proc,           (* operand occurrences in procedure  division *)
   N2,               (* operand occurrences in the program *)
   N :integer;       (* actual length of a program *)
   EstLen : real;    (* estimated length of a program *)
   head      : ptr;
   ch,  division, WhetherCnt  : char;
   comment, paragraph,       (* indicates whether token is in comment, *)
   pic     : boolean         (* paragraph name or picture clause        *)
```

```
        (*  Initialize variables used in the program  *)

procedure initialization;
var
    i : integer;

begin
    flag := 0;
    Dcnt := 0;
    Pcnt := 0;
    belongs := 1;
    head := nil;
    division := 'I';
    for I := 1 to maxchar do
        word[i] := ' ';
    comment := false;
    paragraph := false;
    pic := false;
    n1Data := 0;
    n1Proc := 0;
    n1 := 0;
    n2Data := 0;
    n2Proc := 0;
    n2 := 0;
    N1Data := 0;
    N1Proc := 0;
    N2Data := 0;
    N2Proc := 0
end;


    (*   Every COBOL reserved word and character set except 26 alphabets   *)
    (*   is first built into a binary tree. For every token read if not in the   *)
    (*   tree inserted it otherwise occurrence count and category are updated   *)

procedure search(word : alfa; var head : ptr; WhetherCnt : char; var belongs : integer);
var
    p : ptr;

begin
    p := head;
    if p = nil then          (* insert token into the tree *)
    begin
        new(p);
        with p^ do
        begin
            token := word;
            if flag = 0 then     (* not start counting yet *)
            begin
                CntOrNot := WhetherCnt;
                category := belongs;
```

```
            UseInData := 0;
            UseInProc := 0;
            FirstInData := 0;
            FirstInProc := 0
        end
        else        (* flag = 1, start counting *)
        begin
            CntOrNot := 'Y';
            if belongs = 0 then
                category := 2
            else
                category := belongs;
            if division = 'D' then
            begin
                FirstInData := Dcnt;
                UseInData := 1
            end
            else    (* token occurs 1st time in procedure division *)
            begin
                FirstInProc := Pcnt;
                UseInProc := 1
            end;
        end;
        left := nil;
        right := nil;
    end; (* with *)
    head := p
end         (*  if *)
else        (*  current token is already in the tree *)
    if word < p^.token then
        search(word, p^.left, WhetherCnt, belongs)
    else
        if word > p^.token then
            search(word, p^.right, WhetherCnt, belongs)
        else   (* find the token *)
        with p^ do
        begin
            if flag = 0 then
                if category <> belongs then     (* reserved word as operand  *)
                    category := belongs
                else
            else
            begin
                if   ((category = 1) and (belongs = 2)) or
                     ((category = 2) and (belongs = 1)) then
                     search(word, p^.left, WhetherCnt, belongs)
                else
                begin
                    if paragraph then           (* is a paragraph name *)
                        category := 3;
                    if division = 'D' then
                    begin
```

```
                        UseInData := UseInData + 1;
                        if FirstInData = 0 then
                            FirstInData := Dcnt
                    end
                    else
                        if division = 'P' then
                        begin
                            UseInProc := UseInProc + 1;
                            if FirstInProc = 0 then
                                FirstInProc := Pcnt
                        end
                end
            end
        end  (* with  *)
end;    (*  of procedure search  *)


    (*   Read in the external file which contains  all reserved words;        *)
    (*   operators which  include arithmetic,  relational,  and  logic         *)
    (*   operators;  punctuation and dollar sign $, and insert into            *)
    (*   the binary tree                                                       *)

procedure readdata(var F1 : text;  WhetherCnt : char);
var
    i, j        : integer;
    chars       : alfa;
    bk1,  bk2 : char;

begin
    reset(F1);
    while not eof(F1) do
    begin
        if not eoln(F1) then
        begin
            i := 0;
            read(F1, WhetherCnt);
            read(F1, bk1, bk2);
            while not eoln(F1) do
            begin
                i := i + 1;
                read(F1, chars[i])
            end;
            if i < maxchar then
                for j := i + 1 to maxchar do
                    chars[j] := ' '
        end;        (*  of not eoln  *)
        readln(F1);
        search(chars, head, WhetherCnt, belongs)
    end        (*  of while  *)
end;       (*  of procedure  readdata  *)
```

