A Study on Software Science Models for The Physical Structure of Computer Programs

A THESIS SUBMITTED TO THE JAWAHARLAL NEHRU UNIVERSITY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF PHILOSOPHY

by

P. K. JHA

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI-110067

1981

C E R T I F I C A T E

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110 067

The research work presented in this dissertation has been carried out at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi 110 067.

This work is original and has not been submitted in part or full for any other degree or diploma of any University.

Pranav K. Jha
Student

Prof. M.P. Mukherjee
Dean

Dr. R. Sadananda
Supervisor

Synopsis

Software Science is a field of study related to the structural composition of computer programs. It evolved from the observation that "human factor" is an integral part of any written text and that natural language texts as well as algorithmic texts are governed by certain natural laws. M.H. Halstead has made certain hypotheses in this regard. These have been validated on standard programs written in various programming languages. The essential motivation is to predict the performance of computer software in terms of the time taken to implement it and the number of errors committed while programming.

As part of these studies, several theories have been proposed recently to explain the distribution of operators and operands in computer programs and their inter-relationships. In this thesis, we concentrate on operator models. We review all models proposed till date and then propose modifications to them. The modified models are seen to provide better fit than the old ones. Experiments performed to arrive at these conclusions with a prior description of the basics of Software Science have been explained in detail.

Acknowledgments

At the outset I express my heartiest thanks to my supervisor Dr. R. Sadananda for his excellent guidance and affectionate behaviour.

I owe a sense of gratitude to Dr. K.B. Lakshmanan, Visiting Fellow, NCSDCT (now Asst. Prof., IIT Madras). He encouraged me all along and gave invaluable suggestions.

My special thanks are due to Dr. J.L. Elshoff, General Motors, Warren, Michigan, USA. The data on PL/I programs that have been used for the testing of the models were obtained from him.

Friendship is an invaluable asset. I wonder my friends do not know the extent to which their good words and company invigorate and strengthen me. My heartiest thanks to Jose Abraham, D. Narasimha, G.S.R. Murthy, Ajit Singh, Hari Prakash and Ravi Kant Prasad.

The typing has been done by Mr.D.B. Sawant. He took pains to put it neatly and correctly. I am thankful.

I availed the library and other facilities of JNU, New Delhi, IIT, New Delhi and TIFR, Bombay.

For a part of my M.Phil. course at JNU, I was a recipient of the University Junior Research Fellowship. I am grateful.

New Delhi P. K. J.

Dated 21.2.81

CONTENTS

Chapter I

INTRODUCTION

1.1 What is Software Science?

Software Science is a recently developed field of study concerned mainly with the structural composition of algorithms and their implementations in the form of computer programs. It aims at giving an objective estimate of software quality and complexity. Halstead was the first to put it as a separate discipline. In the preface to his book, "Elements of Software Science" [1], he puts forward an interesting observation :

"Even such intangible objects as written abstracts and computer programs are governed by natural laws, both in their preparation and in their ultimate form".

Actually an idea of this sort was propounded by Zipf [2,3] as early as late forties when he observed that language is, among other things, a biological, psychological and social process. He carried out experiments on natural language texts and found that the words therein, more or less, follow certain specific distribution. He stated that if the words appearing in a natural language text are sorted in the descending order of their frequency of occurrence then the ith most common word has a frequency of approximately,

$$f_i = C/i, \qquad (1.1.1)$$

for some constant C. This, no doubt, is an interesting observation as it opens a new vista of information science. Now, algorithms, too, are comparable to natural language texts to a great extent since the human thought process involved is the same. Moreover, it should make little difference in which language one is thinking or writing. Based on these reasonings, Halstead hypothesized that algorithms may well be characterized by certain laws similar to those in physical sciences. He used a simple count of operators and operands appearing in computer programs and postulated certain empirical relationships among them that form the basis of Software Science. These relationships, in turn, can be used to predict certain interesting and useful properties of algorithms. Some of them (to be elaborated later) include

(a) prediction of programming requirements for proposed software, projects,

(b) prediction of programming time for a normal programmer,

(c) prediction of initial error rates,

(d) quantitative evaluation of programming languages,

(e) effect of modularity, and

(f) a method of measuring the difference between programs written by experts and those by novices.

1.2 Basic Metrics of Software Science

A well-formed program may be regarded as comprising, barring comments, non-executable statements and declaratives,

(a) operands and

(b) operators.

The classification is simple and clear in the case of programming languages like ALGOL, PL/I and FORTRAN. By operands we mean all variables and constants appearing in the executable statements. By operators we mean those which affect the value or ordering of operands. Examples of operators include arithmetic operators, Boolean operators, keywords, and delimeters.

For an algorithm expressed in a programming language, the following are directly measurable and form the basic metrics [1] of Software Science .

n_1 = number of distinct operators,

n_2 = number of distinct operands,

N_1 = total number of operators,

N_2 = total number of operands, and

n_2^* = number of I/O parameters.

From these basic metrics, the vocabulary n is defined as

$$n = n_1 + n_2 ,$$

and the program length N as

$$N = N_1 + N_2 .$$

Further, if we denote

$f_{1,i}$ = number of occurrence of the ith most common operator, where $i = 1,2,\ldots,n_1$, and

$f_{2,i}$ = number of occurrences of the ith most common operand, where $i = 1,2,\ldots,n_2$, then

it is obvious that the following relationships hold :

$$N_1 = \sum_{i=1}^{n_1} f_{1,i} , \qquad (1.2.1)$$

$$N_2 = \sum_{i=1}^{n_2} f_{2,i} , \qquad (1.2.2)$$

and

$$N = \sum_{j=1}^{2} \sum_{i=1}^{n_j} f_{j,i} . \qquad (1.2.3)$$

Software Science attempts to provide quantitative measures to the various predictable properties of an algorithm in terms of the above directly measurable metrics.

1.3 Length Equation

Based on intuitive reasonings, Halstead [1] hypothesized that, given the basic counts n_1 and n_2, the program length can

be estimated as

$$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2 , \qquad (1.3.1)$$

where $\hat{N}$ represents the quantity estimated as against N which denotes the quantity actually observed. It should be noted here that no rigorous mathematical proof to this equation is currently known. However, when put to test in several experiments (e.g., [5,6]), it has been found that $\hat{N}$ is in good agreement with N. It should be remembered that computer programs taken for such experiments in Software Science are all supposed to be polished and free from "impurities" [1].

1.4 Models for the Frequency Distribution of Operators in Computer Programs

The primary hypothesis of Software Science about estimation of program length, viz., equation (1.3.1) opens a new area of investigation as to why such a relationship should hold. Many investigators have tried to answer this question by means of models for the frequency distribution of operators and operands appearing in computer programs. While giving support to the foregoing relationship, such a model may well help one understand the structural composition of algorithms in a better way and thus provide explanation for other observed software phenomena.

In this thesis, we concentrate on the frequency distribution of operators in computer programs. We shall survey, in brief, the various models proposed so far and then attempt to investigate the possible improvement in this area. An elaborate description will follow in the later chapters.

(i) Zipf Model : Zipf law for the frequency distribution of words in natural language texts (equation (1.1.1)) is applied in the case of computer programs. The only quantity used in predicting the distribution is n_1. Zwreben [7], Elshoff [8] and others have carried out experiments on data from programs written in ALGOL and PL/I. Although it does not give a good approximation to the observed distribution, the high correlation coefficient between the predicted distribution and the observed one confirms that this model does give an insight into the nature of distribution of operators in computer programs.

(ii) Bayer Models : Bayer proposed three models [7,9,10]. These concern both operators and operands. Like Zipf model, the only information employed for predicting the frequency distribution of operators is n_1.

(iii) Zwreben Model : It is basically a revised version [7] of the above Zipf model. Zipf law corrected by Mandelbrot is further

modified to suit the behaviour of operators in computer programs. Here, both n_1 and n_2 are used to predict the distribution of operators. Experiments on data from programs written in ALGOL, PL/I and FORTRAN show that this model is superior to the aforementioned ones.

(iv) Elshoff Models : Elshoff [8] has proposed two models, Zipf-A and Zipf-B, each a modified version of the original Zipf model. Each employs information regarding both n_1 and n_2 to predict the distribution of operators in computer programs. Experiments [8] on PL/I data show that Zipf-B model is comparable to the Zweben model above.

(v) Zweben-Halstead Model : This model [11] has recently been reported for PL/I programs. It uses n_1 and, unlike other models discussed above, N_1 to predict the distribution of operators. It gives an excellent prediction in the case of PL/I programs, and is much superior to other models.

In spite of these various models proposed so far, several questions remain unanswered. For example, we stated that the Zweben-Halstead model uses the total count of operators, i.e., N_1 and/or n_2. Now, can the improved performance of the model be attributed to the fact that additional information regarding N_1 is used? Is it possible to modify the previous models to take into consideration the knowledge of N_1, and if so, how do they perform in comparison to the Zweben-Halstead model? Moreover, we also

stated that the Zweben-Halstead model has been validated only for PL/I data. Hence, whether or not the superior performance holds in the case of programs written in other languages also needs investigation.

We have attempted to provide answers to these questions and thus explore the possibility of a new model which presumably could perform better than the existing ones. In this exercise, we have considered the original Zipf model, Zweben model and Elshoff models for further study. Each of these models has been modified to take the information about the total operator count N_1 in order to compare against the Zweben-Halstead model. The original Bayer models have been found unsatisfactory [8] and hence we do not consider them.

The conclusions regarding goodness of fit of various models are based on data derived from programs written in ALGOL and PL/I (see Appendix A for the source of these programs). The bases of judgement are regression analysis and correlation coefficient (see Appendix B for the description) between the observed distribution and the predicted distribution in each case.

Our findings are basically the following :

The Modified Zipf and Modified Zipf-B models show considerable improvement over the old models. Both compare well

with the Zwoben-Halstead model which has been seen to be the best among existing models. The Modified Zipf-A model, as will be explained in Chapter 4, simply coalesces with the Modified Zipf model. Regarding Modified Zwoben model, its behaviour, unlike others, does not show any improvement, rather it shows a decline. Between Modified Zipf and Modified Zipf-B models, the latter is seen to be better, considering the results of the experiments on the data of both ALGOL and PL/I programs. This result about Modified Zipf-B model giving the best fit is the most important finding of our research. Finally, to increase the scope of validation, these models may be put to test on programs written in other higher level languages. What we have established is the suspected relationships amongst the measurable quantities in computer programs and their tendency towards following predictable distributions.

1.5 Organisation of the Thesis

This thesis consists of five chapters. Chapter 1, as described so far, is an introduction to the field of study and the specific problem aimed at. Together with a brief survey of the work done till date in this area, it discusses the possibility of further research. The results arrived at and the conclusions drawn therefrom are described in short.

Chapter 2 describes, in detail, Software Science and its potential applications. We want to state here that those readers interested mainly in the frequency distribution of operators in computer programs may well skip this chapter. It consists of three sections. The first one describes the so-called "Impurity Classes" appearing in computer programs. The second section describes, in detail, the measurable properties of computer programs which are further used in understanding the applications of Software Science. The last section discusses these potential applications in the area of software quality and complexity.

Chapter 3 is relatively small. It describes the strategy adopted in counting the operators and operands appearing in computer programs. The description includes the programs written in ALGOL, PL/I and FORTRAN.

Chapter 4 is a clear description of the main problem of the thesis, i.e., the models for the frequency distribution of operators in computer programs. It consists of two sections. Section 1 is an elaborate description of all existing models proposed till date. In addition, it explains the scope of possible investigation in this area. Section 2 describes the ideas developed in course of this investigation. We arrive at modified versions of the existing models.

Chapter 5 is the last chapter which discusses the results obtained and comments thereon. This consists of three sections. The first one describes the testing strategy and gives tables of n_1, n_2 and N_1 of all ALGOL and PL/I programs taken for experimentation. Also given are the sample results concerning the frequency distribution of operators as predicted by the various models against observed distribution. Section 2 discusses the results obtained and the conclusions drawn. Section 3 comments on the findings and discusses the scope of further study in this area. This is followed by the references of the various books and papers. The appendices at the end contain, respectively, the source of the sample programs, regression analysis and correlation studies, results obtained and the listing of the program.

Chapter II

SOFTWARE SCIENCE AND ITS POTENTIAL APPLICATIONS

In Chapter 1, we described the basic metrics of Software Science and the empirical expression for program length in terms of them. Here we shall describe other measurable properties of algorithms which are later used in exhibiting how Software Science aims at estimating the software quality and complexity. Before initiating such a description, we want to stress on the following important point. The programs taken for study in all experiments of Software Science are supposed to be well-written. To distinguish such programs from so-called "impure" ones, Halstead [1] identified six classes of impurity. The description follows.

2.1 Impurity Classes

(i) Complementary Operators : Two complementary operators, e.g., "add" and "subtract" are applied successively to the same operand. Obviously, an impurity of this type may arbitrarily increase the program size without altering the program vocabulary.

(ii) Ambiguous Operands : The same variable-name is used to represent two or more operands in a program. Such usage reduces the comprehensibility of a program.

(iii) Synonymous Operands : Two or more variable-names are used to represent the same operand. This unnecessarily increases

the program size. It is complement to the above impurity.

(iv) Common Subexpressions : The same subexpression, i.e., combination of terms, appears more than once. Such a subexpression may well be replaced by a local variable or a common procedure.

(v) Unnecessary Assignments : A subexpression is assigned to a local variable which is used only once. This obviously serves no purpose as the subexpression does not appear anywhere else in the program. It is complement to the above impurity.

(vi) Unfactored Expressions : An expression that can be factored is left unfactored affecting clarity of the program. This impurity however, is different from the previous ones in the sense that no mechanical method of removing it is known.

In each of the above cases, Halstead [1] has shown that the impurities do corrupt the programs and considerably distort the measurable properties relevant to Software Science.

2.8 Measurable Properties of Computer Programs

We shall describe some useful measurable characteristics of computer programs in terms of the basic metrics, viz., n_1, n_2, N_1, N_2, N and n_2^* described in Chapter 1. The definitions that follow are as by Halstead [1].

(i) Program Volume : It is defined as

$$V = N \log_2 n . \qquad (2.2.1)$$

For a vocabulary of n elements, the minimum number of bits required to represent each element uniquely is $\log_2 n$. Thus, V refers to the total number of bits required to represent the program.

It depends on the language of implementation and is lower in the case of a higher-level language.

(ii) Potential Volume : It is defined as the volume of the implementation employing the most compact language, presumably hypothetical. In such a language, the function to be performed is supposed to be already defined and a simple call will be required. In equation form,

$$V^* = (N_1^* + N_2^*) \log_2 (n_1^* + n_2^*) , \qquad (2.2.2)$$

where all quantities with asterisk represent the corresponding minimal value of those parameters. Now, in the minimal case,

$$N_1^* = n_1^* \text{ and}$$

$$N_2^* = n_2^* .$$

Furthermore, the total number of operators in this case will be only two; one for the name of the function or procedure and the

other serving as an assignment or grouping symbol. Therefore, $n_1^* = 2$. n_2^*, of course, is the number of I/O parameters. Equation (2.2.2) thus becomes

$$V^* = (2 + n_2^*) \log_2 (2 + n_2^*) \qquad (2.2.3)$$

It should be noted that V^* is an intrinsic characteristic of an algorithm - independent of the language employed in the computer program. It appears in the definitions of program level and language level described below.

