DOCUMENT RESUME

ED 271 100 IR 012 177

AUTHOR Kaiser, Javaid
TITLE Identification of Factors That Affect Software Complexity.
PUB DATE Dec 85
NOTE 93p.; Requirement for master's degree, University of Kansas. Small print in the appendixes.
PUB TYPE Dissertations/Theses - Undetermined (040) -- Tests/Evaluation Instruments (160)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS Adults; *Computer Software; *Difficulty Level; Factor Analysis; Integrated Activities; Programing; Questionnaires; Research Methodology; Surveys; *Systems Analysis; Tables (Data)
IDENTIFIERS *Software Design; *Software Maintenance; System Dynamics

ABSTRACT

A survey of computer scientists was conducted to identify factors that affect software complexity. A total of 160 items were selected from the literature to include in a questionnaire sent to 425 individuals who were employees of computer-related businesses in Lawrence and Kansas City. The items were grouped into nine categories called system planning, system characteristics, system design, system testing, system documentation, system correctness, system clarity, programming style, and system management. Factor analysis and central tendency measures were used to analyze the data. Based on 147 respondents, 98% of the items were found to affect system complexity. Large item variance was attributed to the lack of formal education or experience of respondents in some areas of software development. The factors extracted in this study were found to be useful in regrouping items to determine the complexity of system attributes, system components, or of various phases of system development. The need for a weighting scheme to add component complexity to determine the overall system complexity was identified. It is suggested that replication of this study on a larger scale would prove useful and that the complexity of various system components be weighted to determine the overall complexity of the system. A list of references, a copy of the questionnaire, and factor matrices are also provided. (JB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED271100

IDENTIFICATION OF FACTORS THAT AFFECT SOFTWARE COMPLEXITY

U.S. DEPARTMENT OF EDUCATION
OERI
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☒ This document has been reproduced as received from the person or organization originating it
☐ Minor changes have been made to improve reproduction quality
• Points of view or opinions stated in this document do not necessarily represent official position or policy

By

Javaid Kaiser

B.S., University of Punjab, 1970
M.A., University of Punjab, 1972
M.S., University of Kansas, 1980
Ph.D., University of Kansas, 1984

Submitted to the Department of Computer Science and to the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Javaid Kaiser
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)"

Thesis Committee:

Chairman

Thesis defended: December, 1985.

-R012177

ABSTRACT

A survey was conducted on computer scientists to identify factors that affect software complexity. A total of 160 items were selected from the literature to include in the questionnaire. The items were grouped into nine categories called system planning, system characteristics, system design, system testing, system documentation, system correctness, system clarity, programming style, and system management. Factor analysis and central tendency measures were used to analyse data.

Based on the data on 147 respondents, 98% of the items were found to affect system complexity. Large item variance was attributed to the lack of formal education or experience of respondents in some areas of software development. The categories used in the present study, were not found mutually exclusive. The factors extracted in this study were found useful in regrouping items to determine complexity of system attributes, system components, or of various phases of system development. The need for a weighting scheme to add component complexity to determine the overall system complexity was identified. The replication of this study on a larger sample was suggested.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1

INTRODUCTION

Software complexity is a general, non-standard, and relative term describing the composition of the system. It is a relative term because it does not have an absolute value assigned to it. A system with high software complexity may be less complex than another system. The term is non-standard as it is not delimited in scope and may be used on different occasions to mean different things. A system with a large code having several interliking modules may be considered a complex system. On the other hand a short program with a difficult algorithm may equally be called complex.

Because of the generality associated with the term, software complexity may be used to define complexity level for various components of the system i.e. algorithm complexity, code complexity, programming language complexity, module linkage complexity, or I/O complexity. Software complexity may also be used as an index for various stages of system development and maintenance. These developmental phases may be system planning, system characteristics, system design, system testing, system management, system coding, and programming style.

The term 'software complexity' gives an overall view of the complexity level in a system and does not

necessarily mean that all system components have the same complexity level. For example, a system may have a complex code but a simple testing procedure or may have a complex design but simple code. Likewise, various stages of system development may not have the same complexity level. A system may be very simple at the design stage but very complex in the testing phase. Therefore, the term 'software complexity' gives a general impression about the system but does not completely describe all its attributes.

The complexity of a system is very much affected by the way the system is designed, coded, and tested. Human factors including programming style, language characteristics, and hardware limitations also affect the complexity of the system. It is a known fact that maintenance of a complex system is a difficult and expensive task. This concern has resulted in a research activity to find ways to develop less complex systems. The present study is a step forward in this direction. The identification of factors that affect software complexity will help computer scientists to build less complex systems.

Statement of the Problem

The problem addressed in this research study was stated as 'Identification of factors that affect software

complexity'. The term 'factors' as used in the statement, referred to the underlying dimensions that supposedly affect software complexity. The term software complexity was used interchangeably with system complexity and referred to large programs only. Complexity was defined in terms of system maintenance and included repair maintenance, adaptive maintenance, and productivity maintenance. The definitions of these terms are given below:

System Maintenance:

System maintenance was defined as an amouont of effort needed to add, delete or modify segment(s) of a program. A program that takes less effort in modification is less complex than the one that takes more effort.

Repair Maintenance:

Repair maintenance is the maintenance needed to correct logic errors discovered in a program after it has been released into production (Vessey and Weber, 1983).

Adaptive Maintence:

Adaptive maintenance is the maintenance needed by a program to better meet users' needs (Lientz and Swanson, 1981).

Productivity Maintenance:

Productivity maintenance is the maintenance needed to improve the efficiency of the program in terms of consumption of resources (Lientz, Swanson, and Tompkins, 1978).

Significance of the Study

Several attempts have been made in the past to identify the factors that affect software complexity and to develop a procedure to measure the complexity level in a system (Curtis, Sheppard, Millman, Borst, and Love, 1979; Lientz et. al., 1981; Vessey et. al., 1983).

The uniqueness of this study stems from rationale that overall system complexity cannot be measured accurately unless the complexity of its major components is measured correctly, because system components tend to have different levels of complexity. In order to produce an index for the overall complexity of a system, component complexity should be measured such that it has an additive property. The other unique characteristic of this study is the belief that overall system complexity is not of much value compared to the system component complexity. It is the component complexity that determines the cost of particular system maintenance.

The significance of this study therefore, lies in its attempt to identify major components of the system and the various elements that affect the complexity of those components.

Purpose of the Study

The purpose of this study was to develop an exhaustive list of elements that affect system complexity measured in terms of system maintenace and then to use these elements to identify underlying dimensions called factors, that affect system complexity of various system components. The factors identified based on statistical significance were considered to provide a framework to conduct experimental study to determine the impact of each of these factors on various softwares under various conditions and to help in developing procedures to make less complex systems.

Chapter 2

REVIEW OF LITERATURE

Software complexity may have different meanings in different contexts. It may refer to the complexity of the algorithm, complexity of developing the system, complexity of interlinking modules, or the complexity of I/O interface. In the present study, software complexity means the amount of the effort needed to do repair maintenance.

Software complexity is affectted by several factors. Structured programming, module programming, and top down programming have considerably reduced the software complexity of systems (Al-Suwaiyel, 1983). These factors contribute to the simplicity and clarity of code and therefore make it easy to perform repair maintenance (Canning, 1972). Formal proofs of correctness also reduce software complexity (DeMillo, Lipton, and Perlis, 1979). Personnel characteristics of programmers affect system complexity as well (Weinberg, 1971). Endres (1975) found empirical evidence that high quality programmers have less cost of repair maintenance. However, more formal and operational measures of module complexity, programming style, and programmer quality are needed to reduce system complexity (Vassey et.al., 1983).

The issue of software complexity is tied up with the repair maintenance of the system. Vassey et. al., (1983) considered repair maintenance a function of system

complexity. Repair maintenance implies modifications in the program already in production to (1) fix logical errors discovered after its release, (2) to improve the operational efficiency of the program by economizing resources, or (3) to meet users' needs in a better way (Lientz et. al., 1981).

The complexity measure provides an index of relative cost to implement or comprehend a system. Rising cost of software development and maintenance has aroused interest in tools and measures to quantify and analyse software complexity (Jensen and Nairavan, 1985). Leintz et. al. (1978) found that the repair maintenance cost may go from 40% to 75% of the total life cycle of a system. Davis (1974) discovered the relationship between system complexity and the entropy. He found that complex systems undergo greater entropy. The high correlation between system complexity and repair maintenance as well as the need to reduce the cost of system maintenance has triggered tremendous amouont of research activity on issues related to system complexity (Thayer, Lipow, and Nelson, 1977).

The measurement of software complexity is still in its infancy (DeMillo, et. al., 1980). It is generally measured by the number of modules, size of modules, and inter-linkage of modules (Al-Suwaiyel, 1983). Software metrics are also used in determining the system complexity. A strong relationship between software metrics and system

complexity has been identified by McCabe (1976), Henry, Kafura, and Harris (1981), Ottenstein (1981), and Schneider (1981). In spite of these indices, it is very difficult to determine system complexity accurately because of less precise factors that determine such complexity. Such factors include but are not limited to ease of design, clarity of code, ease in understanding inter-module communication, programming style, and team work of programmers (Pashtan, 1985).

Types and Levels of Complexity

Al-Suwaiyel (1983) has categorized software complexity into inherent complexity and complexity of developing the system. Inherent complexity deals with the complexity of the algorithm and depends on the computational model used for the solution (Savage, 1976). It may be defined as the minimum over all the complexities of all the algorithms available as solution. Inherent complexity is generally measured in terms of time and space needed for execution. Quality and effectiveness of design and the total project cost can best be determined through inherent complexity.

The second type of complexity that deals with system development have attributes like size of the system, number of modules, number of functions within the system and the linkage between modules. It is heavily influenced by

system design methods that include flow oriented methods, data structure oriented methods, and perspective methods (Peter, 1981). System development complexity is independent of the inherent complexity of a system.

Software complexity may also be categorized as (1) logical, (2) structural, or (3) psychological. Logical complexity deals with program control graph and may be measured by a graph model called Cyclomatic (McCabe, 1976). Structural complexity deals with size and linkages within the sytem. Psychological complexity, on the other hand, refers to program comprehensibility.