```
    (*      Change the category of reserved words which are              *)
    (*      figurative constants from  operator to operand               *)

procedure readchange(var F2 : text);
var
    I, j      : integer;
    chars : alfa;

begin
    reset(F2);
    while not eof(F2) do
    begin
        i := 0;
        while not eoln(F2) do
        begin
            I := i + 1;
            read(F2, chars[i])
        end;
        if i < maxchar then
            for j := i + 1 to maxchar do
                chars[j]:= ' ';
        readln(F2);
        belongs := 2;
        search(chars, head, WhetherCnt, belongs)
    end;
    belongs := 0
end;        (* of procedure readchange *)


        (* Ignore comment in the counting of operators and operands *)

procedure Comment(var col: integer);
begin
    comment := true;
    repeat
        write(ch);
        read(ch);
        col := col + 1
    until eoln or (col = 72);
    write(ch)
end;        (* of procedure comment *)


        (* Read in a "*" to see if this is in a comment line, if    *)
        (* not then check whether it is an operator "*" or "**"     *)

procedure asterisk(var col, Pcnt, belongs : integer;  word: alfa;  var head : ptr);
var
 I  : integer;
```

```
begin
    if col = 7 then   (* this is comment, ignore it *)
        Comment(col)
    else
    begin
        write(ch);
        i := 1;
        word[i] := ch;
        read(ch);
        col := col + 1;
        if ch = '*' then
        begin
            write(ch);
            i := i + 1;
            word[i] := ch;
            read(ch);
            col := col + 1
        end;
        if division = 'P' then
        begin
            Pcnt := Pcnt + 1;
            search(word, head, WhetherCnt, belongs)
        end;
        for i := 1 to maxchar do
            word[i] := ' '
    end;
    belongs := 0
end;  (*  of  procedure asterisk  *)


        (*  The token is in picture clause. Every picture character symbol    *)
        (*  is counted as an operator unless it is enclosed by  parentheses   *)

procedure picture(var col, Dcnt, belongs : integer;  var head : ptr);
var
  i , j : integer;

begin
    repeat
        write(ch);
        word[1] := ch;
        Dcnt := Dcnt +1;
        belongs := 1;
        search(word, head, WhetherCnt, belongs);
        for j := 1 to maxchar do
            word[j] :=  ' ';
        if ch = '(' then
        begin
            read(ch);
            col := col + 1;
            i := 0;
```

```
            while (ch <> ')') do
            begin
                write(ch);
                i := i + 1;
                word[i] := ch;
                read(ch);
                col := col + 1
            end;
            belongs := 2;
            Dcnt := Dcnt +1;
            search(word, head, WhetherCnt, belongs);
            for j := 1 to maxchar do
                word[j] := ' ';
        end    (* of if ch = '(' *)
        else
        begin
            read(ch);
            col := col + 1
        end;
    until (ch = ' ') or (eoln) or (col = 72);
    belongs := 0;
    pic := false
end;        (* procedure picture *)



    (* Character read in is a relational operator, arithmetic operator or a delimiter *)

procedure mark(var col, Dcnt, Pcnt, belongs : integer; word : alfa; var head : ptr);
var
  i : integer;

begin
    if (ch = '/') and (col = 7) then
        Comment(col)     (* program continues on next page *)
    else
        if pic then
            picture(col, Dcnt, belongs, head)
        else
        begin
            word[1] := ch;
            write(ch);
            if (division = 'D') or (division = 'P') then
            begin
                if division = 'D' then
                    Dcnt := Dcnt + 1
                else
                    Pcnt := Pcnt + 1;
                belongs := 1;
                search(word, head, WhetherCnt, belongs)
            end;
```

```
            read(ch);
            col := col + 1
        end;
    for i := 1 to maxchar do
        word[i] := ' ';
    belongs := 0
end;   (* of procedure  mark  *)