(iii) Program Level : Since there may be more than one implementations of an algorithm in a given programming language, we define program level as a measure of distinction among various implementations. It is defined as

$$L = V^*/V . \qquad (2.2.4)$$

For a given algorithm, the program written by an expert will presumably have a smaller volume and hence a larger program level than the one written by a novice.

To have an empirical formula for the program level in terms of the known parameters, Halstead [1] argued that the program level should increase with the number of distinct operands n_2 and decrease with both the number of distinct operators n_1 and the total number of operands N_2. He, therefore, proposed the following

program level estimator :

$$\hat{L} = \frac{n_1^*}{n_1} \cdot \frac{n_2}{N_2},$$

or,
$$\hat{L} = \frac{2}{n_1} \cdot \frac{n_2}{N_2} \qquad (2.2.5)$$

This empirical equation was put to test for various standard CACM programs [1]. The good statistical results validate this relationship.

(iv) Language Level : For a given programming language, when different algorithms are considered, it is observed that as V^* increases, L decreases proportionately so that the product LV^* remains fairly constant for a given language. This product is defined as the language level λ so that

$$\lambda = LV^*. \qquad (2.2.6)$$

Halstead [1] considered some English texts and a variety of programs written in different programming languages and computed the value of λ in each case. The table follows.

Language	λ	Variance
English	2.16	0.74
PL/I	1.53	0.92
ALGOL 58	1.21	0.74
FORTRAN	1.14	0.81
PILOT+	0.92	0.43
Assembly	0.88	0.42

+ Purdue Instructional Language for Operating Systems and Translators

With the increasing effort made in the recent past towards devising new programming languages, λ may serve as a good measure of determining the programming productivity expected by employing a certain language.

(v) Programming Effort : It is a measure of the mental effort required to implement an algorithm in a given programming language. We note that the volume V (equation (2.2.1)) of a program may be regarded as a measure of the total number of mental comparisons required to generate the program if we take binary search as the basis of selection. Further, each mental comparison requires a number of elementary mental discriminations which define the difficulty of the task. Recalling the definition of the program level L (equation (2.2.4)), we may take 1/L as the measure of this difficulty. Thus the total number of elementary mental discriminations, termed as programming effort E, may be defined as

$$E = V/L \qquad (2.2.7)$$

Using equation (2.2.4), i.e., $L = V^*/V$,

$$E = V^2/V^* \qquad (2.2.8)$$

This shows that the mental effort required to implement an algorithm should vary as the square of its volume in any language.

E may be represented in the following other forms also.

$$E = V/L = V^*/L^2 = (V^*)^3/(LV^*)^2 = (V^*)^3/\lambda^2. \qquad (2.2.9)$$

Here, use has been made of the definition of the language level λ, i.e., equation (2.2.6).

Since V^* is fixed for a given algorithm, the last equality in the above equation shows that higher-level languages should reduce the effort of programming.

(vi) Programming Time : The programming time anticipated in a proposed software project may be an important estimate to make. It is defined as the ratio between the programming effort E and the speed S of the programmer. Thus the time-estimator may be expressed as

$$\hat{T} = E/S . \qquad (2.2.10)$$

Halstead [1] has made use of a concept developed by a psychologist John Stroud in a paper titled "The Fine Structure of Psychological Time". Stroud has suggested that

$$S \simeq 18 \text{ discriminations per second}$$

for a person with normal mental concentration.

Another form of $\hat{T}$ may be the following :

$$\hat{T} = E/S = V/(SL) . \qquad (2.2.11)$$

Substituting for V and L from equations (2.2.1) and (2.2.5), $\hat{T}$ may be expressed in terms of the basic metrics as follows :

$$\hat{T} = \frac{n_1 N_2 N \log_2 n}{2 S n_2} \qquad (2.2.12)$$

Further, N from equation (1.3.1) may be substituted for N giving

$$\hat{T} = \frac{n_1 N_2 (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 n}{2 S n_2} . \qquad (2.2.13)$$

With S taken as 18/second, all other parameters to the right are directly measurable for any computer program.

Having defined the various measurable properties of computer programs, we want to state the following. Given a clear description of a problem and the language to be employed, i.e., knowing n_2^* and λ , it is possible to estimate the basic metrics of the computer program to be written. Halstead [1] has derived such relationships. We shall simply state them.

$$n^* = 2 + n_2^* . \qquad (2.2.14)$$

$$n_2 = n_2^* (\log_2 (n^*/2)) (n_1 - 2)/n^* + n_2^* \qquad (2.2.15)$$

$$(n^* \log_2 n^*)^2 = \lambda (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 (n_1 + n_2). \qquad (2.2.16)$$

As is apparent, an iterative solution may be required to get the values of n_1 and n_2 knowing λ and n_2^*.

The importance of the above solution lies in the fact that an approximate estimate of the important measurable properties of the algorithm can be made before actually writing the program.

2.3 Potential Applications of Software Science

In the light of the discussions in the previous sections about various measurable properties of computer programs, we shall describe some possible applications of Software Science.

(i) Estimation of Programming Time : From equation (2.2.9), we know that

$$E = (V^*)^3/\lambda^2 .$$

Now, V^*, which is a characteristic property of an algorithm, is directly expressible in terms of n_2^* (equation (2.2.3)). Thus under the conditions that

(a) a clear description of the problem is available,

(b) the programmer is fluent in the selected programming language, and

(c) high concentration exists,

equation (2.2.13), viz.,

$$\hat{T} = \frac{n_1 N_2 (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 n}{2 S n_2}$$

may be used to predict the programming time.

Several experiments (e.g., [1,12,13]) have been performed

to test the validity of this estimator. In one experiment [18], a dozen CACM algorithms were each written in three different languages - FORTRAN, PL/I and APL. The actual times taken had the following correlation coefficients with the corresponding predicted times :

FORTRAN	0.87
PL/I	0.94
APL	0.93

These figures seem to validate the theory.

(ii) Estimation of Initial Error Rates : An important part of software development consists in debugging or detecting and removing the errors introduced during the initial writing phase. Consequently, a knowledge about initial number of errors to be expected in a given project may be very useful. One practical usefulness of Software Science lies in the fact that it provides a quantitative idea of the possible initial errors in a computer program.

As has been shown, the amount of time required to implement a computer program depends on the number of elementary mental discriminations E that it requires. It, therefore, follows that the number of points at which it is possible to make an error should also be related to the value of E. Under the assumption that human brain can handle, i.e., produce a result from, five "chunks" in its memory,

Halstead [1] has derived the following estimator $\hat{B}$ for the number of bugs in a given implementation :

$$\hat{B} = E^{2/3}/3000 . \qquad (2.3.1)$$

Positive statistical results [1] have established the close correlation between the effort E and the number of bugs in a given untested implementation. However, it should be noted that it predicts only the initial number of errors, it does not converge as these errors are removed.

(iii) Evaluating Modularity Concepts : Software Science is useful in analysing programming methodology principles that are concerned with modularity [1,15]. The Effort measure E provides a quantitative answer to this question. We recall the definition of E from equation (2.2.8), i.e., $E = V^2/V^*$. Since sum of the squares is less than the square of the sum, it is apparent that the programming effort reduces for partitionable programs.

Halstead [1] has discussed the following four approaches that characterize modularity :

(a) equalisation of length,

(b) minimization of the modular potential volume,

(c) limiting length of error-free programs, and

(d) psychological concept of "chunks".

Baker and Zweben [15] carried out experiments on the published data from 500 programs. They wanted to study the conditions

under which modularization is beneficial. They found that the Effort measure E gives a quantitative idea to this effect.

(iv) Measuring the Difference between Programs Written by Experts and Those by Novices : This application is quite straightforward, though obviously very useful. Program level L is an effective measure for this purpose. Also, it gives an idea about the possible improvement in a program.

However, it is worth mentioning here the results of an experiment by Johnston and Lister [16]. They wanted to investigate

(a) whether Software Science measures could form a reliable basis for automatic grading of student programs,

(b) the accuracy of certain approximations in Software Science, and

(c) the language level of PASCAL in the sense of "high" or "low" as quantified by Software Science.

A notable feature of the experiment was the large number of programs measured : about 13000 programs written by over 500 students. The results of the experiments, as Johnston and Lister claim, are not encouraging for Software Science metrics to be taken as a measure of software quality and complexity. The findings are the following :

(a) The correlation between the effort measure and the tutors' assessment of merit and clarity was small,

(b) The reliability of the length estimator N was good. However, the level estimator $\hat{L}$ was not very reliable - it showed considerable discrepancy from L, and

(c) λ was not constant for PASCAL.

However, as the authors themselves admit, there could be valid reasons for the negative results. Some of these include

(a) student programs often contain impurities and they do not always use language features to best advantage,

(b) the marking scheme was not finely grained, and

(c) it was difficult to compute V^* correctly.

(v) Measuring the Psychological Complexity of Software Maintenance Tasks : Software maintenance is as important as the development. In fact, resources required for maintenance are found to be higher than those for development. Software Science may be used to provide an initial estimate of the reliability of modifications and/or the time required to implement them. Curtis et.al.[17] employed Halstead and McCabe [18] metrics in two experiments to investigate the following problems :

(a) understanding an existing program, and

(b) implementing modifications to it.

It was found that the Halstead metrics, like McCabe metrics, provided adequate information regarding the difficulty programmers might experience in understanding and modifying an existing software.

(vi) Quantitative Analysis of English Prose : As has been described earlier in Section 1.1, the basis of Software Science is the "human factor" in the written composition and that it is independent of a particular language. However, interestingly enough, similar relationships are seen to hold in the case of natural languages also [1]. This gives greater support to this field of study in a more general context. Just as a computer program is separable into operators and operands, English prose may also be divided into two classes :

(a) function words, and

(b) content words.

Function words comprise articles, pronouns, propositions, conjunctions and auxiliary verbs while content words include nouns, verbs, adjectives and adverbs.

Halstead [1] carried out experiments on standard English texts to examine the applicability of Software Science there. Taking into consideration some of the inherent differences between natural languages and computer languages, he found the results to be all positive, supporting the present study.

We sum up this chapter with the following comment by Fitzsimmons and Love [14] :

"Software Science is much like actuarial studies of populations : the results make it possible to predict gross properties

of the whole. Thus the Software Science measures may be inaccurate when applied to individual programs, but they become more accurate when applied to large number of programs, such as are found in large software development projects."

Chapter III

THE COUNTING STRATEGY

As stated earlier, the various properties of Software Science are defined in terms of the following basic metrics :

(a) number of unique operators n_1,

(b) number of unique operands n_2,

(c) total count of operators N_1,

(d) total count of operands N_2, and

(e) number of I/O parameters n_2^*.

Therefore, before discussing the experiments on standard programs and the conclusions drawn, it is imperative to describe clearly the counting procedure adopted. Our description about the counting strategy will mainly concern the constructs of ALGOL, PL/I and FORTRAN. However, the procedure followed may well be extended, with minor revisions, to the programs in other languages, e.g., PASCAL.

Basically, the counting program may include a lexical analyzer and a parser - similar to those found in a compiler. It recognises the symbols, constants, variable names and keywords and parses the various statements. Karl J. Ottenstein [19] has written a program in ANSI-FORTRAN to count operators and operands in ANSI-FORTRAN modules. Specifically, the basic rules followed are

as under :

(i) Materials extraneous to the pure algorithm, i.e., comments, declaratives, non-executable statements etc. are ignored.

(ii) All arithmetic, Boolean and replacement operators are counted.

(iii) Function names including built-in ones are counted as operators.

(iv) The grouping operators, viz., BEGIN-END pair, DO-CONTINUE and the grouping paranthesis "( )" are treated alike as an operator.

(v) The subscripting operator "[ ]" is treated distinct from the grouping operator.

(vi) The initialization statements of ALGOL and PL/I programs, as opposed to DATA statements in FORTRAN, are considered as operators.

(vii) IF is treated as an operator. However, when IF..THEN.. ELSE appears, ELSE is taken as a separate operator. Same applies to operators like FOR..UNTIL..DO.

(viii) A label is not an operand.

(ix) GOTO L_1, say, is an operator distinct from, say, GOTO L_2. A computed GOTO statement, e.g., GOTO(L_1,L_2,...) i, i = 1,2,... is taken as consisting of i GOTO's.

(x) An ASSIGN statement for an assigned GOTO is counted as a replacement operator.

(xi) A comma appearing in all executable statements, e.g., as a separator in an argument list, is counted as an operator.

(xii) All constants are mapped into real numbers and treated as any other operand. The boolean constants .TRUE. and .FALSE. are mapped to 1.0 and 0.0 respectively. Hollerith constants are ignored since they are permitted only in DATA statements.

(xiii) Statement functions are ignored.

(xiv) Positional notation denoting the end of a statement, e.g., " ; " in PL/I is counted as an EOS (End Of Statement) operator.

(xv) In I/O statements, only the keywords which are operators and the referenced variables which are operands are counted.

(xvi) CALL-entry point is treated in a fashion similar to GOTO-label, i.e., CALL A and CALL B are counted as separate operators.

(xvii) Unless they are involved in a separate impurity class [20] to which the following three rules apply, all variables and constants in FORTRAN programs are counted as operands.

(xviii) Formal parameters are not considered unique because they are synonymons with actual parameters which are recognized as

distinct in the calling program. Their occurrences contribute to N_2, but not to n_2.

(xix) Repeated uses of a given array variable with the same index are considered to be common subexpressions and counted as occurrences of a temporary variable replacing that subscripted variable.

(xx) Ambiguity in the use of local variables which have the same names in different subprograms is resolved by the counting algorithm.

All rules above except the last three may be taken as axiomatic or intuitively obvious. The last three [6] result from the fact that the FORTRAN language does not provide suitable mechanisms by which the programmer can avoid all occurrences of common subexpressions, ambiguous operand usage and synonymous operand usage.

It is worthwhile stating here that the rules above may not be complete in themselves and that there may be differences among programmers regarding a few of these rules. However, such differences will be minor and it is expected that the results about the same program by different persons will not be considerably different from one another.

Chapter IV

MODELS FOR THE FREQUENCY DISTRIBUTION OF OPERATORS IN COMPUTER PROGRAMS

In Chapter 1, we described, in brief, the need for and development of models for the frequency distribution of operators and operands in computer programs. Here we would elaborate it in full detail concentrating more on the operator distribution.