The complexity level in a system may be documented in several ways. Vassey et. al., (1983) based his classification on the length of code. A system with a code length of 1 to 300 lines was called a simple system. Code length of 301 to 600 lines represented a moderately complex system while systems with more than 600 lines were labeled as complex systems.

Software Metrics

The software metrics that are most commonly used in determining software complexity have been reviewed briefly in the following text.

Program Length:

This is the most commonly used metric and is denoted by N such that

$$N = N_1 + N_2$$ where

N_1 and NN_2 represent total number of operator and operand occurances, respectively. This metric was first introduced by Halstead (1977) to measure software complexity. Program length can also be measured as

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

n_1 and n_2 represent unique operators and opernads in the system, respectively.

If size of the system is the sole determinant of system complexity, program length is the most suited metric. Davis (1984) criticised the use of operator and operand counts and considered it as an obsolete measure to determine software complexity. Jensen (1982) however, has proposed anohter alternative to measure program length and it is represented by the follcwing expression.

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

This equation is considered an improvement over Halstead's original proposal (Jensen et. al., 1985). Jensen et. al. (1985) also reported that the normalized difference between the program length and Halstead's length estimator is higher than reported by Halstead (1977). It, therefore, confirmed the findings of Shen, Conte, and

Dunsmore (1982) that program length is an imprecise measure to determine software complexity.

Program Volume:

Program Volume is the size of the program and is denoted by V. Mathematically,

$$V = (N_1 + N_2) \log_2 (n_1 + n_2)$$ where

N_1 and N_2 are the number of operators and operands that occur in a system, respectively. The n_1 and n_2 refer to unique number of operators and operands, respectively (Halstead, 1977).

Programming Effort:

The amount of programming effort needed to perform repair maintenance on a system is considered an index of the complexity of that system. It is denoted by E and may be expressed as

$$E = V (n_1 / n_2) (N_2 / 2)$$ where

n_1 , n_2 , and N_2 represent number of unique operators, number of unique operands, and the number of occurances of opernads, respectively. This metric was also proposed by Halstead (1977).

Programming effort may also be interpreted as decision to implement an algorithm. It is highly correlated with program length and program volume measures. Programming effort, therefore, is also algebrically equivalent to

$$E = (n_1 N_2 V) (36 n_2)$$ where

V refers to program volume. The notation n_1, n_2, and N_2 have already been defined.

Shen et. al. (1983) has developed a linear relationship between programming time and programming effort. It is mathematically represented as

$$E = V^2 / V^* \qquad \text{where } V = N \log_2$$

In this equation, effort refers to elementary mental discriminations, V to the number of mental comparisons needed to write a program of length N. V^* refers to program potential volume (smallest value).

Pashton (1985) has used programming effort metric to compare process model and monitor model in the design of operating systems.

McCabe's Metric:

McCabe's metric denoted by MC deals with program control complexity. According to this scheme, lineraly independent control paths are determined in a connected graph representing the software system. Mathematically,

$$MC = e - v + 2 \qquad \text{where}$$

e and v are the edges and vertices of the graph (McCabe, 1976). The metric MC may also be computed readily by the equation

$$MC = D + 1 \qquad \text{where}$$

D denotes the number of decision nodes in the graph.

The correlation of program length, program volume, and programming effort with McCabe's complexity measure (MC) was found low and confirmed the hypothesis that high correlation between MC and N is typical of decision bound systems (Henry et. al., 1981). Prather (1984) considered McCabe's metric insensitive to restructuring of code, nesting level in the program, and found it correltaed highly with the length of the program.

Bond Metrics:

This metric was introduced by Belady (1980) and refers to the average level of nesting or average width of the control graph. It is also called Belady's Bond Metric and is denoted by B. Mathematically,

$$B = \left(\sum_{i} i \, L(i) \right) / K$$ where

K refers to the number of nodes in the control graph while L(i) denotes the number of nodes at level i. Jensen et. al., (1985) found consistently high correlation between McCabe's MC and Belady's B measures. However, because of less precision of B over MC and because of the computational ease, MC was considered a better estimate of system complexity.

Program Chunk:

Davis (1984) has introduced a new metric called program chunk to measure software complexity. The rationale of this appraoch is that experienced programmers

use chunks to understand a program while beginners concentrate on individual statements (Shneiderman, 1976b). Therefore, the cost of repair maintenance can best be estimated if system complexity is expressed in terms of chunk complexity.

Based on the work of Woodfield (1980) on modules and their inter-relationship, Davis (1984) proposed the following equation to determine chunk complexity.

$$C = {}^{n}({}^{m}C_i \, R^j) \qquad \text{where}$$

C = Complexity of the chunk i,

m = Number of other chunks affected by the chunk i,

n = Number of chunks, and

R = 2/3, a constant.

The constant R is based on the assumption that the chunk is reviewed as many times as there is inter-relationship with other chunks and that the complexity of the chunk in terms of understanding it, decreases with every subsequent review.

From the perspective of the maintenance programmer, the effort required to modify the program depends on whether the code is familiar or unfamiliar. The idea of chunks, therefore, may easily be extended to familiar and non-familiar chunk. A chunk is considered familiar if it has a high frequency of occurance. The formation of chunks recognizable by a programmer is a function of programming language and the application environment. It is because

semantic representation is formed by the syntactic knowledge of the programming language. For example, the recall of statements to switch the values of two variables is spontaneous to an experienced programmer. Moreover, the chunks that are meaningful can be remembered better (Shneiderman, 1976).

The identification of chunks and defining their boundaries is a difficult task. Norcio (1980) and Brotsky (1981) have addressed this issue in detail.

Chunks may be classified as mandatory or non-mandatory (Mayer, 1979). Mandatory chunks are a set of statements that have to occur together. For example, opening and closing of a FOR loop. Soloway (1982) however, has categorized chunks as strategic, tactical, and implementation.

The overall complexity of a system, according to this metric, may be determined by the complexity of chunks and their inter-relationship. The latter is also called program structure.

In spite of large number of research efforts, the software metrics that are available are not precise enough to accurately measure the system complexity. Moreover, the existing literature on the subject does not provide enough direction to rank these metrics in terms of their efficiency. The metrics of Halstead (1977) that are supported by Sanshara, Vehara, and Ohkawa (1981) and Curtis

et. al., (1979) are criticised by Baker and Zweben (1980) for being insensitive to program nesting and program structure. Similarly, the metrics of McCabe (1976) and Belady (1980) are considered too coarse to tap the intricasies of the program (Prather, 1984). The chunk complexity metric that taps the cognitive process needed to understand programs look promising but needs further research.

Chapter 3

METHODOLOGY

This chapter describes in detail the process used to identify factors that reduce software complexity measured in terms of repair maintenance.

Development of the Questionnaire

The first task of this study was to review the existing literature on software complexity and to identify elements that supposedly affect the complexity of the system. After a thorough review, 160 items were selected as a potential set of elements that affect software complexity measured in terms of system maintenance. The items were generally drawn from Arthur (1983), Bersof, Handerson, and Siegel (1980), Dunn and Ullman (1982), Kernigham and Planger (1974), London (1974), Meek and Heath (1981), and Tassel (1978).

Based on content, the items were categorized into nine logical categories named system planning, system characteristics, stystem design, system testing, system documentation, system correctness, system clarity, programming style, and system management. System planning category included items related to system specifications and overall planning considerations. System characteristics

category included terms that defined various attributes of the system like efficiency, portability, maintainability, modularity, reliability and so on. Items related to design considerations were grouped under system design category. System testing category included items on testing options and tools. Items that heavily influenced the correctness of the system output were grouped under system correctness. The items that influenced writing simple and easy to understand code were grouped under the program clarity category. Programming style elements describing Do's and Don'ts were grouped under programming style category. Items describing management of systems from planning to the completion stage were grouped under system management.

Items were written like a Likert Scale to provide enough variance. Five choices numbered 1 to 5 were given to respondents on each item. The choices were labeled as strongly agree, agree, neither agree nor disagree, disagree, and strongly disagree. For consistency considerations, the format of these choices was kept the same throughout the questionnaire. This forced some of the items to be worded negatively and they were spread randomly within the category to which they belonged.

A statement describing the content of each category preceded before the items. In each lead paragraph, it was

explicitly mentioned that the system complexity should be interpreted in terms of system maintenance.

After the questionnaire was developed, a cover letter was prepared describing the purpose of the study, definition of the term 'complexity', and the use of the data collected. The need for respondents' cooperation was also emphasized. This cover letter was used as the front page of the questionnaire.

The last section of the questionnaire requested demographic information on respondents. It included questions on sex, education, type of employment, nature of job, job designation, and computer related experience. This section was included to identify and exclude from study those respondents who were associated with software firms but were not necessarily computer scientists. For example, a Data Processing Manager who is an administrator working in a software firm may not have any formal training in the field of computer science.

A copy of the questionnaire including cover letter and demographic section is given in Appendix A.

Sampling

Because of the length of the questionnaire, the cooperation of computer related businesses in administering the questionnaire to their employees was not frequently

available. Since it was an exploratory study and intended to be exhaustive, the reduced version of the questionnaire was not considered appropriate. Therefore, the whole questionnaire was mailed out or was delivered in person to businesses in Lawrence and Kansas City without any regard to sampling strategy.

A total of 425 questionnaires were distributed of which 152 were received in complete form. The completed questionnaires were examined to identify the ones that were completed by computer professionals. This task was accomplished with the help of demographic information collected on respondents on the last page of the questionnaire. Seven of the completed questionnaires did not have complete demographic information to identify their status as computer scientists. These questionnaires were therefore excluded from the analysis leaving behind a sample of 147.

Data Analysis

The information from the completed questionnaires was entered on to a computer system. Item statistics in terms of means and standard deviation was computed for each item. Items with a mean value of 3.0 were considered non-discriminatory in terms of increasing or decreasing system complexity. Items with a standard deviation of 1.0 or

greater were either poor discriminators or the respondents did not have the theoretical or practical background of concepts used in those statements.