        (*  Character read is an alphabetic, it can  be part of an operator   *)
        (*  or an operand.   If is not a paragraph name or picture  clause   *)
        (*  then it is an operand                                               *)

procedure alphabet(var col, Dcnt, Pcnt, belongs : integer;   word: alfa;   var head : ptr);
var
  i : integer;

begin
    if pic then
        picture(col, Dcnt, belongs, head)
    else
    begin
        i := 0;
        if (division = 'P') and (col = 8) then
        begin
            paragraph := true;
            belongs := 3
        end;
        repeat
            write(ch);
            i := i + 1;
            word[i] := ch;
            read(ch);
            col := col + 1;
        until (ch in ([' ',  '.',  '(', ')',  ';',  ','])) or (eoln) or (col = 72);
        if ((eoln) or (col = 72)) and  (ch in (['A'..'Z', '0'..'9'])) then
        begin
            write(ch);
            word[i+1] := ch;
        end;
        if (word = 'DATA                        ')  and  (col = 12) then
        begin
            division := 'D';
            flag := 1
        end
        else
            if (word = 'PROCEDURE                  ')  and  (col = 17) then
                division := 'P';
        if (word='PIC                    ')  or (word='PICTURE                   ') then
            pic := true;
```

```
        if ((division = 'D') or (division = 'P')) then
        begin
            if division = 'D' then
                Dcnt := Dcnt + 1
            else
                Pcnt := Pcnt + 1;
                (*  same as the symbol in picture clause  *)
            if   (word = 'A                    ') or (word = 'B                    ') or
                 (word = 'P                    ') or (word = 'S                    ') or
                 (word = 'V                    ') or (word = 'X                    ') then
                 belongs := 2;
            search(word, head, WhetherCnt, belongs)
        end;
        paragraph := false;
        if (eoln or (col = 72)) and  (ch in ['A'..'Z', '0'..'9']) then
        begin
            readln;
            writeln;
            col := 0;
            if not eof then
            begin
                read(ch);
                col := col + 1
            end
        end
    end;
    for i := 1 to maxchar do
        word[i] := ' ';
    belongs := 0
end;    (* of procedure  alphabet *)


        (*   Read in a digit number    *)

procedure digit(var col, Dcnt, Pcnt, belongs : integer;  word : alfa;  var head : ptr);
var
    i : integer;
    tem : char;

begin
    if pic then
        picture(col, Dcnt, belongs,head)
    else
    begin
        tem := ' ';
        if (col = 8) and (division = 'P') then      (* paragraph name  *)
        begin
            paragraph := true;
            belongs := 3
        end
```

```
else
    belongs := 2;
i := 0;
repeat
    if col < 7 then      (* skip the sequence number *)
        write(' ')
    else
        write(ch);
    i := i + 1;
    word[i] := ch;
    read(ch);
    col := col + 1;
until (ch in ([' ', '(', ')', ',', '"'])  or (eoln) or (col = 72);
if (ch = ' ') and (word[i] = '.') then
begin
    tem := word[i];
    word[i] := ' '
end;
if ((eoln) or (col = 72)) and  (ch in ['A'..'Z', '0'..'9']) then
begin
    if col > 7 then
        write(ch);
    word[i+1] := ch
end;
if (col>7) and ((division='D') or  (division='P')) then
begin
    if division = 'D' then
        Dcnt := Dcnt + 1
    else
        Pcnt := Pcnt + 1;
    search(word, head, WhetherCnt, belongs);
    if not (tem = ' ') then
    begin
        paragraph := false;
        for i := 1 to maxchar do
            word[i] := ' ';
        word[1] := tem;   (*  this is a period *)
        belongs := 1;
        search(word, head, WhetherCnt, belongs);
        if  division = 'D'  then  Dcnt := Dcnt + 1
        else  Pcnt := Pcnt + 1;
        tem := ' '
    end     (*  not tem = ' '  *)
end;        (*  col  >  7  *)
paragraph := false;
if (eoln or (col = 72)) and  (ch in ['A'..'Z', '0'..'9']) then
begin
    readln;
    writeln;
    col := 0;
```

```
            if not eof then
            begin
                read(ch);
                col := col + 1
            end
        end     (* of eoln *)
    end;        (* of not pic *)
    for i := 1 to maxchar do
        word[i] := ' ';
    belongs := 0
end;   (* of procedure digit *)