We recall the primary hypothesis of Software Science about the length estimate of a computer program (equation (1.3.1)), viz.,

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 .$$

This relationship has been validated on data from standard computer programs (e.g.,[1,6]). However, no rigorous mathematical proof is currently known. This, therefore, leaves an open question as to why such an equation should hold. An interesting area of investigation may be to analyse the characteristics of operators and operands and predict their frequencies of distribution in computer programs. The basic entities, viz., n1, n2, N1, N2 etc. may be used as parameters for this purpose. In addition to providing support to the above relationship, a study to this effect may explain other observed software phenomena also and thus help understand the structural composition of algorithms in a better way. Workers in this field have suggested models for the frequency distribution of operators in

computer programs. We shall describe them all and then present new models which evolved after exploring the possibility of improving the existing ones. We have carried out experiments on standard ALGOL and PL/I programs (for source, see Appendix A) to have a comparative study of all models including the new ones. The performance of the various models has been judged on the basis of the regression analysis and the correlation coefficient (see Appendix B for the description) between the actual distribution and the predicted distribution in each case.

4.1 Existing Models

(1) Zipf Model : Recalling the fact that the human thought process involved in the composition of computer programs is more or less similar to that in the natural language texts, Zipf law for word frequencies in natural language texts may well be extended for examination in the case of algorithmic texts. Zipf law in its original form (equation (1.1.1)) states that the ith most common word in a natural language text occurs with a frequency equal to C/i, for some constant C. This may be adopted for the case of computer programs if one of the frequencies is assumed to be known, thus enabling the estimation of the constant C. From observation, we may say that the least frequent operator in a computer program appears only once, i.e. $f_{1,n_1}^{*} = 1$. This gives, for Zipf model, $C = n_1$. Thus the model in

---

* Starting from Chapter 1, we follow the convention of denoting the operator frequency by $f_{1,i}$ and operand frequency by $f_{2,i}$.

mathematical form is

$$f_{1,i} = n_1/i\,, \qquad (4.1.1)$$

$$i = 1,2,\ldots,n_1.$$

Actually, Zipf model was tested [7] for the distribution of both operators and operands, but it was found to be reasonable only in the case of operators. This is not surprising when we recall that Zipf law was originally given for the word frequencies in natural language texts and that there exists a similarity between words in natural language texts and operators in algorithmic texts. It is the following. The grammar of the natural language prohibits certain parts of speech from occurring together, just the same way certain combinations of operators are not allowed in computer programs.

When tested on ALGOL programs [7], which are of relatively small size, this model is seen to perform well except for the fact that, in most cases, it tends to overpredict the more frequently occurring elements and underpredict the less frequent ones. When tested on PL/I programs [8], which are of relatively large size, the performance is not that good. However, the reasonably high correlation coefficient between the actual distribution and the predicted one in either case suggests that this model does give an insight into the behaviour of operators in computer programs and

that there is a scope of further improving it. In fact, most of the later models are modifications to this model only.

(ii) Bayer Models : There are three models due to Bayer [9,10].

(a) Model A : It does not distinguish between operators and operands. It is assumed that a computer program consists of $n(= n_1 + n_2)$ total distinct entities and that the program may be generated by randomly choosing one of these n elements, with possible replacement, until each has been chosen at least once.

(b) Model B : It recognises operators and operands separately. It is assumed that the operators and operands tend to alternate in a computer program and that generation of the program consists in first randomly choosing one of the operands and then alternately choosing operators and operands, with possible replacement, until each has been selected at least once.

(c) Model C : It is a further modification to Model B. It makes use of the property of elimination of common subexpressions in computer programs. If a combination of terms appears more than once in a computer program, all but one occurrence of such a subexpression may be replaced by a temporary variable or procedure. At any stage, the operator-operand pair is such that there are no subsequences of operand-operator-operand triples with two or more disjoint occurrences.

For Model A, the expected frequency of the elements and the expected length of the program are shown [7] to be given respectively by

$$E_A(f_{n-i}) = 1 + \sum_{j=1}^{i} \frac{1}{j}, \qquad (4.1.2)$$

where the sum is defined to be 0 when $i = 0$, and

$$E_A(N) = n \sum_{i=1}^{n} \frac{1}{i} \qquad (4.1.3)$$

As is evident, the expected program length is the same as that predicted by Zipf model.

When tested [7] on the same first 14 CACM ALGOL programs, it has been found that Bayer model A, unlike Zipf model, tends to underpredict the more common elements and overpredict the less common ones.

As regards models B and C which do distinguish between operators and operands, the following also should hold good :

$$N_2 = N_1 + 1 .$$

However, from tests on the same set of CACM programs, it was found that this equation does not hold true for any of the algorithms. But one good characteristic of Bayer modes as reported by Halstead and Bayer [14] is that the lengths of programs predicted by each of these models agree reasonably well with those predicted by equation (1.3.1).

A remark here is worthwhile. Zipf model and Bayer models do provide some insight into the frequency distribution of operators in computer programs though they are not adequate to be considered as good models. It should be noted that our experiments on data from standard programs do not include any of the Bayer models. The reason for this is that they do not compare [8] favourably with the observed distribution and are, therefore, of historical importance only.

(iii) Zweben Model : Zweben [7], after proper examination of a few possible models including the above-mentioned ones, found that Zipf model could be taken as the basis of further investigation. He observed that the discrepancies appearing at the high frequency end in the case of Zipf model could be reduced to advantage by introducing Mandelbrot's correction to Zipf law in the case of word frequencies in natural language texts. Mandelbrot had suggested that, for small i,

$$f_{1,i} = f_{1,1} \frac{1+m}{i+m}, \qquad (4.1.4)$$

$$i = 1,2,\ldots,n_1,$$

in the case of words in natural language texts, where m is a parameter whose value depends on the way the words are encoded and is usually about 1.

However, Zweben observed that no clear distinction can be

made in this equation between small and large values of i. In order to remove this ambiguity, he defined m in the above equation as

$$m = \frac{n_1 - i}{n_1} , \qquad (4.1.5)$$

$$i = 1,2,\ldots,n_1 .$$

As is obvious, m is significant for small values of i only and as $i \to n_1$, $m \to 0$ so that the equation (4.2.4) approaches Zipf law.

Finally, Zweben considered the effect of the operands on operator frequencies. Since, for most algorithms, the number of distinct operands exceeds that of the distinct operators, this effect is accounted for by multiplying each operator frequency by n_2/n_1 whenever this ratio exceeds unity.

Keeping in view all arguments above, Zweben put forward the model in the following form :

$$f_{1,i} = \frac{1 + m}{i + m} f_{1,1} , \qquad (4.1.6)$$

$$i = 1,2,\ldots,n_1 ,$$

where $f_{1,1} = \max(n_1,n_2)$ and m is given by equation (4.1.5) above.

When tested on the same set of 14 CACM ALGOL programs,

Zweben model is seen to give better prediction than the previous models. This is apparent from the positive results of regression analysis and correlation coefficient. Zweben tested it for programs written in other languages, viz., FORTRAN, COMPASS (the Assembly language of CDC 6500) also and found it to be reasonably good, at least better than the previous ones. We find similar fit in the case of PL/I programs also.

(iv) Elshoff Models : Elshoff [8] put forward two models by refining Zipf law in his own way.

(a) Zipf-A Model : It states that

$$f_{1,i} = \max(n_1, n_2)/i, \qquad (4.1.7)$$

$$i = 1,2,\ldots,n_1.$$

(b) Zipf-B Model : It assumes that the original expression due to Zipf should have an additional factor c to account for the discrepancies at the end points of the distribution. Elshoff thus proposed that

$$f_{1,i} = c \cdot n_1/i, \qquad (4.1.8)$$

$$i = 1,2,\ldots,n_1,$$

where c could be such that, in addition to reducing the anomalies at the end points, for each intermediate point in the distribution, c is linearly related to the difference between n_2 and n_1. He

suggested that

while $n_1 < n_2$,

$$c = \frac{1}{n_1}\left\{n_2 - \frac{(n_2 - n_1)(i - 1)}{(n_1 - 1)}\right\}, \qquad (4.1.9)$$

$i = 1,2,\ldots,n_1$.

However, Elshoff is not explicit about the case when $n_1 > n_2$, and the above expression for c cannot be taken in this case. Following the arguments presented by Elshoff, a value of $c = 1$ appears reasonable for $n_1 > n_2$. Thus, the Zipf-B model takes the following forms :

$$f_{1,i} = \frac{1}{i}\left\{n_2 - \frac{(n_2 - n_1)(i - 1)}{(n_1 - 1)}\right\}, \qquad (4.1.10a)$$

while $n_2 > n_1$, and

$$f_{1,i} = n_1/i , \qquad (4.1.10b)$$

while $n_1 > n_2$,

$i = 1,2,\ldots,n_1$.

Clearly, this model approximates the end-point features of the Zeben model for the more commonly used operators and the Zipf model for the less frequently used operators.

Elshoff [8] tested both of these models on PL/I data against all other existing models and arrived at good statistical

results, especially in the case of Zipf-B model which compares well with Zweben model. Actually, in some cases of PL/I programs, Zipf-B model is seen to be better than Zweben model. It is apparent from the results of our own experiments.

(v) Zweben-Halstead Model : This model was first reported [11] for PL/I programs. For the first time, a knowledge about the total operator count N_1 in addition to the number of distinct operators n_1, was used to advantage. Zweben and Halstead employed an "Algorithmic Search Procedure for Program Generation"[25] to determine the functional relationship between the operator frequencies and their ranks. The procedure works in two steps :

(a) estimation of the frequency of the most common operator, and

(b) estimation of the frequency of all other operators in terms of the most common one and the rank i.

The final form of the equations pertaining to this model follows :

$$f_{1,i} = 2^{X_1 \cdot 2^{-(i-1)/X_1}}, \qquad (4.1.11)$$

$$i = 1,2,\ldots,n_1,$$

where $X_1 = \log_2 f_{1,1}$ and is obtained from

$$N_1 = \sum_{i=1}^{n_1} f_{1,i}, \text{ i.e.,}$$

$$N_1 = \sum_{i=1}^{n_1} 2^{X_1 \cdot 2^{-(i-1)/X_1}} . \qquad (4.1.12)$$

As is seen, the above equation may require an iterative solution to get X_1 from known n_1 and N_1. Once X_1 is thus calculated, the distribution can be determined from equation (4.2.11) above. The information about the total count of the operators, i.e., N_1 puts additional constraint on the distribution such that the total count of the operators predicted thus, equals that actually observed. This feature is absent in all previous models and presumably aids in the statistical results here.

When tested [11] on the same set of Elshoff's PL/I programs, this model is seen to give an excellent fit.

4.2 New Models

A close scrutiny of the Zweben-Halstead model raises two important questions :

(a) As is seen, is the superior performance limited to PL/I programs only?

(b) This model, unlike previous ones, employs the knowledge of N_1 also. Can the improved performance be attributed to this

additional information? What happens when other models, too, are modified to take this information?

We have tried to investigate these questions keeping in mind the possible evolution of new model(s). The models taken for further study are

(a) Zipf model,

(b) Zweben model, and

(c) Zipf-B model.

Regarding Zipf-A of Elshoff, as will be seen shortly, modification to it leads to the same model obtained by modifying the original Zipf model. We have maintained the spirit of the original models in each case. The modification consists mainly in introducing the information about the total operator count N_1. This is done by putting the following constraint :

$$N_1 = \sum_{i=1}^{n_1} f_{1,i} \quad . \qquad (4.2.1)$$

Although it is straightforward, still none of the aforementioned models, except the Zweben-Halstead model, has got this property.

We first recall equation (4.1.1), i.e., $f_{1,i} = n_1/i$ corresponding to Zipf model. When it is modified under the constraint of equation (4.2.1) above, the model obtained will

have the form

$$f_{1,i} = C_{ZPF}/i \,, \qquad (4.2.2)$$

where C_{ZPF} is given from

$$C_{ZPF} = N_1 / \sum_{i=1}^{n_1} \frac{1}{i} \,. \qquad (4.2.3)$$

We note here that Zipf-A model of Elshoff, under the above constraint, will coalesce to this form only.

Next, we recall equation (4.1.6) corresponding to the original Zweben model. The constraint of the equation (4.2.1) would yield the following form of the Modified Zweben model :

$$f_{1,i} = \frac{(1+m)}{(i+m)} C_{ZWM} \,, \qquad (4.2.4)$$

$$\text{where} \qquad C_{ZWM} = N_1 / \sum_{i=1}^{n_1} \frac{(1+m)}{(i+m)} \,. \qquad (4.2.5)$$

m, of course, is the same as before, i.e.,

$$m = (n_1 - 1)/n_1 \,.$$

Lastly, we recall equation (4.1.8) or, for that purpose, equations (4.1.10a) and (4.1.10b) corresponding to the original Zipf-B model due to Elshoff. Here the modification may be introduced as follows. We define a factor α such that

$$\alpha = N_1 / \sum_{i=1}^{n_1} f_{1,i(org)} \,, \qquad (4.2.6)$$

$$i = 1,2,\ldots,n_1,$$

where $f_{1,i(org)}$ correspond to the original Zipf-S model. The modified version, in this case, is obtained by simply multiplying each frequency of the original distribution by the factor α. In fact, modifications in the former two cases are also equivalent to multiplying the respective original distributions by a factor similar, in each case, to α.

The results obtained for the existing as well as new models will be presented in the next chapter. The conclusions drawn will be apparent from the values of the regression coefficients and the correlation coefficient. Comments on the present study and the scope of further research in this area will also be discussed in the next chapter.

Chapter V

RESULTS AND COMMENTS

5.1 Testing the Models on ALGOL and PL/I Programs

The existing models, viz., Zipf, Zweben, Elshoff's Zipf-A & Zipf-B and Zweben-Halstead together with the new ones, viz., Modified Zipf, Modified Zweben and Modified Zipf-B are used to predict the operator distributions for 14 ALGOL programs and 34 PL/I programs. The data concerning n_1, n_2 and N_1 appear in Tables 1A and 1B for the ALGOL and PL/I programs respectively. The typical observed distributions of the operators in one each of the ALGOL and PL/I programs appear respectively in Tables 2A and 2B. The distributions as predicted by the various models for the same two sample programs are shown in Tables 3A and 3B. We have calculated the intercept and slope of the least square line and the correlation coefficient corresponding to each model. These detailed statistics for all ALGOL and PL/I programs appear respectively in Appendices C and D. The mean and standard deviation of the intercept, slope and correlation coefficient for the ALGOL and PL/I programs are shown respectively in Tables 4A and 4B. Finally, the listing of the program written for the testing of the models appears at the end of the Appendices.