Factor analysis was performed on each of the nine categories separately. A principal component solution was performed on each category to determine the total number of factors. The factors with eigenvalue of 1.0 or greater were considered significant. After this initial analysis for each category, factor analysis was repeated several times for each category by varying the number of factors to be extracted and examining rotated and unrotated solutions, until such a time that a final solution was obtained. A solution was considered final if it was the most logical and meaningful in terms of naming factors and cross-loadings. Varimax rotation was used whenever a factor matrix was rotated. The assignment of items to factors depended on item loadings. The item was assigned to a factor on which it loaded the highest. Items with a loading of less than .40 were considered insignificant and were not assigned to any factor. Statistical Package for Social Sciences (SPSS, 1983) was used to perform statistical analysis.

Chapter 4

RESULTS

This chapter includes the results obtained from data analysis on items related to software complexity. The items, divided into nine categories called system planning, system characteristics, system design, system testing, system documentation, system correctness, system clarity, programming style, and system management, were analyzed within the respective categories.

Table 1 describes the item means and standard deviations on all the items. The category to which items belong is identified on the top of each column. The analysis revealed that all the items on system planning had a mean value of less than 3.0. Item 15 was close to 3.0 and was considered a non-discriminator of software complexity. Items 3 and 14 had a standard deviation greater than 1.0.

Item statistics on items related to system characteristics revealed that items 2 and 14 have means greater than 3.0. This happened because these items were negatively worded. On the remaining items, the mean value was less than 3.0. In terms of standard deviation, items 1 to 5, 8, and 14 had a standard deviation of greater than 1.0.

Table 1

ITEM STATISTICS

System Planning

ITEMS	MEAN	S.D.
1	1.27	0.45
2	1.42	0.70
3	2.19	1.17
4	1.92	0.94
5	2.12	0.95
6	1.58	0.95
7	1.92	0.98
8	2.00	0.85
9	2.17	0.64
10	2.27	0.83
11	2.42	0.99
12	2.00	0.89
13	2.04	0.87
14	2.58	1.03
15	2.92	0.89
16	2.00	0.85
17	2.12	0.86

System Characteristics

ITEMS	MEAN	S.D.
1	2.19	1.10
2	3.50	1.11
3	2.04	1.15
4	2.46	1.27
5	2.85	1.16
6	1.46	0.71
7	2.58	0.99
8	1.96	1.02
9	2.08	0.85
10	1.77	0.77
11	2.23	0.91
12	1.54	0.81
13	1.42	0.58
14	3.42	1.24
15	1.58	0.76

TABLE 1 (Contd.)

System Design			System Testing		
ITEMS	MEAN	S.D.	ITEMS	MEAN	S.D.
1	3.85	1.05	1	1.85	0.68
2	2.86	1.21	2	2.39	0.64
3	2.12	0.71	3	2.04	0.60
4	2.19	0.75	4	2.08	0.89
5	1.23	0.43	5	2.00	0.80
6	2.35	0.69	6	2.00	0.89
7	1.85	0.78	7	2.12	0.77
8	2.58	0.86	8	1.73	0.67
9	2.23	0.71	9	2.39	0.75
10	2.85	0.83	10	2.04	0.82
11	2.69	1.23	11	2.58	1.27
12	3.00	0.80	8	1.73	0.67
9	2.23	0.71	9	2.39	0.75
10	2.85	0.83	10	2.04	0.82
11	2.69	1.23	11	2.58	1.27
12	3.00	0.80	16	2.48	1.16
17	1.96	1.08	17	2.04	0.54
18	2.23	0.86	18	1.80	0.65
19	1.85	0.88	19	2.40	0.71
20	2.54	0.91			
21	3.58	0.81			
22	3.46	1.14			
23	3.73	1.04			

TABLE 1 (Contd.)

System Documentation

ITEMS	MEAN	S.D.
1	3.80	1.35
2	1.80	0.76
3	1.80	0.76
4	1.76	0.78
5	1.68	0.63
6	1.68	0.56
7	1.52	0.71
8	1.88	0.68
9	2.67	0.82
10	1.56	0.65
11	1.84	0.75
12	1.32	0.56
13	2.20	0.96
14	3.40	0.96
15	3.24	1.05
16	1.40	0.58

System Correctness

ITEMS	MEAN	S.D.
1	2.24	0.78
2	2.00	0.82
3	1.77	0.65
4	2.08	0.80
5	4.04	0.87
6	2.23	0.95
7	2.19	0.85
8	1.77	0.71
9	1.60	0.76
10	1.89	0.77
11	1.92	0.89
12	2.81	0.90
13	3.40	0.96
14	1.65	0.75
15	1.88	0.73
16	1.54	0.81
17	2.27	0.96
18	3.31	0.88
19	2.19	0.69
20	1.84	0.75

TABLE 1 (Contd.)

System Clarity

ITEMS	MEAN	S.D.
1	1.96	0.72
2	4.19	0.80
3	4.23	0.91
4	1.85	0.97
5	1.69	0.55
6	2.00	0.85
7	3.08	1.09
8	2.00	0.85
9	1.65	0.49
10	2.04	0.72
11	2.12	1.03
12	2.65	1.13
13	1.85	0.54
14	1.73	0.60
15	1.54	0.76
16	2.12	0.71

Programming Style

ITEMS	MEAN	S.D.
1	2.00	1.06
2	2.28	0.98
3	1.89	0.82
4	2.58	1.14
5	3.62	1.20
6	3.31	0.93
7	2.46	0.95
8	1.73	0.67
9	1.69	0.68
10	3.58	0.90
11	2.27	0.67
12	2.00	1.04
13	2.62	0.85
14	1.77	0.59
15	1.73	0.60
16	2.15	0.78
17	2.04	0.66
18	2.15	0.78
19	3.81	0.75
20	2.92	0.80

TABLE 1 (Contd.)

System Management

ITEMS	MEAN	S.D.
1	2.23	0.96
2	2.19	0.90
3	1.69	0.62
4	2.77	1.03
5	2.77	0.95
6	2.12	0.86
7	2.58	1.17
8	1.65	0.69
9	2.19	0.94
10	1.92	0.63
11	2.35	0.69
12	2.77	1.03
13	2.15	0.78
14	3.65	1.23

The analysis of system design items revealed that items 1, 16, and 21 to 23 had a mean value greater than 3.0. Of these items, 1, 16, 21, and 23 were negatively worded and this kind of outcome was expected. Item 22, however, was found inappropriate to determine complexity because of its high mean value. The items, 1, 2, 11, 17, 22, and 23, had a standard deviation greater than 1.0 and reflected large variances present in the opinions of respondents.

Item 15 was the only item on the list of items on system testing that had a mean value of greater than 3.0. It happened because of the negatively worded statement. It was also observed that the items 11, 14, and 19 had high standard deviations ($S \geq 1.0$).

The analysis of system documentation items revealed that items 1, 14, and 15 were negatively worded and therefore, showed a high mean value ($X > 3.0$). Items 1 and 15 were found to have a high standard deviation value ($S > 1.0$).

The analysis of items under system correctness category identified items 5, 13, and 18 with high mean values ($X > 3.0$). All these three items happened to be negatively worded statements. Regarding standard deviation, all items behaved as expected ($S < 1.0$).

Items 2, 3, and 7 related to system clarity had high mean values because of their negatively worded stem. The

standard deviation of items 7, 11, and 12 was higher than 1.0.

The analysis of programming style items identified items 5, 6, 10, and 19 with high mean value ($X > 3.0$). Content of these items revealed that they were negatively worded. Standard deviation on items 1, 4, 5, and 12 was found greater than 1.0.

Item 14 related to system management had a high mean value ($X > 3.0$) because of its negatively worded statement. The remaining items of this category had a mean value of less than 3.0. Items 4, 7, and 12 were found to have standard deviation of greater than 1.0.

Factor analysis was performed on items for each of the nine categories identified earlier. The results of the analysis are summarized below under the subheading representing each category.

System Planning

The principal component solution of items on system planning identified six significant factors. They together explained 77.4% of the total variance. The factors, their eigenvalues, and the percent of variance explained by each factor is given in Table 2.

After exploring several possibilities, the solution with 3 factors was found meaningful. The original factor

TABLE 2
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM PLANNING ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
PLAN1	1	3.94697	23.2	23.2
PLAN2	2	2.94792	17.3	40.6
PLAN3	3	2.14215	12.6	53.2
PLAN4	4	1.59915	9.4	62.6
PLAN5	5	1.35919	8.0	70.6
PLAN6	6	1.16695	6.9	77.4
PLAN7	7	.89620	5.3	82.7
PLAN8	8	.69659	4.1	86.8
PLAN9	9	.61145	3.6	90.4
PLAN10	10	.44935	2.6	93.0
PLAN11	11	.35599	2.1	95.1
PLAN12	12	.26114	1.5	96.7
PLAN13	13	.21328	1.3	97.9
PLAN14	14	.15064	.9	98.8
PLAN15	15	.10343	.6	99.4
PLAN16	16	.07582	.4	99.9
PLAN17	17	.02378	.1	100.0

matrix was rotated orthognally to minimize crossloadings. The three factors explained 53.2% of the total variance. The factors, the items that loaded on these factors, and the loading of the individual items are given in Table 3.

According to this solution, items 3, 4, 5, 10, 15, and 16 loaded on Factor 1. Items 6 to 9 loaded on Factor 2. Items 1, 2, 11, 13, 14, and 17, loaded on Factor 3. Item 12 loaded poorly on all the three factors and was therefore dropped from the analysis. Its highest loading was .38. Based on item content, the three factors were named as High level planning, design level planning, and Implementation level planning.

The item loading on factor 1 ranged from .45 to .89. Item 16 had the lowest loading while item 5 had the highest loading. Item 4 also loaded high (.83). The remaining items loaded in .50's. On factor 2, items 6 to 8 loaded in .70's. The range of item loading was .63 to .77. Item 9 had the lowest loading. Item loadings on factor 3 ranged from .50 to .70. The lowest loading was .50 on item 11. Item 17 loaded the highest.