        (*  Read in the delimiter of a literal ',  ' is an operator    *)
        (*  and the literal is an operand                              *)

procedure string(var col, Dcnt, Pcnt, belong s: integer; word : alfa; var head : ptr);
var
    i, j : integer;

begin
    write(ch);
    word[1] := ch;
    if (division = 'D') or (division = 'P') then
    begin
        if division = 'D' then
            Dcnt := Dcnt + 3
        else
            Pcnt := Pcnt + 3;
        belongs := 1;
        search(word, head, WhetherCnt, belongs)
    end;
    for j := 1 to maxchar do
        word[j] := ' ';
    read(ch);
    col := col + 1;
    i := 0;
    while not (ch = '"') do
    begin
        write(ch);
        i := i + 1;
        word[i] := ch;
        read(ch);
        col := col + 1
    end;
    if (division = 'D') or (division = 'P') then
    begin
        belongs := 2;
        search(word, head, WhetherCnt, belongs)
    end;
```

```
    for j := 1 to maxchar do
        word[j] := ' ';
    if (not eoln) and (col < 73) then
    begin
        write(ch);
        word[1] := ch;
        if (division = 'D') or (division = 'P') then
        begin
            belongs := 1;
            search(word, head, WhetherCnt, belongs)
        end;
        for j := 1 to maxchar do
            word[j] := ' ';
        belongs := 0;
        read(ch);
        col := col + 1
    end
end;  (* of procedure string  *)



    (*    Read in the COBOL program     *)

procedure readinput(var word : alfa;  var belongs : integer;  var head : ptr);
var
    i, col : integer;

begin
    col := 0;
    read(ch);
    col := col + 1;
    while not eof do
    begin
        if eoln or (col = 72) then
        begin
            if ((ch = '*') or (ch = '/')) and (col = 7) then
                comment := true;
            if   (not comment) and (not(ch in ['A'..'Z', '0'..'9']) or
                ((word = '                    ')  and (ch in ['A' ..'Z', '0'..'9']))) then
            begin
                if (col >6) and (col < 73) then
                    write(ch);
                if (ch <> ' ') and  ((division = 'D') or (division = 'P')) then
                begin
                    word[1] := ch;
                    if (ch in ['A'..'Z', '0'..'9']) then
                        belongs := 2
                    else
                        belongs := 1;
                    search(word, head, WhetherCnt, belongs);
```

```
                if division = 'D'  then  Dcnt := Dcnt + 1
                else    Pcnt := Pcnt + 1;
                for I := 1 to maxchar do
                     word[i] := ' '
            end
        end;
        comment := false;
        belongs := 0;
        readln;
        writeln;
        col := 0;
        if not eof then
        begin
            read(ch);
            col := col + 1
        end
    end  (* of eoln  *)
    else
    begin
        if ch = '*' then
            asterisk(col, Pcnt, belongs, word, head)
        else
        if ch in (['+',  '-', ' /',  '=',  '<', ' >', ' ,', '.',  ';', ' (', ')']) then
            mark(col, Dcnt, Pcnt, belongs, word, head)
        else
        if ch in (['A'..'Z', '$']) then
            alphabet(col, Dcnt, Pcnt, belongs, word, head)
        else
        if ch in (['0'..'9']) then
            digit(col, Dcnt, Pcnt, belongs, word, head)
        else
        if ch = '"' then
            string(col, Dcnt, Pcnt, belongs, word, head)
        else
        begin
            write(ch);
            read(ch);
            col := col + 1
        end
    end
  end     (* of  not  eof *)
end;  (* of  procedure  readinput  *)