Table 1A

Observed Parameters of ALGOL Programs

Program #	Eta1 (η_1)	Eta2 (η_2)	N1
1.	11	18	53
2.	13	7	45
3.	20	39	271
4.	20	12	61
5.	17	12	64
6.	14	12	56
7.	9	8	30
8.	15	16	74
9.	20	26	142
10.	10	10	26
11.	10	10	29
12.	12	10	34
13.	12	9	34
14.	13	22	92

Table 1B

Observed Parameters of PL/I Programs

Program #	Eta1 ($\hat{\eta}_1$)	Eta2 ($\hat{\eta}_2$)	N1
1.	33	161	930
2.	46	318	1678
3.	50	214	892
4.	45	195	782
5.	96	661	3579
6.	95	539	3036
7.	45	73	417
8.	63	299	1579
9.	43	343	1529
10.	54	501	2685
11.	51	522	2616
12.	37	624	3065
13.	185	746	4539
14.	37	163	600
15.	90	271	1794
16.	73	387	1818
17.	28	62	381
18.	32	81	612
19.	26	66	438
20.	33	135	647
21.	37	99	556
22.	48	483	2738
23.	53	450	3024
24.	26	110	490
25.	22	61	277
26.	23	31	150
27.	22	33	175
28.	19	25	119
29.	24	64	340
30.	24	103	395
31.	51	267	1269
32.	31	161	869
33.	28	109	339
34.	41	530	2799

Table 2A

Sample Observation

Distribution of Operators in ALGOL Program #3

Sl.No.	Operator	Frequency
1.	assignment	52
2.	BEGIN-END or ( )	47
3.	semicolon	38
4.	(subscript)	32
5.	-	20
6.	+	15
7.	IF	12
8.	*	11
9.	<	10
10.	/	8
11.	FOR ...	7
12.	ABS	7
13.	GOTO NEXT	4
14.	SQRT	2
15.	ENTIRE	1
16.	GOTO STEP	1
17.	**	1
18.	> =	1
19.	=	1
20.	GOTO OUT	1

$n_1 = 20, \quad N_1 = 271$

Table 2B

Sample Observation

Distribution of Operators in PL/I Program #7

Sl.No.	Operator	frequency
1.	semicolon	111
2.	(subscript)	53
3.	assignment	49
4.	DO	27
5.	IF	25
6.	EQ	21
7.	TO-BY	15
8.	(expression)	14
9.	(argument)	13
10.	binary -	11
11.	binary +	10
12.	commas	8
13.	*	4
14.	GT	4
15.	unary -	4
16.	ELSE	4
17.	initial	4
18	|	3
19.	CALL P001	3
20.	CALL P002	3
21.	LT	2
22.	NE	2
23.	ON	2
24.	WRITE	2
25.	READ	2
26.	CALL P003	2
27.	**	1
28.	&	1
29.	BEGIN	1
30.	OPEN	1
31.	CLOSE	1
32.	PUT	1
33.	GOTO A001	1
34.	GOTO A002	1
35.	GOTO A003	1
36.	GOTO A004	1
37.	GOTO A005	1
38.	GOTO A006	1
39.	GOTO A007	1
40.	GOTO A008	1
41.	GOTO A009	1
42.	GOTO A010	1
43.	GOTO A011	1
44.	GOTO A012	1
45.	GOTO A013	1

$n_1 = 45, \quad N_1 = 417$

Table 3A
Sample Result
Comparison of various Models on Operator Distribution
ALGOL Program #3

SL. NO.	OBSERVED	ZIPF	ZWEBEN	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
1.	52	20.000	39.000	39.000	39.000	79.373	75.325	58.961	95.296
2.	47	10.000	25.552	19.500	19.000	50.358	37.663	38.630	41.554
3.	38	6.667	18.740	13.000	12.333	33.498	25.108	28.382	26.974
4.	32	5.000	14.625	9.750	9.000	23.248	18.831	22.111	19.684
5.	20	4.000	11.870	7.800	7.000	16.759	15.065	17.945	15.810
6.	15	3.333	9.896	6.500	5.667	12.500	12.554	14.960	12.393
7.	12	2.857	8.412	5.571	4.714	9.612	10.761	12.717	10.311
8.	11	2.500	7.256	4.875	4.000	7.596	9.416	10.970	8.748
9.	10	2.222	6.330	4.333	3.444	6.151	8.369	9.570	7.583
10.	8	2.000	5.571	3.900	3.000	5.092	7.533	8.423	6.561
11.	7	1.818	4.939	3.545	2.636	4.299	6.848	7.467	5.766
12.	7	1.667	4.403	3.250	2.333	3.694	6.277	6.657	5.103
13.	4	1.538	3.944	3.000	2.077	3.224	5.794	5.962	4.542
14.	2	1.429	3.545	2.786	1.857	2.855	5.380	5.360	4.062
15.	1	1.333	3.197	2.600	1.667	2.560	5.022	4.883	3.645
16.	1	1.250	2.889	2.438	1.500	2.321	4.708	4.368	3.281
17.	1	1.176	2.615	2.294	1.353	2.127	4.431	3.954	2.959
18.	1	1.111	2.370	2.167	1.222	1.966	4.185	3.583	2.673
19.	1	1.053	2.150	2.053	1.105	1.833	3.964	3.250	2.417
20.	1	1.000	1.950	1.950	1.000	1.721	3.766	2.948	2.187
TOTAL COUNT:	271	71.955	179.253	140.312	123.910	270.784	271.000	271.000	271.000
DEVN. OF TOTAL COUNT:		73.448%	33.855%	48.224%	54.277%	0.080%	-0.000%	0.000%	0.000%
INTERCEPT:		0.219	1.315	0.426	-0.563	-2.412	0.824	1.989	1.230
SLOPE:		0.249	0.564	0.486	0.499	1.177	0.939	0.853	1.091
CORR. COEFF.:		0.895	0.963	0.895	0.895	0.946	0.895	0.963	0.895

Table 3B
Sample Result
Comparison of Various Models on Operator Distribution
PL/I Program #7

SL. NO.	OBSERVED	Z_{IPF}	ZWEBEN	Z_{IPF-A}	Z_{IPF-B}	ZWB-HLSD	MOD-Z_{IPF}	MOD-ZWB_N	MOD-ZPFB
1.	111	45.000	73.000	73.000	73.000	115.816	94.882	69.312	108.193
2.	53	22.500	48.301	36.500	36.182	73.335	47.441	45.861	31.147
3.	49	15.000	35.881	24.333	23.909	48.522	31.627	34.069	33.798
4.	27	11.250	28.407	18.250	17.773	33.405	23.720	26.972	25.124
5.	25	9.000	23.415	14.600	14.091	23.839	18.976	22.232	19.919
6.	21	7.500	19.845	12.167	11.636	17.573	15.814	18.842	16.449
7.	15	6.429	17.164	10.429	9.883	13.339	13.555	16.297	13.921
8.	14	5.625	15.078	9.125	8.568	10.398	11.860	14.316	12.112
9.	13	5.000	13.408	8.111	7.545	8.301	10.542	12.731	10.666
10.	11	4.500	12.041	7.300	6.727	6.773	9.488	11.433	9.510
11.	10	4.091	10.902	6.636	6.058	5.635	8.626	10.351	8.563
12.	8	3.750	9.937	6.083	5.500	4.772	7.907	9.485	7.775
13.	4	3.462	9.110	5.615	5.028	4.106	7.299	8.650	7.108
14.	4	3.214	8.393	5.214	4.623	3.584	6.777	7.969	6.536
15.	4	3.000	7.766	4.867	4.273	3.170	6.325	7.374	6.040
16.	4	2.813	7.212	4.563	3.966	2.837	5.930	6.848	5.606
17.	4	2.647	6.720	4.294	3.695	2.567	5.581	6.381	5.224
18.	3	2.500	6.280	4.056	3.455	2.344	5.271	5.962	4.883
19.	3	2.368	5.883	3.842	3.239	2.160	4.994	5.586	4.579
20.	3	2.250	5.524	3.650	3.045	2.006	4.744	5.245	4.305
21.	2	2.143	5.198	3.476	2.870	1.876	4.518	4.936	4.057
22.	2	2.045	4.900	3.318	2.711	1.766	4.313	4.653	3.832
23.	2	1.957	4.627	3.174	2.565	1.672	4.125	4.393	3.626
24.	2	1.875	4.376	3.042	2.432	1.591	3.953	4.155	3.438
25.	2	1.800	4.144	2.920	2.309	1.522	3.795	3.935	3.264
26.	2	1.731	3.929	2.808	2.196	1.461	3.649	3.731	3.104
27.	1	1.667	3.730	2.704	2.091	1.409	3.514	3.541	2.956
28.	1	1.607	3.544	2.607	1.994	1.363	3.389	3.365	2.818
29.	1	1.552	3.371	2.517	1.903	1.323	3.272	3.201	2.690
30.	1	1.500	3.209	2.433	1.818	1.288	3.163	3.047	2.570
31.	1	1.452	3.057	2.355	1.739	1.257	3.061	2.902	2.458
32.	1	1.406	2.914	2.281	1.665	1.230	2.965	2.767	2.353
33.	1	1.364	2.780	2.212	1.595	1.206	2.875	2.639	2.255
34.	1	1.324	2.653	2.147	1.529	1.184	2.791	2.519	2.162
35.	1	1.286	2.533	2.086	1.468	1.165	2.711	2.405	2.075
36.	1	1.250	2.420	2.028	1.409	1.148	2.636	2.298	1.992
37.	1	1.216	2.313	1.973	1.354	1.133	2.564	2.196	1.914
38.	1	1.184	2.211	1.921	1.301	1.119	2.497	2.099	1.840
39.	1	1.154	2.114	1.872	1.252	1.107	2.433	2.007	1.769
40.	1	1.125	2.022	1.825	1.205	1.097	2.372	1.920	1.703
41.	1	1.098	1.935	1.780	1.160	1.087	2.314	1.837	1.639
42.	1	1.071	1.851	1.738	1.117	1.078	2.259	1.758	1.579
43.	1	1.047	1.771	1.698	1.076	1.070	2.207	1.682	1.521
44.	1	1.023	1.695	1.659	1.037	1.063	2.156	1.609	1.466
45.	1	1.000	1.622	1.622	1.000	1.057	2.108	1.540	1.414
TOTAL COUNT:	417	197.773	439.188	320.831	294.992	416.755	417.000	417.000	417.000
DEVN. OF TOTAL COUNT:		52.573%	-5.321%	23.062%	29.259%	0.059%	-0.000%	0.000%	0.000%
INTERCEPT:		0.873	3.354	1.416	0.792	-0.785	1.841	3.185	1.120
SLOPE:		0.380	0.691	0.617	0.622	1.084	0.801	0.656	0.879
CORR. COEFF.:		0.993	0.984	0.993	0.993	0.988	0.993	0.984	0.993

Table 4

Mean and Std. Dev. of the Intercept, Slope and Corr. Coeff. corresponding to ALGOL Programs

	INTERCEPT		SLOPE		CORR. COEFF.	
	MEAN	SD	MEAN	SD	MEAN	SD
ZIPF	-0.011	0.431	0.846	0.343	0.943	0.028
ZWEBEN	0.645	0.666	1.007	0.257	0.969	0.014
ZIPF-A	0.046	0.511	0.941	0.261	0.943	0.028
ZIPF-B	-0.191	0.451	0.948	0.258	0.943	0.028
ZWB-HLSD	0.201	0.778	0.908	0.121	0.969	0.015
MOD-ZIPF	0.105	0.546	1.008	0.135	0.943	0.028
MOD-ZWBN	0.642	0.672	0.884	0.112	0.969	0.014
MOD-ZPFB	-0.239	0.514	1.051	0.130	0.943	0.028

Table 4_B

Mean and Std. Dev. of the Intercept, Slope and Corr. Coeff. Corresponding to P_T/I Programs

	INTERCEPT		SLOPE		CORR. COEFF.	
	MEAN	SD	MEAN	SD	MEAN	SD
ZIPF	1.037	0.481	0.178	0.113	0.986	0.011
ZWEBEN	12.427	8.813	0.716	0.117	0.977	0.012
ZIPF-A	5.919	4.435	0.644	0.109	0.986	0.011
ZIPF-B	1.646	2.231	0.656	0.113	0.986	0.011
ZWB-HLSD	-3.527	4.357	1.093	0.099	0.980	0.013
MOD-ZIPF	6.879	5.372	0.767	0.087	0.986	0.011
MOD-ZWBN	10.548	7.626	0.630	0.083	0.977	0.012
MOD-ZPFB	2.194	3.109	0.923	0.105	0.986	0.011

5.2 Discussion of the Results

As stated earlier, the bases of judgement about how closely a model approximates the actual distribution are the regression coefficients and the correlation coefficient between the actual and the predicted distributions. The goodness of fit of a model is seen from how close the slope in the regression analysis and the correlation coefficient are each to unity and how close the intercept is to zero (see Appendix B for detailed description). With these in mind, the summary of the results of the experiments on ALGOL and PL/I programs are as under.

(i) In all ALGOL and PL/I programs, the correlation coefficients for Zipf, Zipf-A, Zipf-B, Modified Zipf and Modified Zipf-B models are the same. This ascertains the similar nature of distribution in these models. Similar is seen in the case of Zweben and Modified Zweben models. Zweben-Halstead model, of course, has got altogether different nature of distribution.

(ii) Unlike old models, viz., Zipf, Zweben, Zipf-A and Zipf-B, the total operator count predicted by each of the modified models, i.e., Modified Zipf, Modified Zweben and Modified Zipf-B is same as the actual count N_1 in all ALGOL and PL/I programs. This, of course, is due to the inherent constraint put on the modified

models. In the case of the Zweben-Halstead model, the predicted total count is slightly deviated (of the order of 0.1% or less) from the actual count N_1. This, however, is due to the error involved in the iterative solution of equation (4.2.12).

(iii) Among old models, viz., Zipf, Zweben, Zipf-A and Zipf-B, the Zweben model is seen to give the best approximation in both ALGOL and PL/I programs. This is apparent from the slope of the least square line in most programs, as also from the average slope. However, surprisingly enough, when these models are each modified to take the information of N_1 also, the Modified Zipf, Modified Zipf-B as also Zweben-Halstead model are each seen to be better than the Modified Zweben model in both ALGOL and PL/I programs. In fact, while the Modified Zipf and Modified Zipf-B models are better than the Zipf and Zipf-B models, respectively, the Modified Zweben model performs poorly in comparison to the old Zweben model.

(iv) In the case of PL/I programs, Zweben-Halstead model is seen to be the best among all models. This is apparent from the average slope which has got a value closest to unity and the smallest standard deviation. The correlation coefficient, of course, is slightly less.

However, in the case of ALGOL programs, the behaviour of Zweben-Halstead model is not that good. Modified Zipf and Modified

Zipf-B models are seen to perform better. One reason for this could be the small size of the ALGOL programs compared to that of the PL/I programs. The difference in the programming languages such as ALGOL, PL/I and FORTRAN should not make much difference if the relationships are indeed of a general nature.

(v) In the case of ALGOL programs, Modified Zipf and Modified Zipf-B models are seen to perform better than all other models. The distinction between these two models, however, is very subtle. Examination of the behaviour of the two models in individual ALGOL programs suggests that Modified Zipf model is slightly better than the Modified Zipf-B model in the sense that it provides a better fit to the actual distribution of operators for more number of programs. Also, the average slope of the Modified Zipf model as compared to that of the Modified Zipf-B model is slightly closer to unity, although with a larger dispersion. The average intercept also is smaller. The correlation coefficient, however, is the same. These figures are all apparent from Table 4A.