System Characteristics

The factor analysis of items on system characteristics found four significant factors that met Kaiser's criterion of eigenvalue greater than 1.0 (Kaiser, 1960). However,

Table 3

Factors, Items Loaded on Factors, and Item Loadings on System Planning Category

FACTOR 1 High Level Planning		FACTOR 2 Design Level Planning		FACTOR 3 Implementation Level Planning	
Items	Loading	Items	Loading	Items	Loading
3	.58132	6	.71252	1	.65085
4	.83292	7	.76632	2	.67906
5	.88704	8	.75154	11	.50273
10	.50264	9	.63042	13	.57553
15	.56487			14	.55251
16	.44937			17	.70456

three factor solution was found more appropriate and meaningful. The three factors combined, explained 63.1% of the total variance. The fourth factor contributed 8.7% to the total variance. The factors, their eigenvalues, and the percent of variance as extracted by principal component solution are given in Table 4.

The analysis of the rotated factor matrix revealed that items 6, 8, 9, 12, and 13, loaded on Factor 1; items 2, 3, 4, 5, 7, and 14, loaded on factor 2; and items 1, 10, 11, and 15 loaded on Factor 3. The items that loaded on Factor 1 were related to system characteristics that dealt with maintenance activity. The factor was therefore named as Maintenance. Factor 3 included items that related to the correctness of the output and was named Correctness. The second factor included characteristics other than those related to Correctness or Maintenance, and was therefore named Others. The factors, the items comprising those factors, and their loadings are given in Table 5.

Item loadings on factor 1 range from .64 to .86. The lowest loading was on item 12. Itme 8 loaded the highest. The range of item loading on factor 2 was .53 to .78. Item 2 loaded the highest while item 7 had the lowest loading. On factor 3, items 1 and 10 loaded in .50's while items 11 and 15 loaded in .80's. The range of loading was .57 to .898. Items 10 and 15 had the lowest and the highest loadings, respectively.

TABLE 4
FACTORS. EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM CHARACTERISTICS ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
CHAR1	1	3.75633	25.0	25.0
CHAR2	2	3.71272	24.8	49.8
CHAR3	3	1.99046	13.3	63.1
CHAR4	4	1.31103	8.7	71.8
CHAR5	5	.98471	6.6	78.4
CHAR6	6	.71253	4.8	83.1
CHAR7	7	.64425	4.3	87.4
CHAR8	8	.55377	3.7	91.1
CHAR9	9	.37971	2.5	93.6
CHAR10	10	.27914	1.9	95.5
CHAR11	11	.20451	1.4	96.9
CHAR12	12	.18207	1.2	98.1
CHAR13	13	.13432	.9	99.0
CHAR14	14	.10076	.7	99.6
CHAR15	15	.05369	.4	100.0

Table 5

Factors, Items Loaded on Factors, and Item Loadings on System Characteristics Category

FACTOR 1 Simplicity		FACTOR 2 Others		FACTOR 3 Correctness	
Items	Loading	Items	Loading	Items	Loading
6	.78225	2	.77855	1	.59229
8	.85695	3	.68902	10	.56969
9	.72422	4	.70783	11	.83310
12	.64100	5	.76050	15	.89831
13	.76106	7	.52989		
		14	.64722		

System Design

The principal component analysis of items on system design extracted nine factors with eigenvalue of 1.0 or greater. They explained 35.1% of the total variance. After repeating analysis with different number of factors, the solution with five factors was found more meaningful. The five factor solution explained 60.9% of the total variance. The factors, their eigenvalues, and the proportion of variance explained by each factor are given in Table 6.

The item loadings for the five factor solution after varimax rotation are given in Table 7. Factor 1, labeled a General factor, was comprised of items 2, 10, 14, 21, and 23. The items 8, 17, 18, and 19, loaded on Factor 2 and this factor was named System Environment. Factor 3 was comprised of items 3 to 6 and 20. It was given a name of Programming Considerations. The fourth factor was called Programming Style and included items 1, 9, and 11. Items 12, 13, 15, and 16 loaded on Factor 5 called Efficiency Considerations.

Item 7 did not meet the criterion of minimum loading of .4 and was therefore dropped from further consideration. Item 22 was found inappropriate after the questionnaire was printed. Therefore, it was not included in any analysis. These deletions dropped the total number of items in this section to 22.

TABLE 6
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM DESIGN ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
DESIGN1	1	3.61591	15.7	15.7
DESIGN2	2	3.43662	14.9	30.7
DESIGN3	3	2.89156	12.6	43.2
DESIGN4	4	2.10637	9.2	52.4
DESIGN5	5	1.95081	8.5	60.9
DESIGN6	6	1.65630	7.2	68.1
DESIGN7	7	1.59642	6.9	75.0
DESIGN8	8	1.25301	5.4	80.5
DESIGN9	9	1.07044	4.7	85.1
DESIGN10	10	.77270	3.4	88.5
DESIGN11	11	.67533	2.9	91.4
DESIGN12	12	.51980	2.3	93.7
DESIGN13	13	.40201	1.7	95.4
DESIGN14	14	.27990	1.2	96.6
DESIGN15	15	.20422	.9	97.5
DESIGN16	16	.17545	.8	98.3
DESIGN17	17	.11714	.5	98.8
DESIGN18	18	.10450	.5	99.3
DESIGN19	19	.08481	.4	99.6
DESIGN20	20	.04894	.2	99.8
DESIGN21	21	.03216	.1	100.0
DESIGN22	22	.00454	.0	100.0
DESIGN23	23	.00106	.0	100.0

Table 7

Factors, Items Loaded on Factors, and Item Loadings on System Design Category

FACTOR 1 General Factor		FACTOR 2 System Environment		FACTOR 3 Program Consider-ations		FACTOR 4 Programming Style		FACTOR 5 Efficiency Consider-ations	
Items	Loading	Items	Loading	Items	Loading	Items	Loading	Items	Loading
2	.74770	8	.60084	3	.59183	1	.73281	12	.64593
10	.61624	17	.83495	4	.65513	9	-.60385	13	.66006
14	-.60566	18	.78685	5	.75944	11	.71115	15	.74960
21	.69443	19	.74569	6	.66780			16	.62822
22	.71095			20	.48605				
23	.58179								

Item loadings on factor 1 ranged from .60 to .75. The lowest loading was on item 14 which also happened to be the only negative value on this factor. The loading on item 2 was the highest. The highest loading on factor 2 was .83 (item 17). The lowest loading was on item 8 (.60). The range of item loading on factor 3 was from .49 to .76. Item 20 had the lowest loading while item 5 loaded the highest. Item 9 on factor 4 was the only negatively loaded item. It also had the lowest loading on this factor. The highest loading (.73) was found for item 1. The range of item loadings on factor 5 was .63 to .75. Item 15 loaded the highest while item 16 loaded the lowest. Items 12 and 13 loaded .64 and .66 respectively.

System Testing

Seven factors having eigenvalue of 1.0 or greater were extracted from the 19 items on Testing. They together accounted for 75.3% of the total variance. The factors, their eigenvalues, and the proportion of variance explained by each factor is given in Table 8.

After repeating the analysis with different number of factors, the solution with four factors was found to be the best. The solution explained 56.5% of the total variance. Varimax rotation could not be performed due to

TABLE 8
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM TESTING
ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
TEST1	1	4.11176	21.6	21.6
TEST2	2	2.67095	14.1	35.7
TEST3	3	2.19451	11.6	47.2
TEST4	4	1.76227	9.3	56.5
TEST5	5	1.27842	6.7	63.3
TEST6	6	1.19170	6.3	69.5
TEST7	7	1.10598	5.8	75.3
TEST8	8	.99465	5.2	80.6
TEST9	9	.85524	4.5	85.1
TEST10	10	.76674	4.0	89.1
TEST11	11	.58512	3.1	92.2
TEST12	12	.41027	2.2	94.4
TEST13	13	.31239	1.6	96.0
TEST14	14	.25821	1.4	97.4
TEST15	15	.21203	1.1	98.5
TEST16	16	.12395	.7	99.1
TEST17	17	.08774	.5	99.6
TEST18	18	.05368	.3	99.9
TEST19	19	.02437	.1	100.0

nonconvergence. The factors and the items that loaded on these factors are given in Table 9.

The items that loaded on Factor 1 included 1, 2, 3, 5, 8, 13, 15, and 18. Based on item contents, the factor was named Test Planning. The second factor was named Testing Techniques and was comprised of items 9, 11, 12, 14, 17, and 19. Items 6, 10, and 16, loaded on Factor 3. The item content suggested Testing as an appropriate name for this factor. Factor 4 was comprised of items 4 and 7 only. This factor stayed in every solution examined and basically loaded all the items that could not be loaded on other factors. Therefore, this factor was given the name Residual.

The item loadings on Factor 1 ranged from .50 to .76. Item 3 had the highest loading while item 13 had the lowest loading. Item 15 was the only item with negative loading (-.66). Items 9, 11, and 19 loaded negatively on Factor 2. Disregarding the sign of loadings, the range was found to be .50 to .75. The lowest loading was on item 17. Item 12 loaded the highest. Items 10 and 16 on Factor 3 had almost the same loadings; .495 and .496, respectively. Item 6 with a loading of .58 was the only negatively loaded item on Factor 3. On Factor 4, item 4 loaded .63. The only other item on this factor was item 4 and had a loading of .63

Table 9

Factors, Items Loaded on Factors, and Item Loadings on System Testing Category

FACTOR 1 Test Planning		FACTOR 2 Testing Technique		FACTOR 3 Testing		FACTOR 4 Residual	
Items	Loading	Items	Loading	Items	Loading	Items	Loading
1	.65402	9	-.61911	6	-.58582	4	.63362
2	.69137	11	-.54400	10	.49491	7	-.58325
3	.75801	12	.75380	16	.49619		
5	.56680	14	.56077				
8	.51267	17	.49590				
13	.49862	19	-.52302				
15	-.66493						
18	.57430						

System Documentation

Principal component analysis of the 16 items assessing program documentation extracted five factors that met Kaiser's criterion of an eigenvalue (Kaiser, 1960) and explained 72.7% of the total variance. The factors extracted, their eigenvalues, and the proportion of variance explained by each factor is given in Table 10.