    (*    Call library function,   logorithm base 2,  to calculate    *)
    (*    eatimated length of a program                                *)

function log(x:real):real;
external;
```

```
      (*  Print the token with its information   in ascending order    *)

procedure print(head : ptr);
begin
   with head^ do
   if head <> nil then
   begin
      print(left);
      if CntOrNot = 'Y' then
         if category = 1 then     (* is operator *)
         begin
            if UseInData > 0 then
            begin
               n1Data := n1Data + 1;
               if token = '"                          ' then
                  UseInData := UseInData  div 2;
               N1Data := N1Data + UseInData;
               n1 := n1 + 1
            end;
            if UseInProc > 0 then
            begin
               n1Proc := n1Proc + 1;
               if token = '"                          ' then
                  UseInProc := UseInProc  div 2;
               N1Proc := N1Proc + UseInProc;
               if UseInData = 0 then
                  n1 := n1 + 1
            end
         end
         else                (* is operand *)
         begin
            if UseInData > 0 then
            begin
               n2Data := n2Data + 1;
               N2Data := N2Data + UseInData;
               n2  := n2 + 1
            end;
            if UseInProc > 0 then
            begin
               n2Proc := n2Proc + 1;
               N2Proc := N2Proc + UseInProc;
               if UseInData = 0 then
                  n2 := n2 + 1
            end
         end;  (* of else *)
      writeln(token, category:3,'     ', CntOrNot, UseInData:9, UseInProc:9);
      print(head^.right)
   end
end;        (*  of procedure  print  *)
```

```
      (* Procedure to calculate numbers of distinct operators  and  *)
      (* operands, total numbers of  operator and operand           *)
      (* occurrence, and the estimated length of a program          *)

procedure calculate;
begin
   N1 := N1Data + N1Proc;
   N2 := N2Data + N2Proc;
   N  := N1 + N2;
   EstLen := n1 * (log(n1)/log(2)) + n2 * (log(n2)/log(2));
end;   (* of precedure calculate *)



   (*    Print out the result of calculation   *)

procedure printresult;
begin
   writeln;
   writeln('Number of distinct operators in  data division is', n1Data:15);
   writeln('Number of distinct operators in procedure division is', n1Proc:10);
   writeln('Number of distinct operators in  the program is', n1:17);  writeln;
   writeln('Number of distinct operands in  data division is', n2Data:16);
   writeln('Number of distinct operands in procedure division is', n2Proc:11);
   writeln('Number of distinct operands in  the program is', n2:18);  writeln;
   writeln('Total operator occurrence in  data division is',N1Data:13);
   writeln('Total operator occurrence in procedure division is',N1Proc:8);
   writeln('Total operator oocurrence in the program is', N1:15);     writeln;
   write('Total operand occurrence in  data division is', N2Data:14);
   write('Total operand occurrence in procedure division is', N2Proc:9);
   write('Total operand occurrence in the program is' , N2:16);       writeln;
   writeln('Total operator & operand occurrence  in the program is',N:5);  writeln;
   writeln('The estimated length of the program is', EstLen:10:3)
end;   (* of procedure printresult *)



begin   (* main program *)
   initialization;
   readdata(f1, WhetherCnt);
   readchange(fchange);
   readinput(word, belongs, head);    writeln;
   writeln(' Number of tokens in data div is', Dcnt:9);
   writeln(' Number of tokens in procedure div is', Pcnt:5);   writeln;
   writeln('                              ',' Count  ','Use in  ','Use in');
   writeln('                              ',' Category  ','or not  ', 'Data   ', 'Proc');
   print(head);
   calculate;
   printresult
end.   (* of main program *)
```

ANALYZING HALSTEAD'S COUNTING RULES IN COBOL

by

MANA HUNG

B. S., National Chengchi University, 1970

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Software complexity measures are measures of how difficult it is to comprehend, modify, and generally maintain a program. The objective is to have a measure that will identify the complexity of a program and will aid in detecting program difficulties; assessing programming techniques; and achieving cost-effective, timely, reliable and high-quality software. The measures of Halstead's software science which is one of the popular software complexity measures, are based on static lexical analyses of the vocabulary of operators and operands as well as the number of occurrences of each class in a computer program. From these measures other quantitative measures for many useful properties of programs can be obtained, such as program length, estimated length, program volume, programming time, and language level.

In this paper, a set of six counting strategies of operators and operands was developed and used to count 45 commercial COBOL programs by the counting program written in Pascal. Several statistical techniques were then used to analyze the outputs of the counting program to see the accuracy of the estimated length derived from the length equation of the software science. The counting strategy sensitivity of the estimated length was also investigated.

The results show that N_{est} as a predictor of N is insensitive to the counting strategy.