(vi) In the case of PL/I programs, Modified Zipf-B model is seen to compare fairly well with the Zweben-Halstead model which was earlier seen to be the best among all existing models. This, of course, is the most important finding of our research. The distinction between these two models is again very narrow, just the same way as that between Modified Zipf and Modified Zipf-B

models for ALGOL programs. When individual PL/I programs are examined, Zweben-Halstead model is seen to perform better than the Modified Zipf-B model in more number of cases. However, the mean of the slope of the Modified Zipf-B model is closer to unity than that of the Zweben-Halstead model, although with a slightly larger standard deviation. Average intercept and average correlation coefficient also suggest in favour of the Modified Zipf-B model. These figures are apparent from Table 4B.

5.3 Conclusions and Scope for Further Study

Keeping in mind the findings of the experiments, the following conclusions have been drawn :

(a) Halstead's hypotheses regarding functional relationships among the operators and operands of an algorithm seem to be justified.

(b) Two models among the modified ones, viz., Modified Zipf and Modified Zipf-B show clear improvement over the old ones. However, there does not seem to be a single model that does better than all others for both ALGOL and PL/I programs.

(c) Modified Zipf-B model appears among the best two models in both ALGOL and PL/I programs. In this sense, this model can be taken as the best one available so far.

One reason why ALGOL and PL/I programs do not quite conform to each other may be the relatively small size of the ALGOL programs. What is required further is testing of the models on programs of fairly large size. Also, programs in other languages, e.g., FORTRAN, PASCAL should be included.

Finally, it should be noted that the more we know about how information is organised, whether in natural language texts or in algorithmic texts, the better use we can make of that information. It is this point which is most important in studies like this.

References :

[1] Halstead, M.H., "Elements of Software Science", Elsevier Computer Science Library, New York, 1977.

[2] Zipf, G.K., "Human Behaviour and the Principle of Least Effort", Addison-Wesley, Reading, Mass., 1949.

[3] Zipf, G.K., "The Psycho-biology of Lenguage : An Introduction to Dynamic Philology", MIT Press, 1965.

[4] Oldehoeft and Bass, "Dynamic Software Science with Applications", IEEE Trans. Softw. Eng., Vol.SE-5, No.5, pp.497-504, Sept. 1979.

[5] Elshoff, J.L., "Measuring PL/I Programs using Halstead's criteria", ACM SIGPLAN Notices, Vol.11, No.5, pp. 38-46, May 1976.

[6] Bulut, N., Halstead, M.H. and Bayer, R., "Experimental Validation of a Structural Property of FORTRAN Algorithms", CSD TR 115, Purdue University, Dept. of Computer Sciences, W.Lafayette, Ind., April 1974.

[7] Zweben, S.H., "Physical Structure of Algorithms", IEEE Trans. Softw. Eng., Vol. SE-3, No.3, pp.250-258, May 1977.

[8] Elshoff, J.L., "A Study of the Structural Composition of PL/I Programs", ACM SIGPLAN Notices, Vol.13, No.6, pp. 29-37, June 1978.

[9] Halstead, M.H. and Bayer, R., "Algorithm Dynamics", Proc. ACM Annual Conf., Atlanta, pp. 126-135, Aug. 1973.

[10] Bayer, R., "A Theoretical Study of Halstead's Software Phenomenon", CSD TR 69, Purdue University, Dept. of Computer Sciences, W. Lafayette, Ind., May 1972.

[11] Zweben, S.H. and Halstead, M.H., "The Frequency Distribution of Operators in PL/I Programs", IEEE Trans. Softw. Eng., Vol. SE-5, No.2, pp. 91-95, March 1979.

[12] Gordon, R.D. and Halstead, M.H., "An Experiment Comparing FORTRAN Programming Time with the Software Physics Hypothesis", Proc. AFIPS, 1976, National Computer Conf., Vol. 45, pp. 935-937, AFIPS Press, Montvale, N.J.

[13] Halstead, M.H., "Using the Methodology of Natural Science to understand Software", CSD TR 190, Purdue University, Dept. of Computer Sciences, W. Lafayette, Ind.

[14] Fitzsimmons, A. and Love, T., "A Review and Evaluation of Software Science", ACM Computing Surveys, Vol. 10, No.1, pp. 3-18, March 1978.

[15] Baker, A.L. and Zweben, S.H., "The Use of Software Science in Evaluating Modularity Concepts", IEEE Trans. Softw. Eng., Vol. SE-5, No.2, pp. 110-120, March 1979.

[16] Johnson, D.B. and Lister, A.M., "An Experiment in Software Science", Proc. Symposium on Language Design and Programming Methodology, Sydney, Sept. 1979, Lecture Notes in Computer Science, pp. 195-215, No. 79.

[17] Curtis, B. et.al., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", IEEE Trans. Softw. Eng., Vol.SE-5, No.2, pp.96-104, March 1979.

[18] McCabe, T.J., "A Complexity Measure", IEEE Trans. Softw. Eng., Vol. SE-2, No.4, pp. 308-320, Dec. 1976.

[19] Ottenstein, K.J., "A Program to count Operators and Operands for ANSI-FORTRAN Modules", CSD TR 196, Purdue University, Dept. of Computer Sciences, W. Lafyette, Ind., June 1976.

[20] Bulut N. and Halstead, M.H., "Impurities Found in Algorithmic Implementations", ACM SIGPLAN Notices, Vol. 9, No. 3, pp. 9-12, March 1974.

[21] Moranda, P.B., "Surveyors' Forum", ACM Computing Surveys, Vol. 10, No.4, pp. 503-504, Dec. 1978.

[22] Baker, T.P., "Surveyors' Forum", ACM Computing Surveys, Vol. 10, No.4, p. 504, Dec. 1978.

[23] Fenichel, R., "Surveyors' Forum", ACM Computing Surveys, Vol. 11, No.3, p. 277, Sept. 1979.

[24] Zweben, S.H., "Surveyors' Forum", ACM Computing Surveys, Vol. 11, No.3, pp. 277-278, Sept. 1979.

[25] Halstead, M.H., Uber, G.T. and Gielow, K.R., "An Algorithmic Search Procedure for Program Generation", 1967 Spring Joint Computer Conf., AFIPS Conf. Proc., Atlantic City, NJ. AFIPS Press, Vol. 30, pp. 657-662, 1967.

[26] Love, L.T. and Bowman, A.B., "An Independent Test of the Theory of Software Physics", ACM SIGPLAN Notices, Vol.11, No.11, pp. 42-49, Nov. 1976.

[27] Curtis, B., "Measurement and Experimentation in Software Eng.", Proc. IEEE, Vol. 68, No.9, pp. 1149-1150, Sept. 1980.

[28] Kavipuraup, "Quantification of Architectures using Software Science", Computer Architecture News, Vol.7, No. 10, Oct. 15, 1979.

[29] Hansen, W.J., "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", ACM SIGPLAN Notices, Vol. 13, pp. 29-33, March 1978.

Appendix A

All our experiments are based on data derived from programs written in ALGOL and PL/I. The ALGOL data have been obtained by studying the first 14 algorithms published in the Communications of the Association for Computing Machinery (starting CACM, Vol. 3, No. 2, Feb. 1960). The PL/I data, on the other hand, have been obtained from Dr. J.L. Elshoff who analysed 34 commercial PL/I programs in use at the General Motor Laboratories, Warren, Michigan, USA.

Regarding counting of the programs taken by us for the experiment purposes, the counting of the CACM ALGOL programs was manually done by us while that of the PL/I programs was done by Elshoff [8] and we simply adopted it.

Appendix B

Regression Analysis and Correlation Studies

The regression analysis of a set of data points, say, $(x_1,y_1), (x_2,y_2),\ldots,(x_n,y_n)$ consists in finding a straight line having the least mean squared error relative to the given points $(x_i,y_i)|i = 1,n$. It attempts to discover a relationship between the two variables. If the straight line approximating the above set of data points has the equation

$$y = a + bx , \qquad (4.1.1)$$

then the constraint of least mean squared error yields the following equations :

$$\begin{aligned} \Sigma y_i &= an + b\Sigma x_i , \\ \Sigma x_i y_i &= a\Sigma x_i + b\Sigma x_i^2 , \\ i &= 1,2,\ldots,n. \end{aligned} \qquad (4.1.2)$$

Simultaneous solution of these equations gives the values of the intercept a and slope b of the least square line which are as follows :

$$a = \frac{(\Sigma y_i)(\Sigma x_i^2) - (\Sigma x_i)(\Sigma x_i y_i)}{n\Sigma x_i^2 - (\Sigma x_i)^2} \qquad (4.1.3)$$

$$b = \frac{n\Sigma x_i y_i - (\Sigma x_i)(\Sigma y_i)}{n\Sigma x_i^2 - (\Sigma x_i)^2}$$

$$i = 1,2,\ldots,n.$$

Regarding correlation coefficient, it is a measure of the dispersion of the set of data points $(x_i,y_i)|i = 1,2,\ldots,n$ about a straight line. It is used as a measure of the strength of the relationship between random variables x and y. It is given from

$$r = \frac{n\Sigma x_i y_i - (\Sigma x_i)(\Sigma y_i)}{(n\Sigma x_i^2 - (\Sigma x_i)^2)(n\Sigma y_i^2 - (\Sigma y_i)^2)}, \quad (4.1.4)$$

$$i = 1,2,\ldots,n.$$

Consequent to the definitions of the regression coefficients and the correlation coefficient above, we shall state their respective applications towards deciding the goodness of fit of the various models. Corresponding to every program taken for experiment, we obtain the distributions predicted by each model. Each of these distributions is then compared with the actual distribution yielding regression coefficients and correlation coefficient. However, it is worth mentioning here the distinction between the two statistics. Some scientists [21,22,23] have commented on the way the appropriateness of software science metrics is gauged on the basis of these statistics. The correlation coefficient, as discussed above, is a measure of the degree of linear dependence between two random variables. For example, the correlation coefficient between, say, (x_i), $i = 1,2,\ldots,n$ and (y_i), $i = 1,2,\ldots,n$ will be the same as that between (αx_i), $i=1,2,\ldots,n$ and (βy_i), $i = 1,2,\ldots,n$ where α and β are constants. This suggests that the correlation coefficient can only give an idea

about the nature of the observed distribution compared to that of the actual distribution. This is seen from how close it is to unity. Regression analysis, on the other hand, provides an idea about how best the predicted distribution approximates the actual one. This is seen from how close the intercept a is to zero and the slope b to unity. Regression analysis [21,22,23,24], therefore, is a better barometer to judge the goodness of a model. We have, however, determined both statistics.

APPENDIX C

SUMMARY OF THE RESULTS OF ALGOL PROGRAMS

	OBSERVED	ZIPF	ZWEBEN	ZIPF-A	ZIPF-B	ZWB-RLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 1									
TOTAL COUNT:	53	33.219	65.382	54.358	48.772	52.959	53.000	53.000	53.000
DEVN. OF TOTAL COUNT:		37.323%	-23.362%	-2.562%	7.978%	0.078%	0.000%	0.000%	-0.000%
INTERCEPT:		-0.126	0.334	-0.206	-0.914	0.160	-0.201	0.271	-0.993
SLOPE:		0.653	1.164	1.068	1.110	0.966	1.042	0.944	1.206
CORR. COEFF.:		0.921	0.971	0.921	0.921	0.969	0.921	0.971	0.921
PRG. # 2									
TOTAL COUNT:	45	41.342	50.609	41.342	41.342	45.041	45.000	45.000	45.000
DEVN. OF TOTAL COUNT:		8.129%	-12.465%	8.129%	8.129%	-0.091%	0.000%	0.000%	0.000%
INTERCEPT:		0.157	0.574	0.157	0.157	0.565	0.171	0.511	0.171
SLOPE:		0.873	0.959	0.873	0.873	0.838	0.951	0.852	0.951
CORR. COEFF.:		0.915	0.961	0.915	0.915	0.960	0.915	0.961	0.915
PRG. # 3									
TOTAL COUNT:	271	71.955	179.253	140.312	123.910	270.784	271.000	271.000	271.000
DEVN. OF TOTAL COUNT:		73.448%	33.855%	48.224%	54.277%	0.080%	-0.000%	0.000%	0.000%
INTERCEPT:		0.219	1.315	0.426	-0.563	-2.412	0.824	1.989	-1.230
SLOPE:		0.249	0.564	0.486	0.499	1.177	0.939	0.853	1.091
CORR. COEFF.:		0.895	0.963	0.895	0.895	0.946	0.895	0.963	0.895
PRG. # 4									
TOTAL COUNT:	61	71.955	91.925	71.955	71.955	61.053	61.000	61.000	61.000
DEVN. OF TOTAL COUNT:		-17.959%	-50.696%	-17.959%	-17.959%	-0.087%	-0.000%	0.000%	-0.000%
INTERCEPT:		0.450	1.104	0.450	0.450	0.630	0.381	0.732	0.381
SLOPE:		1.032	1.145	1.032	1.032	0.794	0.875	0.760	0.875
CORR. COEFF.:		0.925	0.952	0.925	0.925	0.962	0.925	0.952	0.925
PRG. # 5									
TOTAL COUNT:	64	58.472	73.543	58.472	58.472	64.051	64.000	64.000	64.000
DEVN. OF TOTAL COUNT:		8.637%	-14.910%	8.637%	8.637%	-0.079%	0.000%	0.000%	0.000%
INTERCEPT:		0.131	0.623	0.131	0.131	0.311	0.143	0.542	0.143
SLOPE:		0.879	0.984	0.879	0.879	0.918	0.962	0.856	0.962
CORR. COEFF.:		0.924	0.970	0.924	0.924	0.969	0.924	0.970	0.924
PRG. # 6									
TOTAL COUNT:	56	45.522	56.153	45.522	45.522	56.055	56.000	56.000	56.000
DEVN. OF TOTAL COUNT:		18.711%	-0.273%	18.711%	18.711%	-0.099%	-0.000%	0.000%	-0.000%
INTERCEPT:		0.081	0.587	0.081	0.081	0.441	0.099	0.586	0.099
SLOPE:		0.793	0.856	0.793	0.793	0.891	0.975	0.854	0.975
CORR. COEFF.:		0.968	0.994	0.968	0.968	0.994	0.968	0.994	0.968
PRG. # 7									
TOTAL COUNT:	30	25.461	29.963	25.461	25.461	30.030	30.000	30.000	30.000
DEVN. OF TOTAL COUNT:		15.131%	0.125%	15.131%	15.131%	-0.100%	0.000%	-0.000%	0.000%
INTERCEPT:		0.277	0.631	0.277	0.277	0.920	0.326	0.632	0.326
SLOPE:		0.766	0.809	0.766	0.766	0.725	0.902	0.810	0.902
CORR. COEFF.:		0.920	0.954	0.920	0.920	0.954	0.920	0.954	0.920

APPENDIX C (contd.)