Factor analysis was repeated several times with varying number of factors until a three factor solution was found. The three factor solution explained 59.5% of the total variance. Varimax rotation provided a better look on factor matrix. The first factor was defined by items 2 to 8, 12, and 16, was named as Documentation Standards. The second factor named Quality and Style included items 10, 11, 13, and 15. Items 1, 9, and 14 loaded on Factor 3. This factor was named Documentation Analysis. The factors and the item loadings on these factors are given in Table 11.

The range of item loadings on Factor 1 was .57 to .86. Item 3 loaded the lowest while item 7 had the highest loading. Items 6 and 8 had factor loadings in the .60's. The remaining items loaded in .70's. The range of item loadings on Factor 2 was .63 to .78. Item 11 had the lowest loading. Item 15 had the highest loading. The highest loading on Factor 3 was found on item 1 (.71). The

TABLE 10
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM DOCUMENTATION
ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
DOCU1	1	5.32211	33.3	33.3
DOCU2	2	2.49793	15.6	48.9
DOCU3	3	1.69724	10.6	59.5
DOCU4	4	1.06802	6.7	66.2
DOCU5	5	1.04301	6.5	72.7
DOCU6	6	.99134	6.2	78.9
DOCU7	7	.88653	5.5	84.4
DOCU8	8	.63145	3.9	88.4
DOCU9	9	.46436	2.9	91.3
DOCU10	10	.43418	2.7	94.0
DOCU11	11	.34424	2.2	96.1
DOCU12	12	.22461	1.4	97.5
DOCU13	13	.13000	1.0	98.5
DOCU14	14	.11284	.7	99.2
DOCU15	15	.08684	.5	99.8
DOCU16	16	.03530	.2	100.0

Table 11

Factors, Items Loaded on Factors, and Item Loadings on System Documentation Category

FACTOR 1 Documentation Standards		FACTOR 2 Quality and Style		FACTOR 3 Documentation Analysis	
Items	Loading	Items	Loading	Items	Loading
2	.75081	10	.74884	1	.71028
3	.57085	11	.63104	9	.62792
4	.75785	13	.74112	14	.48123
5	.78442	15	.77805		
6	.68614				
7	.85776				
8	.63960				
12	.74420				
16	.72493				

lowest loading of .48 was on item 14. Item 9, the only remaining item on this factor, loaded .63.

System Correctness

Principal component analysis performed on 20 items related to program correctness produced seven significant factors that met Kaiser's criterion of eigenvalues (Kaiser, 1960). These factors explained 80.1% of the total variance. The factors, their eigenvalues, and the percent of variance explained by each factor is given in Table 12.

The factor analysis was repeated several times by varying the number of factors to be extracted. The solution with four factors was finally selected for its logical and meaningful structure. This four factor solution explained 60.3% of the total variance. The factors and the items that loaded on these factors are given in Table 13. Item loadings are based on orthognally rotated solution.

Items 7, 9, 10, 17, 19, and 20, loaded on Factor 1. Based on item contents, the factor was named as Programming Style. The second factor named Coding Considerations was comprised of items 5, 12, 13, 14, and 16. Items 1, 3, 4, 6, and 15, constituted Factor 3 which was named Programming Structure. The fourth factor that included items 2, 8, and 18, was named Residual as the items that did not fit to other factors, were loaded on Factor 4.

TABLE 12
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM CORRECTNESS
ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
CORECT1	1	5.16206	25.8	25.8
CORECT2	2	2.60216	13.0	38.8
CORECT3	3	2.36562	11.8	50.6
CORECT4	4	1.92420	9.6	60.3
CORECT5	5	1.54266	7.7	68.0
CORECT6	6	1.37380	6.9	74.9
CORECT7	7	1.04140	5.2	80.1
CORECT8	8	.96188	4.8	84.9
CORECT9	9	.77504	3.9	88.7
CORECT10	10	.62194	3.1	91.9
CORECT11	11	.50412	2.5	94.4
CORECT12	12	.44440	2.2	96.6
CORECT13	13	.27128	1.4	98.0
CORECT14	14	.18606	.9	98.9
CORECT15	15	.06931	.3	99.2
CORECT16	16	.06382	.3	99.5
CORECT17	17	.04067	.2	99.8
CORECT18	18	.03195	.2	99.9
CORECT19	19	.01629	.1	100.0
CORECT20	20	.00133	.0	100.0

Table 13

Factors, Items Loaded on Factors, and Item Loadings on System Correctness Category

FACTOR 1 Coding Considerations		FACTOR 2 Programming Style		FACTOR 3 Program Structure		FACTOR 4 Residual	
Items	Loading	Items	Loading	Items	Loading	Items	Loading
7	.61130	5	-.66383	1	.87715	2	.85286
9	.80668	12	-.41989	3	.67233	8	.46509
10	.90569	13	-.49621	4	.79171	18	.81536
11	.88483	14	.64655	6	-.52551		
17	.56146	16	.86023	15	.50372		
19	.60276						
20	.59991						

Item loadings on Factor 1 ranged from .56 to .90. The items with the lowest and the highest loadings were item 17 and item 10, respectively. Items 9 and 11 loaded in .80's. The remaining items were loaded in .60's. Items 5, 12, and 13 on Factor 2 were loaded negatively. Disregarding the sign, the range of item loadings was .42 to .86. The lowest loading was on item 12 while item 16 had the highest loading. The range of item loadings on Factor 4 was .46 to .85. Items 2 and 8 were found as the highest and lowest loaded items respectively. Item 18 loaded .81 on Factor 4.

System Clarity

Factor analysis was performed on 16 items on program clarity. Six factors met Kaiser's criterion (Kaiser, 1960) and explained 79.4% of the variance. The factors, their respective eigenvalues and the percent of variance explained are given in Table 14.

The factor analysis with three factors was accepted as a more logical and explainable solution. The initial factor matrix was rotated orthognally. The factors, the items that loaded on these factors, and their respective loadings are given in Table 15.

Items 2 to 5, 11, 14, 15, and 16 were loaded on Factor 1. This factor was named as Program Quality. The second factor named Coding Considerations consisted of items 1, 6, 9, and 13. Items 7, 8, 10, and 12 constituted

TABLE 14
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM CLARITY
ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
CLAR1	1	3.92909	24.6	24.6
CLAR2	2	2.78223	17.4	41.9
CLAR3	3	1.77073	11.1	53.0
CLAR4	4	1.66256	10.4	63.4
CLAR5	5	1.35292	8.5	71.9
CLAR6	6	1.20630	7.5	79.4
CLAR7	7	.87960	5.5	84.9
CLAR8	8	.70374	4.4	89.3
CLAR9	9	.52316	3.3	92.6
CLAR10	10	.37365	2.3	94.9
CLAR11	11	.27487	1.7	96.6
CLAR12	12	.18721	1.2	97.8
CLAR13	13	.16085	1.0	98.8
CLAR14	14	.08972	.6	99.4
CLAR15	15	.06692	.4	99.8
CLAR16	16	.03645	.2	100.0

Table 15

Factors, Items Loaded on Factors, and Item Loadings on System Clarity Category

FACTOR 1 Programming Quality		FACTOR 2 Coding Considerations		FACTOR 3 Programming Style	
Items	Loading	Items	Loading	Items	Loading
2	-.58995	1	.75874	7	.73963
3	-.78412	6	.81327	8	.56674
4	.54674	9	.51472	10	.66008
5	.63232	13	.52702	12	.68103
11	.62855				
14	.56825				
15	.74500				
16	.44680				

Factor 3. This factor was named Programming Style based on the content of items defining the factor.

Items 2 and 3 loaded negatively on Factor 1. The lowest loading was on item 16 (.45). Item 3 had the highest loading (.78). On Factor 2, the range of item loadings was .51 to .81. Items 6 and 9 were the highest and lowest loaded items, respectively. Item 1 loaded .76 while itme 13 loaded .53. The range of item loadings on Factor 3 was .56 to .74. Items 7 and 8 were the highest and lowest loaded items, respectively. The remaining items were loaded in .60's.

Programming Style

The principal component solution performed on 20 items of programming style extracted seven factors having eigenvalues greater than 1.0. The factors explained 80.2% of the total variance. Table 16 lists the factors, their eigenvalues and the percent of variance explained by these factors.

After several trials, a four factor solution after varimax rotation was found more logical and explainable than the other solutions and was therefore adopted for this analysis. According to this solution, the four factors explained 60.0% of the total variance. Factor 1 was comprised of items 5 to 10, 12, and 19. The factor was

TABLE 16
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON PROGRAMMING STYLE
ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
STYLE1	1	4.39826	22.0	22.0
STYLE2	2	2.83706	14.2	36.2
STYLE3	3	2.70445	13.5	49.7
STYLE4	4	2.06803	10.3	60.0
STYLE5	5	1.55852	7.8	67.8
STYLE6	6	1.29267	6.5	74.3
STYLE7	7	1.17177	5.9	80.2
STYLE8	8	.89350	4.5	84.6
STYLE9	9	.73479	3.7	88.3
STYLE10	10	.58217	2.9	91.2
STYLE11	11	.51737	2.6	93.8
STYLE12	12	.34431	1.7	95.5
STYLE13	13	.32311	1.6	97.1
STYLE14	14	.20276	1.0	98.1
STYLE15	15	.18492	.9	99.1
STYLE16	16	.11259	.6	99.6
STYLE17	17	.03994	.2	99.8
STYLE18	18	.01695	.1	99.9
STYLE19	19	.01390	.1	100.0
STYLE20	20	.00292	.0	100.0

named Progam Quality. Items 2, 3, 14, and 15, loaded on Factor 2. The factor was called Programming Style. The third factor called Coding Consideration included items 1, 4, 11, 16, and 20. Items 13, 17, 18, produced the fourth factor called Efficiency Considerations.

The factors, the items that loaded on these factors, and their respective loadings after varimax rotation are given in Table 17.