	OBSERVED	ZIPF	ZWEBFN	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 8									
TOTAL COUNT:	74	49.773	65.950	53.092	52.257	73.933	74.000	74.000	74.000
DEVN. OF TOTAL COUNT:		32.739%	10.878%	28.255%	29.382%	0.091%	-0.000%	0.000%	-0.000%
INTERCEPT:		-0.084	0.424	-0.090	-0.162	-0.028	-0.126	0.476	-0.229
SLOPE:		0.690	0.805	0.736	0.739	1.005	1.025	0.903	1.046
CORR. COEFF.:		0.955	0.989	0.955	0.955	0.986	0.955	0.989	0.955
PRG. # 9									
TOTAL COUNT:	142	71.955	165.464	129.519	115.706	141.875	142.000	142.000	142.000
DEVN. OF TOTAL COUNT:		49.328%	-16.524%	8.790%	18.517%	0.088%	-0.000%	-0.000%	0.000%
INTERCEPT:		0.565	2.172	1.016	0.198	0.349	1.114	1.864	0.243
SLOPE:		0.427	0.859	0.769	0.787	0.950	0.843	0.737	0.966
CORR. COEFF.:		0.938	0.973	0.938	0.938	0.973	0.938	0.973	0.938
PRG. # 10									
TOTAL COUNT:	26	29.290	34.869	29.290	29.290	26.024	26.000	26.000	26.000
DEVN. OF TOTAL COUNT:		-12.653%	-34.112%	-12.653%	-12.653%	-0.092%	0.000%	-0.000%	0.000%
INTERCEPT:		-0.002	0.596	-0.002	-0.002	0.778	-0.002	0.444	-0.002
SLOPE:		1.127	1.112	1.127	1.127	0.702	1.001	0.829	1.001
CORR. COEFF.:		0.981	0.942	0.981	0.981	0.952	0.981	0.942	0.981
PRG. # 11									
TOTAL COUNT:	29	29.290	34.869	29.290	29.290	29.026	29.000	29.000	29.000
DEVN. OF TOTAL COUNT:		-0.999%	-20.239%	-0.999%	-0.999%	-0.090%	-0.000%	0.000%	-0.000%
INTERCEPT:		-0.353	0.182	-0.353	-0.353	0.482	-0.349	0.151	-0.349
SLOPE:		1.132	1.140	1.132	1.132	0.835	1.120	0.948	1.120
CORR. COEFF.:		0.989	0.971	0.989	0.989	0.977	0.989	0.971	0.989
PRG. # 12									
TOTAL COUNT:	34	37.239	45.206	37.239	37.239	34.033	34.000	34.000	34.000
DEVN. OF TOTAL COUNT:		-9.525%	-32.960%	-9.525%	-9.525%	-0.097%	-0.000%	0.000%	-0.000%
INTERCEPT:		-1.038	-0.615	-1.038	-1.038	-0.094	-0.948	-0.462	-0.948
SLOPE:		1.462	1.547	1.462	1.462	1.034	1.334	1.163	1.334
CORR. COEFF.:		0.943	0.959	0.943	0.943	0.946	0.943	0.959	0.943
PRG. # 13									
TOTAL COUNT:	34	37.239	45.206	37.239	37.239	34.033	34.000	34.000	34.000
DEVN. OF TOTAL COUNT:		-9.525%	-32.960%	-9.525%	-9.525%	-0.097%	-0.000%	0.000%	-0.000%
INTERCEPT:		-0.762	-0.240	-0.762	-0.762	0.115	-0.696	-0.180	-0.696
SLOPE:		1.364	1.414	1.364	1.364	0.960	1.246	1.064	1.246
CORR. COEFF.:		0.989	0.985	0.989	0.989	0.987	0.989	0.985	0.989
PRG. # 14									
TOTAL COUNT:	92	41.342	85.646	69.963	62.598	91.934	92.000	92.000	92.000
DEVN. OF TOTAL COUNT:		55.063%	6.906%	23.953%	31.959%	0.071%	-0.000%	-0.000%	0.000%
INTERCEPT:		0.329	1.336	0.556	-0.175	0.597	0.732	1.435	-0.257
SLOPE:		0.403	0.742	0.682	0.705	0.915	0.897	0.797	1.036
CORR. COEFF.:		0.945	0.983	0.945	0.945	0.982	0.945	0.983	0.945

APPENDIX D

SUMMARY OF THE RESULTS OF PL/I PROGRAMS

	OBSERVED	ZIPF	ZWEBEN	ZIPF-A	ZIPF-B	ZWB-BLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 1									
TOTAL COUNT:	930	134.930	879.102	658.297	542.652	929.340	930.000	930.000	930.000
DEVN. OF TOTAL COUNT:		85.491%	5.466%	29.215%	41.650%	0.071%	-0.000%	-0.000%	-0.000%
INTERCEPT:		1.152	10.850	5.620	1.760	-0.441	7.940	11.478	3.016
SLOPE:		0.104	0.500	0.508	0.521	1.015	0.718	0.593	0.893
CORR. COEFF.:		0.989	0.980	0.989	0.989	0.985	0.989	0.980	0.989
PRG. # 2									
TOTAL COUNT:	1678	203.188	1925.854	1404.507	1153.159	1676.477	1678.000	1678.000	1678.000
DEVN. OF TOTAL COUNT:		87.892%	-14.771%	16.299%	31.278%	0.091%	-0.000%	0.000%	-0.000%
INTERCEPT:		0.987	15.100	6.821	0.906	-6.955	8.149	13.157	1.319
SLOPE:		0.094	0.734	0.650	0.662	1.190	0.777	0.639	0.964
CORR. COEFF.:		0.990	0.907	0.990	0.990	0.980	0.990	0.987	0.990
PRG. # 3									
TOTAL COUNT:	892	224.900	1328.507	962.830	810.542	891.409	892.000	892.000	892.000
DEVN. OF TOTAL COUNT:		74.760%	-48.936%	-7.941%	9.132%	0.066%	0.000%	0.000%	-0.000%
INTERCEPT:		1.452	12.293	6.216	2.966	0.137	5.759	8.254	3.264
SLOPE:		0.171	0.800	0.731	0.742	0.992	0.677	0.537	0.817
CORR. COEFF.:		0.992	0.955	0.992	0.992	0.970	0.992	0.955	0.992
PRG. # 4									
TOTAL COUNT:	782	197.773	1113.010	813.065	683.868	781.526	782.000	782.000	782.000
DEVN. OF TOTAL COUNT:		74.769%	-42.329%	-3.973%	12.549%	0.061%	-0.000%	-0.000%	0.000%
INTERCEPT:		1.103	9.988	4.782	1.683	-1.075	4.599	7.017	1.924
SLOPE:		0.166	0.849	0.765	0.778	1.061	0.735	0.596	0.889
CORR. COEFF.:		0.993	0.974	0.993	0.993	0.981	0.993	0.974	0.993
PRG. # 5									
TOTAL COUNT:	3579	494.089	4905.828	3402.010	2861.673	3576.512	3579.000	3579.000	3579.000
DEVN. OF TOTAL COUNT:		86.195%	-37.073%	4.945%	20.043%	0.070%	0.000%	0.000%	0.000%
INTERCEPT:		1.677	23.402	11.545	5.702	-7.235	12.146	17.073	7.131
SLOPE:		0.093	0.743	0.641	0.647	1.193	0.674	0.542	0.809
CORR. COEFF.:		0.990	0.981	0.990	0.990	0.982	0.990	0.981	0.990
PRG. # 6									
TOTAL COUNT:	3036	487.953	3989.694	2768.491	2344.028	3033.012	3036.000	3036.000	3036.000
DEVN. OF TOTAL COUNT:		83.928%	-31.413%	8.811%	22.792%	0.098%	0.000%	0.000%	0.000%
INTERCEPT:		1.672	19.333	9.484	4.844	-5.737	10.400	14.712	6.274
SLOPE:		0.108	0.709	0.615	0.621	1.179	0.675	0.540	0.804
CORR. COEFF.:		0.992	0.977	0.992	0.992	0.981	0.992	0.977	0.992
PRG. # 7									
TOTAL COUNT:	417	197.773	439.188	320.831	294.992	416.755	417.000	417.000	417.000
DEVN. OF TOTAL COUNT:		52.573%	-5.321%	23.062%	29.259%	0.059%	-0.000%	0.000%	0.000%
INTERCEPT:		0.873	3.354	1.416	0.792	-0.785	1.841	3.185	1.120
SLOPE:		0.380	0.691	0.617	0.622	1.084	0.801	0.656	0.879
CORR. COEFF.:		0.993	0.984	0.993	0.993	0.988	0.993	0.984	0.993

APPENDIX F (cont.)

OBSERVED	ZIPF	ZWEIPF	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 8								
TOTAL COUNT: 1579	297.881	1983.294	1413.752	1191.943	1578.236	1579.000	1579.000	1579.000
DEVN. OF TOTAL COUNT:	81.135%	-25.604%	10.465%	24.513%	0.048%	-0.000%	-0.000%	-0.000%
INTERCEPT:	1.475	14.107	7.002	3.284	-2.261	7.820	11.231	4.351
SLOPE:	0.136	0.693	0.616	0.624	1.090	0.688	0.552	0.826
CORR. COEFF.:	0.990	0.969	0.990	0.990	0.973	0.990	0.969	0.990
PRG. # 9								
TOTAL COUNT: 1529	187.050	2035.359	1492.050	1215.978	1527.785	1529.000	1529.000	1529.000
DEVN. OF TOTAL COUNT:	87.767%	-33.117%	2.417%	20.472%	0.079%	0.000%	-0.000%	-0.000%
INTERCEPT:	0.632	13.711	5.039	-1.999	-10.107	5.164	10.300	-2.514
SLOPE:	0.105	0.946	0.834	0.851	1.283	0.855	0.710	1.071
CORR. COEFF.:	0.981	0.986	0.981	0.981	0.967	0.981	0.986	0.981
PRG. # 10								
TOTAL COUNT: 2685	247.073	3180.781	2292.290	1875.446	2682.606	2685.000	2685.000	2685.000
DEVN. OF TOTAL COUNT:	90.798%	-18.465%	14.626%	30.151%	0.089%	0.000%	0.000%	-0.000%
INTERCEPT:	1.321	24.874	12.260	4.033	-7.537	14.361	20.997	5.774
SLOPE:	0.065	0.684	0.607	0.617	1.151	0.711	0.578	0.884
CORR. COEFF.:	0.995	0.982	0.995	0.995	0.991	0.995	0.982	0.995
PRG. # 11								
TOTAL COUNT: 2616	230.459	3259.451	2358.820	1920.968	2614.481	2616.000	2616.000	2616.000
DEVN. OF TOTAL COUNT:	91.190%	-24.597%	9.831%	26.569%	0.058%	-0.000%	0.000%	-0.000%
INTERCEPT:	1.259	26.325	12.888	3.700	-8.077	14.293	21.128	5.039
SLOPE:	0.064	0.733	0.650	0.662	1.157	0.721	0.588	0.902
CORR. COEFF.:	0.993	0.983	0.993	0.993	0.990	0.993	0.983	0.993
PRG. # 12								
TOTAL COUNT: 3065	155.419	3534.334	2621.790	2086.993	3063.823	3065.000	3065.000	3065.000
DEVN. OF TOTAL COUNT:	94.928%	-15.313%	14.460%	31.909%	0.038%	0.000%	0.000%	0.000%
INTERCEPT:	0.838	31.618	14.136	-1.800	-19.040	16.526	27.419	-2.644
SLOPE:	0.041	0.771	0.685	0.703	1.229	0.801	0.669	1.032
CORR. COEFF.:	0.989	0.995	0.989	0.989	0.989	0.989	0.995	0.989
PRG. # 13								
TOTAL COUNT: 4539	1073.050	6473.310	4327.003	3780.638	4534.906	4539.000	4539.000	4539.000
DEVN. OF TOTAL COUNT:	76.359%	-42.615%	4.671%	16.708%	0.090%	-0.000%	0.000%	0.000%
INTERCEPT:	2.303	18.698	9.285	6.274	-3.817	9.740	13.111	7.532
SLOPE:	0.143	0.614	0.575	0.577	1.155	0.603	0.466	0.693
CORR. COEFF.:	0.988	0.955	0.988	0.988	0.970	0.988	0.955	0.988
PRG. # 14								
TOTAL COUNT: 600	155.459	923.231	684.859	570.064	599.461	600.000	600.000	600.000
DEVN. OF TOTAL COUNT:	74.090%	-53.872%	-14.143%	4.989%	0.090%	0.000%	0.000%	-0.000%
INTERCEPT:	0.904	9.236	4.377	0.971	-1.012	3.835	6.002	1.022
SLOPE:	0.108	0.969	0.871	0.890	1.061	0.764	0.630	0.937
CORR. COEFF.:	0.993	0.986	0.993	0.993	0.992	0.993	0.986	0.993

APPENDIX D (contd.)

	OBSERVED	ZIPF	ZWEBLE	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 15									
TOTAL COUNT:	1794	457.431	1978.206	1377.377	1204.679	1792.897	1794.000	1794.000	1794.000
DEVN. OF TOTAL COUNT:		74.502%	-10.272%	23.223%	32.850%	0.061%	0.000%	-0.000%	-0.000%
INTERCEPT:		1.569	9.779	4.784	2.786	-3.055	6.231	8.868	4.150
SLOPE:		0.175	0.612	0.528	0.532	1.153	0.687	0.555	0.792
CORR. COEFF.:		0.966	0.973	0.980	0.980	0.969	0.980	0.973	0.980
PRG. # 16									
TOTAL COUNT:	1818	355.839	2673.039	1886.435	1589.332	1816.790	1818.000	1818.000	1818.000
DEVN. OF TOTAL COUNT:		80.427%	-47.032%	-3.764%	12.578%	0.067%	0.000%	-0.000%	0.000%
INTERCEPT:		1.795	18.445	9.515	5.261	-0.875	9.170	12.545	6.018
SLOPE:		0.124	0.730	0.656	0.663	1.034	0.632	0.496	0.758
CORR. COEFF.:		0.995	0.957	0.995	0.995	0.983	0.995	0.957	0.995
PRG. # 17									
TOTAL COUNT:	381	109.961	320.675	243.485	213.171	380.622	381.000	381.000	381.000
DEVN. OF TOTAL COUNT:		71.139%	15.833%	36.093%	44.050%	0.099%	-0.000%	0.000%	0.000%
INTERCEPT:		0.491	2.951	1.087	-0.151	-1.744	1.700	3.507	-0.269
SLOPE:		0.253	0.625	0.559	0.571	1.127	0.875	0.742	1.020
CORR. COEFF.:		0.978	0.992	0.978	0.978	0.981	0.978	0.992	0.978
PRG. # 18									
TOTAL COUNT:	612	129.872	437.969	328.738	284.573	611.514	612.000	612.000	612.000
DEVN. OF TOTAL COUNT:		78.779%	28.446%	46.285%	53.501%	0.079%	0.000%	0.000%	-0.000%
INTERCEPT:		0.277	2.983	0.700	-0.867	-4.729	1.303	4.169	-1.865
SLOPE:		0.198	0.560	0.501	0.510	1.246	0.932	0.782	1.097
CORR. COEFF.:		0.960	0.987	0.960	0.960	0.939	0.960	0.967	0.960
PRG. # 19									
TOTAL COUNT:	438	100.215	332.874	254.392	218.959	437.745	438.000	438.000	438.000
DEVN. OF TOTAL COUNT:		77.120%	24.001%	41.920%	50.009%	0.058%	-0.000%	0.000%	-0.000%
INTERCEPT:		0.121	2.217	0.306	-1.287	-4.335	0.527	2.918	-2.574
SLOPE:		0.222	0.628	0.563	0.576	1.257	0.969	0.827	1.153
CORR. COEFF.:		0.979	0.997	0.979	0.979	0.985	0.979	0.997	0.979
PRG. # 20									
TOTAL COUNT:	647	134.930	737.186	551.988	459.833	646.450	647.000	647.000	647.000
DEVN. OF TOTAL COUNT:		79.145%	-13.939%	14.685%	28.928%	0.085%	0.000%	0.000%	0.000%
INTERCEPT:		1.055	8.814	4.314	1.229	-0.339	5.057	7.735	1.729
SLOPE:		0.155	0.690	0.633	0.648	1.016	0.742	0.605	0.912
CORR. COEFF.:		0.998	0.978	0.998	0.998	0.990	0.998	0.978	0.998
PRG. # 21									
TOTAL COUNT:	556	155.459	560.736	415.957	359.471	555.583	556.000	556.000	556.000
DEVN. OF TOTAL COUNT:		72.040%	-0.852%	25.188%	35.347%	0.075%	-0.000%	-0.000%	0.000%
INTERCEPT:		0.754	4.956	2.017	0.330	-1.766	2.697	4.914	0.511
SLOPE:		0.229	0.679	0.614	0.625	1.117	0.821	0.673	0.966
CORR. COEFF.:		0.991	0.979	0.991	0.991	0.976	0.991	0.979	0.991

APPENDIX D (contd.)

	OBSERVED	ZIPF	ZWEBEN	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 22									
TOTAL COUNT:	2738	187.050	2866.117	2101.049	1696.145	2735.377	2738.000	2738.000	2738.000
DEVN. OF TOTAL COUNT:		93.168%	-4.679%	23.263%	38.051%	0.096%	-0.000%	-0.000%	0.000%
INTERCEPT:		1.282	28.476	14.404	4.240	-6.617	18.771	27.203	6.845
SLOPE:		0.048	0.600	0.541	0.553	1.103	0.705	0.573	0.892
CORR. COEFF.:		0.988	0.970	0.988	0.988	0.986	0.988	0.970	0.988
PRG. # 23									
TOTAL COUNT:	3024	241.516	2841.561	2050.610	1680.766	3021.030	3024.000	3024.000	3024.000
DEVN. OF TOTAL COUNT:		92.013%	6.033%	32.189%	44.419%	0.098%	0.000%	-0.000%	0.000%
INTERCEPT:		1.463	24.257	12.420	4.996	-5.855	18.315	25.814	8.988
SLOPE:		0.054	0.515	0.460	0.468	1.102	0.679	0.548	0.842
CORR. COEFF.:		0.996	0.976	0.996	0.996	0.993	0.996	0.976	0.996
PRG. # 24									
TOTAL COUNT:	490	100.215	554.790	423.986	349.577	489.520	490.000	490.000	490.000
DEVN. OF TOTAL COUNT:		79.548%	-13.222%	13.472%	28.658%	0.098%	-0.000%	0.000%	-0.000%
INTERCEPT:		6.669	6.652	2.748	-0.528	-1.387	3.176	5.875	-0.740
SLOPE:		0.170	0.779	0.719	0.741	1.073	0.831	0.688	1.039
CORR. COEFF.:		0.996	0.984	0.996	0.996	0.987	0.996	0.984	0.996
PRG. # 25									
TOTAL COUNT:	277	81.198	290.189	225.140	191.137	276.815	277.000	277.000	277.000
DEVN. OF TOTAL COUNT:		70.667%	-4.762%	18.722%	30.998%	0.067%	0.000%	0.000%	0.000%
INTERCEPT:		0.102	2.722	0.505	-1.336	-1.473	0.622	2.598	-1.937
SLOPE:		0.279	0.831	0.773	0.796	1.116	0.951	0.794	1.154
CORR. COEFF.:		0.965	0.957	0.965	0.965	0.942	0.965	0.957	0.965
PRG. # 26									
TOTAL COUNT:	150	85.869	149.820	115.763	108.757	149.887	150.000	150.000	150.000
DEVN. OF TOTAL COUNT:		42.741%	0.120%	22.825%	27.495%	0.076%	-0.000%	0.000%	0.000%
INTERCEPT:		0.658	2.086	0.887	0.534	0.603	1.150	2.089	0.736
SLOPE:		0.472	0.679	0.636	0.643	0.907	0.824	0.680	0.887
CORR. COEFF.:		0.978	0.960	0.978	0.978	0.973	0.978	0.960	0.978
PRG. # 27									
TOTAL COUNT:	175	81.158	150.988	121.797	112.206	174.849	175.000	175.000	175.000
DEVN. OF TOTAL COUNT:		53.661%	10.293%	30.402%	35.882%	0.086%	0.000%	-0.000%	-0.000%
INTERCEPT:		0.791	2.252	1.081	0.575	0.535	1.554	2.510	0.896
SLOPE:		0.373	0.614	0.560	0.569	0.932	0.805	0.684	0.887
CORR. COEFF.:		0.942	0.952	0.942	0.942	0.963	0.942	0.952	0.942
PRG. # 28									
TOTAL COUNT:	119	67.467	112.759	88.693	83.543	118.906	119.000	119.000	119.000
DEVN. OF TOTAL COUNT:		43.355%	5.245%	25.468%	29.796%	0.079%	0.000%	-0.000%	-0.000%
INTERCEPT:		0.402	1.675	0.608	0.283	0.666	0.816	1.767	0.403
SLOPE:		0.493	0.680	0.648	0.657	0.893	0.870	0.718	0.936
CORR. COEFF.:		0.987	0.964	0.987	0.987	0.964	0.987	0.964	0.987

APPENDIX F (cont..)