Item 19 loaded slightly higher (.406) than the minimum value of .40, needed to assign that item to a particular factor. This item had the lowest loading on Factor 1. Item 10 had the highest loading (.87). Items 8, 9, and 12 were negatively loaded. Item loadings on this factor were comparatively lower than the other three factors. Item loadings on Factor 2 ranged from .50 to .86. The items with the lowest and highest loading on this factor were items 15 and 2, respectively. The loading range on Factor 3 was .61 to .85. Item 4 loaded the highest while item 11 loaded the lowest. Most of the remaining items had factor loadings in .60's. The range of item loadings on Factor 4 was .75 to .88. Items 13 and 17 were the lowest and highest loaded items, repectively. The only remaining item (item 18) had a loading of .80.

Table 17

Factors, Items Loaded on Factors, and Item Loadings on Programming Style Category

FACTOR 1 Programming Quality		FACTOR 2 Programming Style		FACTOR 3 Coding Considerations		FACTOR 4 Efficiency Considerations	
Items	Loading	Items	Loading	Items	Loading	Items	Loading
5	.61753	2	.86058	1	.70405	13	.74984
6	.58687	3	.83146	4	.84666	17	.87682
7	.55423	14	.68506	11	.60707	18	.80412
8	-.54315	15	.49848	16	.63895		
9	-.41708			20	.67019		
10	.87187						
12	-.48739						
19	.40658						

System Management

Fourteen management related items were analyzed by factor analysis to discover underlying dimensions. The principal component solution revealed six factors that explained 80.0% of the total variance. The factors, their respective eigenvalues and the percent of variance explained by each factor are given in Table 18.

Factor analysis was repeated several times by varying the number of factors to be extracted so that a logical and meaningful solution could be obtained. The solution with four factors after varimax rotation met this criterion and is included here for interpretation. According to this solution, the factors explained 64.4% of the total variance. The factors, items loaded on these factors, and item loadings after varimax rotation are given in Table 19.

Factor 1 was defined by items 2, 4, and 12, and was named Management Structure. Factor 2 included items 5, 8, 9, and 10, and was called Personnel Management. Items 3, 11, 13, and 14, comprised the third factor called Management Support. The fifth factor called Management Strategy consisted of items 1, 6, and 7.

Item 2 loaded the lowest on Factor 1 (.59). The highest loading of .90 was found on item 12. Item 4, the only remaining item on this factor, loaded .65. The range of item loadings on Factor 2 was .60 to .81. The lowest loading was on item 9. Item 8 had the highest loading.

ABLE 18
FACTORS, EIGENVALUES, AND THE VARIANCE EXPLAINED
ON SYSTEM MANAGEMENT ITEMS

VARIABLE	FACTOR	EIGENVALUE	PCT OF VAR	CUM PCT
MANAGE1	1	3.64358	26.0	26.0
MANAGE2	2	1.96359	14.0	40.1
MANAGE3	3	1.88064	13.4	53.5
MANAGE4	4	1.53049	10.9	64.4
MANAGE5	5	1.12760	8.1	72.5
MANAGE6	6	1.05838	7.6	80.0
MANAGE7	7	.80897	5.8	85.8
MANAGE8	8	.62984	4.5	90.3
MANAGE9	9	.39490	2.8	93.1
MANAGE10	10	.32116	2.3	95.4
MANAGE11	11	.25376	1.8	97.2
MANAGE12	12	.21356	1.5	98.8
MANAGE13	13	.11252	.8	99.6
MANAGE14	14	.06102	.4	100.0

Table 19

Factors, Items Loaded on Factors, and Item Loadings on System Management Category

FACTOR 1 Management Structure		FACTOR 2 Personnel Management		FACTOR 3 Management Support		FACTOR 4 Management Strategy	
Items	Loading	Items	Loading	Items	Loading	Items	Loading
2	.59563	5	-.65511	3	.45150	1	.79499
4	.65539	8	.81564	11	.52884	6	.74311
12	.90512	9	.59775	13	.88239	7	.55144
		10	.64892	14	-.75144		

Item 5 was the only negatively loaded item on this factor. Item 14 on Factor 3 loaded negatively. The lowest and highest loading were on item 3 and item 13, respectively. The range of item loadings was .45 to .88. Item loadings on Factor 4 ranged .55 to .79. Items 1 and 7 were the lowest and highest loaded items, respectively. Item 6 loaded .74 on this factor.

The factor matrices on all the nine categories are given in Appendix B.

Chapter 5

DISCUSSION AND CONCLUSION

This chapter includes discussion of the results described in Chapter 4. The chapter closes with the conclusion derived from the findings of this research study.

The analysis of item means and standard deviations confirmed that all the items included in the questionnaire affect system complexity to varying degrees. Items with a mean value of less than 3.0 reflected respondents' agreement that the attributes expressed in the statements decrease software complexity. Similarly, item means of greater than 3.0 reflected respondents' opinions that such attributes increase system complexity. The items with a mean value of 3.0 or closer were not considered good discriminators. These items were PLAN 15, DESIGN 12, DESIGN 22 and STYLE 20. The content analysis revealed that all these items except DESIGN 22 were highly related to system complexity but respondents responded to them randomly and the mean value turned out to be 3.0.

An examination of standard deviation values revealed that not all respondents agreed or disagreed with the statements with equal strength. Standard deviation of 1.0 or greater was viewed as high for this analysis. Standard deviation of less than 1.0 was an index of the conformity

of views of respondents regarding those statements. Large standard deviation values were considered to be a result of different interpretations, that respondents attached to standard terms. This fact confirmed the confusion that exists in computer science literature for non-standard definitions and use of more than one term for a single phenomenon.

Factor analysis revealed that the nine logical categories to which the questionnaire was divided were not exclusive. Factor analysis of items of a single category, sometimes, produced factors that were related to the major category. For example, Factor 3 of system characteristics was named correctness which in fact is one of the nine categories. Similarly, Factor 4 of System Design, Factor 2 of Program Correctness, and Factor 3 of Program Clarity discovered underlying dimensions that were supposedly tapped by one of the main categories. This finding confirmed the rationale of this study that a single factor in a system may affect several system components making a cumulative effect on the complexity of a total system. It meant that a system cannot be categorized into non-exclusive components. For example, system testing which is a component of system development may not stand alone because it includes planning for testing, characteristics of testing, testing documentation, testing management, testing accuracy, clarity of test procedures, and

programming style in test code. In other words, each component of a system encompasses the whole developmental span of the system. This finding was important for future research activity in which the factors obtained from this study may be used as logical entities rather than making judgmental categories as was done in this exploratory study.

Factor analysis also revealed that a major category may give rise to factors that identify major components of that category. The example is of System Planning category in which all the three factors obtained refer to system planning but at a different level. Factor 1 referred to high level planning. Factor 2 clustered items that related to design level planning. The third factor identified implementation level planning or lower level planning. Similarly, the system testing category produced four factors called test planning, testing technique, testing, and residual factor. Another example was the system management category in which the four factors identified various components of management. Factor 1 was called management structure, Factor 2 related to personnel management, Factor 3 was management support, and Factor 4 referred to management strategy.

On program documentation category, the three factors identified three dimensions called documentation standards, quality and style of documentation, and documentation

analysis. Under the programming style category, the four dimensions that were discovered by factor analysis were named programming quality, programming style, coding considerations, and efficiency considerations.

On certain categories, factor analysis gave rise to factors that could not successfully be named. The examples included system characteristics category in which Factor 2 was named "Others". It was, in fact, a residual factor such that the items that did not load on other factors were loaded on Factor 2. For the program correctness category, Factor 4 was a residual factor. The items that did not load on the coding consideration factor, programming style factor, or programming structure factor, loaded on Factor 4 called residual factor.

The cross-loading of items, though minimized by varimax rotation was still greater than expected. This confirms the earlier finding made on the basis of item means and standard deviations that the use of several terms for the same phenomenon means different interpretations. The other possible reason of cross-loadings was that people were not sure what they were responding to and therefore selected the degree of agreement or disagreement with the statement at random. This tendency might have been resulted from lack of education or experience by respondents in the respective areas. Most of the subjects of this study were involved in the process of large system

development but not all of them had extensive formal training in system design and development. Those who learned through experience were not familiar with the terminology used in the literature. The third category was those who had taken at least one course in system design and development but did not have experience of designing the system. The perceptions of these three distinct groups produced three unique sets of responses such that their cumulative effect resulted in cross-loading of items on more than one factor.

Conclusion

One hundred and sixty items extracted from the existing literature were categorized into nine categories according to their content. These categories were made on subjective reasons and were system planning, system characteristics, system design, system testing, program documentation, program correctness, program clarity, programming style, and system management.

Item means indicated that at least 98% of the items do affect system complexity. Content analysis indicated that all the items except one are related to system complexity. High variance in respondents' responses was attributed either to their lack of formal training or experience in some areas of system development, or to the confusion

caused by multiple definitions of a single term. Computer science literature has an abundance of terms that have multiple meanings and all of them are recognized as legitimate.

Factor analysis on system planning category identified three factors called high level planning, design level planning and implementation level planning. The system characteristics category produced two factors called simplicity and correctness. The third factor was a general factor. Under system design category, the factors were identified as general factor, system enviornment, program considerations, programming style, and efficiency considerations. Test planning, testing technique, testing, and residual were the factors identified in System testing category. Program documentation category had three underlying dimensions called documentation standards, documentation quality & style, and documentation analysis. Program correctness had coding considerations, programming style, program structure, and residual as factors. Programming quality, coding considerations, and programming style were the dimensions of program clarity category. Programming style category identified programming quality, programming style, coding considerations and efficiency considerations as factors. The last category, system management, had management structure, personnel management, management support, and management stragtegy as factors.

Factor analysis suggested that the items may be regrouped according to the factors extracted to determine complexity of system attributes, system components, or of various phases of system development. It was recommended that the study be replicated with a larger sample and that the complexity of various system components be weighted to determine the overall complexity of the system.

REFERENCES

Al-Suwaiyel, M.I. Man and Software Complexity. Cybernetica, 26,3, 1983, 227-235.

Arthur, Lowell J. Programmer Productivity: Myths, Methods, and Murphology. John Wiley & Sons, N.Y., 1983, pp. 287.

Baker, A.I., and Zweben, S.H. A comparison of measures of control flow complexity. IEEE Transactions on Software Engineering, SE-6, 6, 1980, 506-512.

Belady, B.L.A. Software geometry. In proceedings of 1980 Computer Symposium, Taipei, Republic of China, 1980.

Bersof, E.H., Handerson, V.D., and Siegel, S.G. Software Configuration Management: An investment in product integrity. Prentice Hall Inc., N.J. 1980, pp 385.

Brotsky, D. Program Understanding Through Cliche Recognition. Working Paper 224, AI Lab., MIT, 1981.

Canning, R.G. Modular COBOL programming. EDP Analyzer, 10, 7, 1972, 1-14.

Curtis, B., Sheppard, S.B., Millman, P., Borst, M.A., and Love, T. Measuring the psychological complexity of software maintenance tasks with Halstead and McCabe metrics. IEEE Transactions on Software Engineering, SE-5, 2, 1979, 96-104.

Davis, John S. Chunks: A basis for complexity measurement. Information Processing & Management, 20, 1-2, 1984, 119-127.

De Millo, R.A., Lipton, R.J., and Perlis, A. Social processes and proofs of theorams and programs. Communications of the ACM, 22, 5, 1979, 271-280.

Dun, Robert, and Ullman, Richard. Quality Assurance for Computer Software. McGraw-Hill book Company, N.Y., 1982, pp.351.

Endres, A. An analysis of errors and their causes in systems programs. IEEE Transactions on Software Engineering, SE-1, 2, 1975, 140-149

Halstead, M. Elements of software science. Elsevier Computer Science Library, N.Y., 1977.

Henry, S., Kafura, K., and Harris, K. On the relationship between three software metrics. Proceedings of the ACM Workshop/Symposium. Software Quality, University of Maryland, college Park, 1981.

Jensen, H. An investigation of software metrics for real-time software. Unpublished master's thesis, University of Wisconsin - Milwaukee, 1982.

Jensen, H.A., and Vairavan, K. An experimental study of software metrics for real-time software. IEEE Transactions on Software Engineering, SE-11, 2, 1985, 231-234.

Kaiser, H.F. The application of electronic computers to factor analysis. Educational Psychological Measurement, 20, 1960, 141-151

Kernighan, B.W., and Planger, P.J. The Elements of Programming Style. McGraw Hill Book Company, N.Y., 1974, pp. 147.

Lientz, B.P., and Swanson, E.B. Problems in application software maintenance. Communications of the ACM. 24, 11, 1981, 763-769.

Lientz, B.P., Swanson, E.B., and Tompkins, G.E. Characteristics of application software maintenance. Communications of the ACM, 21, 6, 1978, 466-471.

London, Keith R. Documentaion Standards (Revised Ed.). Petrocelli Books, N.Y., 1974, pp. 253.

Mayer, R.E. A psychology of learning basic. Communications of the ACM, 22, 11, 1979, 589-593.

McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, SE-2, 4, 1976, 308-320.

Meek, Brian, and Heath, Patricia. Guide to Good Programming Practice. Ellis Horwood Ltd., N.Y., 1981, pp. 181.

Norcio, A. Human Memory Processes for Comprehending Computer Programs. Applied Science Department, U.S. Naval Academy, 1980.

Ottenstein, L.M. Quantitative estimates of debugging requirements. IEEE Transactions on Software Engineering, SE-5, 1979.

Pashtan, Ariel. Operating system models in a concurrent Pascal environment: Complexity and performance considerations. IEEE Transactions on Software Engineering, SE-11, 1, 1985, 136-141.

Peter, L.J. Software Design: Methods & Techniques. Yourdon Press, N.Y., 1981.

Prather, Ronald E. An axiomatic theory of software complexity measure. The Computer Journal, 27, 4, 1984, 340-347.

Savage, J.E. The Complexity of Computing. John Wiley & Sons, N.Y., 1976.

Schneider, V. Some experimental estimators for developmental and delivered errors in software development projects. In Proceedings of the ACM Workshop/Symposium. Software Quality, University of Maryland, College Park, 1981.

Shneiderman, B. Measuring computer program quality and comprehension. International Journal of Man-Machine Studies, 9, 1976, 465-478.

Shneiderman, B. Exploratory experiments in programmer behavior. International Journal of CICS, 5, 2, 1976b, 122-143.

Shen, V.Y., Conte, S.D., and Dunsmore, H.E. Software science revisited: A critical analysis of the theory and its empirical support. IEEE Transactions on Software Engineering, SE-9, 1983, 155-165.

Soloway, E. What do novices know about programming? Directions in Human-Computer Interactions, Ablex, 1982.

SPSS Inc. SPSS'X: User's Guide. McGraw Hill Book Company, N.Y., 1983.

Sunohara, T., Takano, A., Vehara, K., and Ohkawa, T. Program complexity measure for software devleopment management. 5th International Conference on Software Engineering, IEEE, N.Y., 1981.

Tassel, Dennie V. Program Style, Design, Efficiency, Debugging, and Testing. Prentice-Hall Inc., 1978, pp 323.

Thayer, T.A., Lipow, M., and Nelson, E.C. Software Reliability. North-Holland, Amsterdam, 1978.

Vassey, Iris, and Weber, Ron. Some factors affecting program repair maintenance: An empirical study. Communununications of the ACM, 26, 2, 1983, 128-134.

Weinberg, G.M. The Psychology of Computer Programming. Van Nostrand Reinhold, N.Y., 1971.

Woodfield, S.N. Enhanced effort estimation by extending basic programming models to include modularity factors. Doctoral dissertation, Purdue University, 1980.

BEST COPY AVAILABLE

The University of Kansas

Institute for Research in Learning Disabilities
Emphasis on Adolescents and Young Adults

Carruth-O'Leary Hall
Room 208
Lawrence, Kansas 66045-2342
(913) 864-4720

Dear Computer Scientists,

There are several factors that are considered very predictive of software complexity. A program which is structured,well documented, and has simple to understand code is believed to have less repair maintenance after its release into production than the programs that do not incorporate these factors. A long list of factors that make later modifications of programs easy, is available from the literature. However, there is no empirical evidence as to what factors are more important than the others. The present study is a step forward to identify the most significant factors that reduce system complexity and to discover their underlying dimensions.

The terms system complexity and software complexity are used interchangably in the enclosed questionnaire and both refer to large programs, only. For this study, the definition of complexity is also delimited to an amount of effort needed to add, delete, or modify segment(s) of a program. For example, a program that takes less effort is less complex than the one that needs more effort.

Your name has been selected for partcipation in this study as a representative of the profession that deals with the development and maintenance of computer software. Your responses on the enclosed questionnaire will provide us valuable information about factors that could reduce system complexity.

Your responses on the questionnaire will be kept confidential and a copy of the results will be sent to you, if desired. If you have any question about the questionnaire, or on any other part of the study, please feel free to contact me at (913) 864-4780.

Thankyou for your participation and completing the questionnaire.

Sincerely,

Kaiser

Javaid Kaiser, Ph.D.

System planning(top level design) is the most crucial part of system development. Thorough understanding of the proposed system not only helps in its development, but also reduces system complexity. Indicate the degree to which you agree or disagree that the following considerations at the planning stage would reduce the complexity of the system. (Complexity, for this study, is defined in terms of repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

System design specifications	1	2	3	4	5
Software requirement specifications	1	2	3	4	5
Performance Specifications	1	2	3	4	5
Product specifications	1	2	3	4	5
Project rationale	1	2	3	4	5
Top level design review	1	2	3	4	5
Module design review	1	2	3	4	5
Data base design	1	2	3	4	5
Integration test plan	1	2	3	4	5
Selection of test procedures	1	2	3	4	5
Configuration management	1	2	3	4	5
Documentation standards	1	2	3	4	5
Quality control plan	1	2	3	4	5

71

Tool specifications	1	2	3	4	5
Vendor survey and survillance	1	2	3	4	5
Knowledge about future development plans	1	2	3	4	5
Future maintenance activity	1	2	3	4	5

The following characteristics represent a system with less complexity. Indicate the degree to which you agree or disagree that the named characteristic would also reduce system complexity. (Complexity is defined as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you neither AGREE or DISAGREE with the statement.

Circle 4 if you DISAREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Correctness of output	1	2	3	4	5
Efficiency(Minimized processing time)	1	2	3	4	5
Flexibility to make enhancements	1	2	3	4	5
Integrity(How well the software and data are protected	1	2	3	4	5
Interopersbility(Interface with other systems)	1	2	3	4	5
Maintainability(Activity to locate or repair errors).	1	2	3	4	5
Portsbility(Change in machine environment)	1	2	3	4	5
Reliability(Degree to which a system is required to perform its functions).	1	2	3	4	5
Usability(Effort to learn, operate, and use the system).	1	2	3	4	5
Testability(Structured testing to insure correctness)	1	2	3	4	5
Tracesbility(Machine operated measurement for correctness).	1	2	3	4	5
Simplicity(Implementation of functions in most understandable way).	1	2	3	4	5
Modularity(Independent functions linked together).	1	2	3	4	5
Concision(Implement a function with minimum code).	1	2	3	4	5
Structured programming(Use of IF-THEN-ELSE etc.)	1	2	3	4	5

Several factors need to be considered at the detailed design stage of system development to reduce software complexity. Indicate the degree to which you agree or disagree that the named factor would reduce the complexity of the system. (Complexity is defined in terms of repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

BEST COPY AVAILABLE

High decision density(# of decisions in a module).	1	2	3	4	5
High program level(# of CALLs to functions per 100 lines of code).	1	2	3	4	5
Choice of procedures for formal corrective action	1	2	3	4	5
Resolution of hardware and software interface	1	2	3	4	5
Well defined data structure	1	2	3	4	5
Establishing control over batches of input	1	2	3	4	5
Use of methods for isolating errors and their causes	1	2	3	4	5
Control on multiprogramming	1	2	3	4	5
Use of generality in program design	1	2	3	4	5
Enhancement to a system	1	2	3	4	5
Formal evaluation of algorithm accuracy	1	2	3	4	5
Differential comparison of programs	1	2	3	4	5
Use of psudocode to document module logic	1	2	3	4	5
Use of graphical tools to display software logic	1	2	3	4	5
Use of cross references in code	1	2	3	4	5
Including modules with multiple entry/exists	1	2	3	4	5
Choice of a programming language	1	2	3	4	5
Adequacy of operational environmenets	1	2	3	4	5
Top-down programming	1	2	3	4	5
State of the art hardware for program development	1	2	3	4	5
Poor distinction between hardware and software functions	1	2	3	4	5
Initial system state not considered	1	2	3	4	5
Poor user training	1	2	3	4	5