	OBSERVED	ZIPF	ZWEBER	ZIPF-A	ZIPF-B	ZWB-HLSD	MOD-ZIPF	MOD-ZWBN	MOD-ZPFB
PRG. # 29									
TOTAL COUNT:	340	90.623	313.905	241.661	206.189	339.739	340.000	340.000	340.000
DEVN. OF TOTAL COUNT:		73.346%	7.657%	28.923%	39.268%	0.077%	0.000%	0.000%	0.000%
INTERCEPT:		0.704	4.394	2.118	0.437	0.226	2.980	4.759	0.719
SLOPE:		0.210	0.613	0.561	0.576	0.983	0.790	0.664	0.949
CORR. COEFF.:		0.905	0.987	0.985	0.985	0.986	0.985	0.987	0.985
PRG. # 30									
TOTAL COUNT:	395	90.623	505.287	388.924	319.459	394.769	395.000	395.000	395.000
DEVN. OF TOTAL COUNT:		77.057%	-27.921%	1.538%	19.124%	0.058%	-0.000%	0.000%	0.000%
INTERCEPT:		0.902	7.671	3.872	0.566	0.651	3.932	5.996	0.700
SLOPE:		0.175	0.813	0.749	0.774	0.960	0.761	0.636	0.957
CORR. COEFF.:		0.991	0.985	0.991	0.991	0.990	0.991	0.985	0.991
PRG. # 31									
TOTAL COUNT:	1269	230.459	1667.191	1206.523	1005.724	1268.162	1269.000	1269.000	1269.000
DEVN. OF TOTAL COUNT:		81.839%	-31.378%	4.923%	20.747%	0.066%	0.000%	-0.000%	-0.000%
INTERCEPT:		1.469	14.835	7.798	3.604	-0.987	8.202	11.292	4.547
SLOPE:		0.122	0.718	0.637	0.648	1.039	0.670	0.546	0.817
CORR. COEFF.:		0.993	0.982	0.993	0.993	0.997	0.993	0.982	0.993
PRG. # 32									
TOTAL COUNT:	869	124.845	861.467	648.386	531.505	868.172	869.000	869.000	869.000
DEVN. OF TOTAL COUNT:		85.634%	0.874%	25.387%	38.837%	0.095%	0.000%	0.000%	-0.000%
INTERCEPT:		0.805	9.679	4.598	0.389	-2.390	6.163	9.764	0.636
SLOPE:		0.112	0.646	0.582	0.598	1.084	0.780	0.652	0.977
COPR. COEFF.:		0.907	0.908	0.987	0.987	0.982	0.987	0.988	0.987
PRG. # 33									
TOTAL COUNT:	339	109.961	563.707	428.062	355.843	338.719	339.000	339.000	339.000
DEVN. OF TOTAL COUNT:		67.503%	-66.303%	-26.272%	-4.968%	0.083%	-0.000%	-0.000%	-0.000%
INTERCEPT:		0.854	7.191	3.326	0.417	0.212	2.634	4.324	0.398
SLOPE:		0.254	1.009	0.988	1.015	0.982	0.782	0.643	0.967
CORR. COEFF.:		0.967	0.979	0.997	0.997	0.987	0.997	0.979	0.997
PRG. # 34									
TOTAL COUNT:	2799	176.426	3099.485	2280.555	1831.933	2796.951	2799.000	2799.000	2799.000
DEVN. OF TOTAL COUNT:		93.697%	-10.735%	18.523%	34.550%	0.073%	-0.000%	0.000%	-0.000%
INTERCEPT:		1.629	27.586	13.301	1.383	-13.313	16.325	24.912	2.113
SLOPE:		0.068	0.703	0.620	0.634	1.194	0.761	0.635	0.969
CORR. COEFF.:		0.914	0.992	0.984	0.984	0.992	0.984	0.992	0.984

```
C--     PROGRAM CORRESPONDING TO ALGOL DATA,
C--     WITH THE PARAMETERS NP,ND & NSAMPL CHANGED,
C--     RESPCTIVELY, TO 34,185 & 7, AND A FEW MORE APPROPPIATE
C--     CHANGES IN THE HEADINGS ETC,, THE PROGRAM APPLIES
C--     TO PL/I DATA.
        PROGRAM PROJCT
        PARAMETER NP=14,ND=20,NSAMPL=3
C--     NP=NUMBER OF PROGRAMS TAKEN FOR STUDY,
C--     ND=MAX(ETA1(I),I=1,NP),
C--     NSAMPL=SEQUENCE NUMBER OF THE SAMPLE PROGRAM (TABLE 3A).
        INTEGER SERIAL(ND),ACTUAL(ND),N1(NP),
       1 ETA1(NP),ETA2(NP),ETA1I,ETA2I
        REAL ZP(ND),ZW(ND),ZA(ND),ZB(ND),
       1 ZH(ND),MZ(ND),MW(ND),MB(ND),
       2 ZPI(NP),ZWI(NP),ZAI(NP),ZBI(NP),
       2 ZHI(NP),MZI(NP),MWI(NP),MBI(NP),
       3 ZPSL(NP),ZWSL(NP),ZASL(NP),ZBSL(NP),
       3 ZHSL(NP),MZSL(NP),MWSL(NP),MBSL(NP),
       4 ZPC(NP),ZWC(NP),ZAC(NP),ZBC(NP),
       4 ZHC(NP),MZC(NP),MWC(NP),MBC(NP)
        REAL MZIMN,MZISD,MZSLMN,MZSLSD,MZCMN,MZCSD,
       1    MWIMN,MWISD,MWSLMN,MWSLSD,MWCMN,MWCSD,
       2    MBIMN,MBISD,MBSLMN,MBSLSD,MBCMN,MBCSD
        COMMON/BLK1/ETA1I,ACTUAL
        COMMON/BLK2/NNP
C--     STATEMENT FUNCTION FOP THE  ZWEBEN-HALSTEAD MODEL.
        FZH(I,X)=2**(X*2**(-(I-1)/X))
        NNP=NP
1       FORMAT(3I)
2       FORMAT(20I)
31      FORMAT(//34X,'Table 1A'/
       1 19X'Observed Parameters of ALGOL Programs'/
       2 21X,'Program #',4X,'Eta1',3X,'Eta2',4X,'N1'/
       3 (25X,I2,'.',5X,3(I4,3X)))
41      FORMAT(//62X,'Table 3A'/
       1          60X,'Sample Result'/
       2 40X,'comparison of Various Models on Operator
       3 Distribution'/
       4 58X,'ALGOL Program #3'//
       5 8X,'SL. NO.',7X,'OBSERVED',6X,'ZIPF',7X,'ZWEBEN',
       6 6X,'ZIPF-A',6X,'ZIPF-B',5X,'ZWB-HLSD',4X,
       7    'MOD-ZIPF',4X,'MOD-ZWBN',4X,'MOD-ZPFB'/
       8 (8X,I4,'.'8X,I5,2X,8(4X,F8.3)))
42      FORMAT(8X,116('-'))
43      FORMAT(8X'TOTAL COUNT: 'I5,2X,8(4X,F8.3))
44      FORMAT(8X'DEVN. OF TOTAL COUNT:'3X,8(F8.3'%'3X))
45      FORMAT(8X'INTERCEPT:'10X,8(4X,F8.3))
46      FORMAT(8X'SLOPE:'14X,8(4X,F8.3))
47      FORMAT(8X'CORR. COEFF.:'7X,8(4X,F8.3))
51      FORMAT(//8X,'APPENDIX C'/
       1 45X,'SUMMARY OF THE RESULTS OF ALGOL PROGRAMS'/)
52      FORMAT('1'//8X'APPENDIX C (contd.)'/)
53      FORMAT(22X'OBSERVED',5X,'ZIPF',7X,'ZWEBEN'
       2 6X'ZIPF-A'6X'ZIPF-B'5X'ZWB-HLSD'4X
       3 'MOD-ZIPF'4X'MOD-ZWBN'4X'MOD-ZPFB')
54      FORMAT(8X,'PRG. #'I3/8X,9('-'))
61      FORMAT(//35X,'Table 4A'/
       1 10X,'Mean and Std. Dev. of the Intercept, Slope and
       2 Corr. Coeff.'/
       3 24X,'Corresponding to ALGOL Programs'//
```

```
00061           4 23X,'INTERCEPT',10X,'SLOPE',10X,'CORR. COEFF.'/
00062           5 21X,3('MEAN',5X,'SD',7X))
00063   62      FORMAT(
00064           1 8X'ZIPF',4X,3(4X,F6.3,2X,F6.3)/
00065           2 8X'ZWEBEN',2X,3(4X,F6.3,2X,F6.3)/
00066           3 8X'ZIPF-A',2X,3(4X,F6.3,2X,F6.3)/
00067           4 8X'ZIPF-B',2X,3(4X,F6.3,2X,F6.3)/
00068           5 8X'ZWB-HLSD',3(4X,F6.3,2X,F6.3)/
00069           6 8X'MOD-ZIPF',3(4X,F6.3,2X,F6.3)/
00070           7 8X'MOD-ZWBN',3(4X,F6.3,2X,F6.3)/
00071           8 8X'MOD-ZPFB',3(4X,F6.3,2X,F6.3))
00072           OPEN(UNIT=1,DEVICE='DSK',FILE='ET.DAT')
00073           OPEN(UNIT=2,DEVICE='DSK',FILE='AL.DAT')
00074           OPEN(UNIT=3,DEVICE='DSK')
00075           OPEN(UNIT=4,DEVICE='DSK')
00076           OPEN(UNIT=5,DEVICE='DSK')
00077           OPEN(UNIT=6,DEVICE='DSK')
00078           READ(1,1)((ETA1(I),ETA2(I),N1(I)),I=1,NP)
00079           CLOSE(UNIT=1)
00080           DO 101 I=1,NP
00081           SERIAL(I)=I
00082   101     CONTINUE
00083           WRITE(3,31)((SERIAL(I),ETA1(I),ETA2(I),N1(I)),I=1,NP)
00084           CLOSE(UNIT=3)
00085   C-------FILL UP THE ARRAY SERIAL.
00086           DO 102 I=1,NP
00087           ETA1I=ETA1(I) ; ETA2I=ETA2(I) ; N1I=N1(I)
00088           READ(2,2)(ACTUAL(J),J=1,ETA1I)
00089           TYPE *,ACTUAL
00090           XMETA=AMAX0(ETA1I,ETA2I)
00091   C--
00092           Z=FUNZH(N1I)
00093   C--     DETERMINE THE DISTRIBUTIONS OF THE OLD MODELS.
00094           DO 103 J=1,ETA1I
00095              ZP(J)=FLOAT(ETA1I)/J
00096              XM=FLOAT(ETA1I-J)/ETA1I
00097              ZW(J)=XMETA*(1.0+XM)/(J+XM)
00098              ZA(J)=XMETA/J
00099              IF(ETA2I.GT.ETA1I)ZB(J)=(FLOAT(ETA2I)-
00100           1  FLOAT(ETA2I-ETA1I)*(J-1.0)/(ETA1I-1.0))/J
00101              IF(ETA1I.GE.ETA2I)ZB(J)=ZP(J)
00102              ZH(J)=FZH(J,Z)
00103   103     CONTINUE
00104   C--     DETERMINE THE REGRESSION AND CORRELATION COEFFICINTS
00105   C--     CORRESPONDING TO THE OLD MODELS.
00106           CALL REGRSN(ZP,AZP,BZP,CZP,DZP,SZP)
00107           CALL REGRSN(ZW,AZW,BZW,CZW,DZW,SZW)
00108           CALL REGRSN(ZA,AZA,BZA,CZA,DZA,SZA)
00109           CALL REGRSN(ZB,AZB,BZB,CZB,DZB,SZB)
00110           CALL REGRSN(ZH,AZH,BZH,CZH,DZH,SZH)
00111   C--     DETERMINE THE DISTRIBUTIONS OF THE NEW MODELS.
00112           FACZP=N1I/SZP
00113           FACZW=N1I/SZW
00114           FACZB=N1I/SZB
00115           DO 104 J=1,ETA1I
00116              MZ(J)=FACZP*ZP(J)
00117              MW(J)=FACZW*ZW(J)
00118              MB(J)=FACZB*ZB(J)
00119   104     CONTINUE
00120           CALL REGRSN(MZ,AMZ,BMZ,CMZ,DMZ,SMZ)
```

```
00121           CALL REGRSN(MW,AMW,BMW,CMW,DMW,SMW)
00122           CALL REGRSN(MB,AMB,BMB,CMB,DMB,SMB)
00123           IF(I.NE.NSAMPL)GOTO 555
00124           WRITE(4,41)((SERIAL(J),ACTUAL(J),ZP(J),ZW(J),
00125           1                 ZA(J),ZB(J),ZH(J),MZ(J),MW(J),
00126           2                 MB(J)),J=1,ETA1I)
00127           WRITE(4,42)
00128           WRITE(4,43)M1I,SZP,SZW,SZA,SZB,
00129           1           SZH,SMZ,SMW,SMB
00130           WRITE(4,44)DZP,DZW,DZA,DZB,
00131           1           DZH,DMZ,DMW,DMB
00132           WRITE(4,45)AZP,AZW,AZA,AZB,
00133           1           AZH,AMZ,AMW,AMB
00134           WRITE(4,46)BZP,BZW,BZA,BZB,
00135           1           BZH,BMZ,BMW,BMB
00136           WRITE(4,47)CZP,CZW,CZA,CZB,
00137           1           CZH,CMZ,CMW,CMB
00138           CLOSE(UNIT=4)
00139   555     CONTINUE
00140   C--
00141           IF(I.EQ.1)WRITE(5,51)
00142           IF(I.NE.1.AND.MOD(I,7).EQ.1)WRITE(5,52)
00143           IF(I.EQ.1.OR.MOD(I,7).EQ.1)WRITE(5,53)
00144           WRITE(5,54)I
00145           WRITE(5,43)M1I,SZP,SZW,SZA,SZB,
00146           1           SZH,SMZ,SMW,SMB
00147           WRITE(5,44)DZP,DZW,DZA,DZB,
00148           1           DZH,DMZ,DMW,DMB
00149           WRITE(5,45)AZP,AZW,AZA,AZB,
00150           1           AZH,AMZ,AMW,AMB
00151           WRITE(5,46)BZP,BZW,BZA,BZB,
00152           1           BZH,BMZ,BMW,BMB
00153           WRITE(5,47)CZP,CZW,CZA,CZB,
00154           1           CZH,CMZ,CMW,CMB
00155           ZPI(I)=AZP; ZPSL(I)=BZP; ZPC(I)=CZP
00156           ZWI(I)=AZW; ZWSL(I)=BZW; ZWC(I)=CZW
00157           ZAI(I)=AZA; ZASL(I)=BZA; ZAC(I)=CZA
00158           ZBI(I)=AZB; ZBSL(I)=BZB; ZBC(I)=CZB
00159           ZHI(I)=AZH; ZHSL(I)=BZH; ZHC(I)=CZH
00160           MZI(I)=AMZ; MZSL(I)=BMZ; MZC(I)=CMZ
00161           MWI(I)=AMW; MWSL(I)=BMW; MWC(I)=CMW
00162           MBI(I)=AMB; MBSL(I)=BMB; MBC(I)=CMB
00163   102     CONTINUE
00164           CLOSE(UNIT=2)
00165           CLOSE(UNIT=4)
00166           CLOSE(UNIT=5)
00167   C--
00168           CALL MEANSD(ZPI,ZPIMN,ZPISD)
00169           CALL MEANSD(ZWI,ZWIMN,ZWISD)
00170           CALL MEANSD(ZAI,ZAIMN,ZAISD)
00171           CALL MEANSD(ZBI,ZBIMN,ZBISD)
00172           CALL MEANSD(ZHI,ZHIMN,ZHISD)
00173           CALL MEANSD(MZI,MZIMN,MZISD)
00174           CALL MEANSD(MWI,MWIMN,MWISD)
00175           CALL MEANSD(MBI,MBIMN,MBISD)
00176   C--
00177           CALL MEANSD(ZPSL,ZPSLMN,ZPSLSD)
00178           CALL MEANSD(ZWSL,ZWSLMN,ZWSLSD)
00179           CALL MEANSD(ZASL,ZASLMN,ZASLSD)
00180           CALL MEANSD(ZBSL,ZBSLMN,ZBSLSD)
```

```
00181            CALL MEANSD(ZHSL,ZHSLMN,ZHSLSD)
00182            CALL MEANSD(MZSL,MZSLMN,MZSLSD)
00183            CALL MEANSD(MWSL,MWSLMN,MWSLSD)
00184            CALL MEANSD(MBSL,MBSLMN,MBSLSD)
00185    C--
00186            CALL MEANSD(ZPC,ZPCMN,ZPCSD)
00187            CALL MEANSD(ZWC,ZWCMN,ZWCSD)
00188            CALL MEANSD(ZAC,ZACMN,ZACSD)
00189            CALL MEANSD(ZBC,ZBCMN,ZBCSD)
00190            CALL MEANSD(ZHC,ZHCMN,ZHCSD)
00191            CALL MEANSD(MZC,MZCMN,MZCSD)
00192            CALL MEANSD(MWC,MWCMN,MWCSD)
00193            CALL MEANSD(MBC,MBCMN,MBCSD)
00194    C--
00195            WRITE(6,61)
00196            WRITE(6,62)ZPIMN,ZPISD,ZPSLMN,ZPSLSD,ZPCMN,ZPCSD,
00197           1           ZWIMN,ZWISD,ZWSLMN,ZWSLSD,ZWCMN,ZWCSD,
00198           2           ZAIMN,ZAISD,ZASLMN,ZASLSD,ZACMN,ZACSD,
00199           3           ZBIMN,ZBISD,ZBSLMN,ZBSLSD,ZBCMN,ZBCSD,
00200           4           ZHIMN,ZHISD,ZHSLMN,ZHSLSD,ZHCMN,ZHCSD,
00201           5           MZIMN,MZISD,MZSLMN,MZSLSD,MZCMN,MZCSD,
00202           6           MWIMN,MWISD,MWSLMN,MWSLSD,MWCMN,MWCSD,
00203           7           MBIMN,MBISD,MBSLMN,MBSLSD,MBCMN,MBCSD
00204            CLOSE(UNIT=6)
00205            END
00206    C-----------------------------------------------------------
00207    C--     SUBROUTINE TO DETERMINE THE REGRESSION & CORRELATION
00208    C--     COEFFICIENTS AND THE DEVIATION OF THE TOTAL COUNT.
00209            SUBROUTINE REGRSN(Y,A,B,C,D,SUMY)
00210            COMMON/BLK1/N,IX
00211            DIMENSION IX(1),Y(1)
00212            SUMX=0.; SUMY=0.; SUMXY=0.; SUMXSQ=0. ;SUMYSQ=0.
00213            DO 101 I=1,N
00214            SUMX=SUMX+IX(I)
00215            SUMY=SUMY+Y(I)
00216            SUMXY=SUMXY+IX(I)*Y(I)
00217            SUMXSQ=SUMXSQ+IX(I)**2 ; SUMYSQ=SUMYSQ+Y(I)**2
00218    101     CONTINUE
00219            B=(N*SUMXY-SUMX*SUMY)/(N*SUMXSQ-SUMX*SUMX)
00220            A=(SUMY-B*SUMX)/N
00221            C=(N*SUMXY-SUMX*SUMY)/SQRT((N*SUMXSQ-SUMX*SUMX)*
00222           1 (N*SUMYSQ-SUMY*SUMY))
00223            D=100.0*(SUMX-SUMY)/SUMX
00224    C--     A=INTERCEPT , B=SLOPE , C=CORR. COEFF.,
00225    C--     D=PERCENTAGE DEVIATION OF THE TOTAL COUNT.
00226            RETURN ; END
00227    C-----------------------------------------------------------
00228    C--     SUBROUTINE TO DETERMINE THE MEAN & STD. DEVN.
00229            SUBROUTINE MEANSD(X,A,B)
00230            COMMON/BLK2/N
00231            DIMENSION X(1)
00232            SUM=0.0 ; SUMSQ=0.0
00233            DO 101 I=1,N
00234    101     SUM=SUM+X(I)
00235            A=SUM/N
00236            DO 102 I=1,N
00237    102     SUMSQ=SUMSQ+(X(I)-A)**2
00238            B=SQRT(SUMSQ/N)
00239            RETURN ; END
00240    C-----------------------------------------------------------
```

```
00241    C--      FUNCTION SUBPROGRAM CORR. TO THE ZWBN-HLSTD MODEL.
00242    C--      ITERATIVE SOLUTION TO DETERMINE X1, GIVEN N1 & ETA1.
00243             REAL FUNCTION FUNZH(N1)
00244             IMPLICIT DOUBLE PRECISION(A-H,O-Z)
00245             COMMON/BLK1/ETA1I
00246             INTEGER ETA1I
00247             F(I,X1)=2**(X1*2**(-(I-1)/X1))
00248    C--------MAKE A GUESS.
00249             X1=1.50*DLOG(DFLOAT(ETA1I))
00250             KOUNT=0
00251    101      CONTINUE
00252             IF(SUM.LE.1.005*N1.AND.SUM.GE.0.995*N1)KOUNT=KOUNT+1
00253             IF(KOUNT.GE.100)GO TO 121
00254             SUM=0.0
00255             DO 102 I=1,ETA1I
00256             SUM=SUM+F(I,X1)
00257    102      CONTINUE
00258             IF(SUM.GT.N1)GO TO 103
00259    C--------ELSE DO AS FOLLOWS
00260             IF(SUM.LE.N1.AND.SUM.GE.0.999*N1)GO TO 104
00261             IF(SUM.LT.0.999*N1.AND.SUM.GE.0.995*N1)X1=1.0001*X1
00262             IF(SUM.LT.0.995*N1.AND.SUM.GE.0.99*N1)X1=1.0002*X1
00263             IF(SUM.LT.0.99*N1.AND.SUM.GE.0.95*N1)X1=1.0005*X1
00264             IF(SUM.LT.0.95*N1.AND.SUM.GE.0.90*N1)X1=1.001*X1
00265             IF(SUM.LT.0.90*N1.AND.SUM.GE.0.75*N1)X1=1.002*X1
00266             IF(SUM.LT.0.75*N1.AND.SUM.GE.0.25*N1)X1=1.005*X1
00267             IF(SUM.LT.0.25*N1)X1=1.01*X1
00268             GO TO 101
00269    103      CONTINUE
00270             IF(SUM.GT.N1.AND.SUM.LE.1.001*N1)GO TO 104
00271             IF(SUM.GT.1.001*N1.AND.SUM.LE.1.005*N1)X1=0.9999*X1
00272             IF(SUM.GT.1.005*N1.AND.SUM.LE.1.01*N1)X1=0.9998*X1
00273             IF(SUM.GT.1.01*N1.AND.SUM.LE.1.05*N1)X1=0.9995*X1
00274             IF(SUM.GT.1.05*N1.AND.SUM.LE.1.10*N1)X1=0.999*X1
00275             IF(SUM.GT.1.10*N1.AND.SUM.LE.1.25*N1)X1=0.998*X1
00276             IF(SUM.GT.1.25*N1.AND.SUM.LE.2.00*N1)X1=0.995*X1
00277             IF(SUM.GT.2.00*N1)X1=0.99*X1
00278             GO TO 101
00279    104      CONTINUE
00280             PRINT 105,X1,SUM,N1
00281    105      FORMAT(X'X1='G10.3/X'SUM & N1 ='2G12.5)
00282             FUNZH=X1
00283             RETURN
00284    121      FUNZH=X1
00285             PRINT 105,X1,SUM,N1
00286             PRINT 122,KOUNT
00287    122      FORMAT(X'5% ACCURACY.....KOUNT='I3)
00288             RETURN
00289             END
```