Testing is a part of system development. The decisions made during the testing stage may affect the system complexity. Indicate the degree to which you agree or disagree that the named characteristic would reduce system complexity. (Complexity is defined as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Choice of test procedures	1	2	3	4	5
Choice of test equipment selected	1	2	3	4	5
Program test and operating instructions	1	2	3	4	5
Appropriateness of tests of reasonability for I/O validation	1	2	3	4	5
Establishing tolerance for accuracy criterion	1	2	3	4	5
Top down testing	1	2	3	4	5
Testing and maintenance history	1	2	3	4	5
Quality of test programming	1	2	3	4	5
Run time analysis	1	2	3	4	5
Retesting of all modules on new data that interact with modified module	1	2	3	4	5
Bottom-up testing	1	2	3	4	5
Testing a big program in small pieces	1	2	3	4	5
Testing program at boundary values	1	2	3	4	5
Checking answers by hand	1	2	3	4	5

BEST COPY AVAILABLE

Lack of exhaustive testing	1	2	3	4	5
Using a simple version to test the basic design	1	2	3	4	5
Using test data for each path	1	2	3	4	5
Adequate time for testing	1	2	3	4	5
Each test representing a differenting class	1	2	3	4	5

Program documentation i important in understanding the system logic. Indicate the degree to which you agree or disagree that the named documentation characteristic would reduce system complexity. (Complexity is defined in terms of repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Inadequate description of data environment	1	2	3	4	5
High self documentation value(# of comment lines per 100 lines of code).	1	2	3	4	5
Documentation for individual in tallation	1	2	3	4	5
Development of operator and maintenance manuals	1	2	3	4	5
Documentation of input, output, and files handled by the system	1	2	3	4	5
Information on specisl diagnostic codes and flags	1	2	3	4	5
Documenting of complex logic, when used	1	2	3	4	5

Documentation on interrupt processing	1	2	3	4	5
Static analysis of documentation and source code	1	2	3	4	5
Quality of written documents	1	2	3	4	5
Documentation of data layouts	1	2	3	4	5
Agreement between comments and code	1	2	3	4	5
Uniformity of style and appearance	1	2	3	4	5
Use of more comments than needed	1	2	3	4	5
Indenting comments and source code the same amount	1	2	3	4	5
Documentation should start at design stage	1	2	3	4	5

Correctness is an important characteristic of system development. Several decisions that are made to make the system function correctly also affect its complexity. You are asked to indicate the degree to which you agree or disagree that implementing the named characteristic for correctness would also reduce software complexity. (Complexity is interprated as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Control structure to process priorities	1	2	3	4	5
Conformity with data base rules	1	2	3	4	5

BEST COPY AVAILABLE

Clarity in addressing scheme	1	2	3	4	5
Proper use of registers	1	2	3	4	5
Patching bad code instead of rewriting it	1	2	3	4	5
Use of recursive procedures for recursively defined data structures	1	2	3	4	5
Terminating input by end-of-file or marker, not by count	1	2	3	4	5
Identify bad output and recover when possible	1	2	3	4	5
Initialize varisbles and constants before use	1	2	3	4	5
Avoid off-by-one error	1	2	3	4	5
Branch the right way on equality	1	2	3	4	5
Arithmatic with floating numbers	1	2	3	4	5
Comparison of floating point numbers for equality	1	2	3	4	5
Making code right before making it faster	1	2	3	4	5
Assure the correctness of solution at the design stage	1	2	3	4	5
Relisbility is important than efficiency	1	2	3	4	5
Initislizing varisbles with executable code	1	2	3	4	5
Use of mixed data types	1	2	3	4	5
Use of debugging compiler	1	2	3	4	5
Introducing debugging aids early	1	2	3	4	5

The level of clarity maintained during system development, to a grester extent, determines the esse in maintenance, after the system goes into production. Indicate the degree to which you agree or disagree that the

following statements about clarity would also reduce system complexity. (Complexity is defined as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Sequence of source code	1	2	3	4	5
High GOTO density(# of GOTO statements per 100 lines of code)	1	2	3	4	5
Sacrifice clarity for efficiency	1	2	3	4	5
Transform hard logical expression to simple ones	1	2	3	4	5
Using mesningful statement labels	1	2	3	4	5
Use of uniform input format	1	2	3	4	5
Using free-form input when possible	1	2	3	4	5
Using blank spaces in source code	1	2	3	4	5
Selecting mnemonic names that won't be confused	1	2	3	4	5
Use of prefix or suffix on file names	1	2	3	4	5
Using single statement per line	1	2	3	4	5
Alphabetizing lists including arguments, Parameters, and declarations	1	2	3	4	5
Use of parenthesis to avoid ambiguity	1	2	3	4	5
Indentation to show program structure	1	2	3	4	5
Making code simple to understand	1	2	3	4	5
Use of prefered varisble type for subscripts	1	2	3	4	5

BEST COPY AVAILABLE

Programming style elements enhance clarity and help in understanding software logic better. Indicate the degree to which you agree or disagree that the following set of statements would also reduce software complexity. (Complexity is interpreted as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Keeping module size small (not to excede 100 executable statements)	1	2	3	4	5
Use of temporary variables	1	2	3	4	5
Replacing repetetive tasks by CALLs to functions	1	2	3	4	5
Avoiding FORTRAN arithmatic IF	1	2	3	4	5
Use of unnecessary branches	1	2	3	4	5
Use of conditional branches as a substitute for logical expression	1	2	3	4	5
Use of data arrays to avoid repetetive control sequence	1	2	3	4	5
Choice of data representation that makes the program simple	1	2	3	4	5
Making output self explanatory	1	2	3	4	5
Strain to reuse code instead of rearranging it	1	2	3	4	5
Making special cases truly special	1	2	3	4	5
Don't diddle code to make it faster, find a better algorithm	1	2	3	4	5
Use of variables not constants for parameters	1	2	3	4	5
Use of library routines and functions when available	1	2	3	4	5
Plan ahead for program changes	1	2	3	4	5
Use of compiler for simple optimization	1	2	3	4	5
Block I/O efficiently	1	2	3	4	5
Use of load modules for repeated runs	1	2	3	4	5
Use of large number of NOT conditional clauses	1	2	3	4	5
Use of several EJECT and SKIP statements	1	2	3	4	5

Management plays an important role in system development and may also affect system complexity. To what degree do you agree or disagree that the following set of management related activities would reduce software complexity. (Complexity is defined as repair maintenance)

Circle 1 if you STRONGLY AGREE with the statement.

Circle 2 if you AGREE with the statement.

Circle 3 if you NEITHER AGREE OR DISAGREE with the statement.

Circle 4 if you DISAGREE with the statement.

Circle 5 if you STRONGLY DISAGREE with the statement.

Management by objectives	1	2	3	4	5
Phased methodology to develop system	1	2	3	4	5
Favorable management environments	1	2	3	4	5
Fixed schedule to complete work	1	2	3	4	5

Homogeneous group of system developer	1	2	3	4	5
Team concept of system development	1	2	3	4	5
Egoless programming	1	2	3	4	5
Programmer's motivation for task	1	2	3	4	5
Inclusion of development staff in the testing team	1	2	3	4	5
Vertical and horizontal intersction of programmers	1	2	3	4	5
Well budgeting of the system	1	2	3	4	5
A heirarchical organization of programmers	1	2	3	4	5
Small design teams	1	2	3	4	5
Differences over interpretation between project manager and general management	1	2	3	4	5

The questions below ask you responses to enable us to see the differences in opinions expressed by groups representing various levels of sex, education, experience, and profession.

CIRCLE the response that describes you the best.

SEX: 1. MALE 2. FEMALE

EDUCATION:

1. BS in computer science
2. MS in computer science
3. Ph.D. in computer science
4. Computer coursework but no degree in Comp. Sc.
5. No formal education in computer science

TYPE OF EMPLOYMENT:

1. Regular job
2. Student monthly
3. Student hourly

NATURE OF JOB:

1. Related to software development/maintenance
2. NOT related to software development/maintenance
3. Any other, explain: _____

WORK DESIGNATION: _____

COMPUTER RELATED EXPERIENCE:

1. Less than 1 year
2. 1-2 years
3. 2-4 years
4. More than 4 years

Do you want a copy of results? 1. YES 2. NO

BEST COPY AVAILABLE

BEST COPY AVAILABLE

FACTOR MATRICES

System Planning

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3
PLAN1	.29764	.03363	.65085
PLAN2	-.15108	-.15660	.67906
PLAN3	.58132	-.33728	.22070
PLAN4	.83292	-.07077	.03433
PLAN5	.88704	-.04583	.02189
PLAN6	-.24696	.71252	-.21493
PLAN7	-.26897	.76632	.18352
PLAN8	.39209	.75154	.16460
PLAN9	.14219	.63042	-.11769
PLAN10	.50264	.44601	.10766
PLAN11	.35332	.47113	.50273
PLAN12	.05230	.21101	.36494
PLAN13	-.17490	.53526	.57553
PLAN14	.31663	-.08639	.55251
PLAN15	.56487	.19333	-.08636
PLAN16	.44937	.03062	.38679
PLAN17	-.07418	-.04887	.70456

System Characteristics

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3
CHAR1	-.00347	.34353	.59229
CHAR2	-.31096	.77855	.10699
CHAR3	.33121	.66902	-.08656
CHAR4	.26893	.70783	.09737
CHAR5	-.05363	.76050	.12641
CHAR6	.78225	.10112	.17049
CHAR7	.22812	.52989	.52763
CHAR8	.85695	.08343	-.07460
CHAR9	.72422	.04421	.11288
CHAR10	.44862	-.34976	.56969
CHAR11	.15897	-.06590	.83310
CHAR12	.64100	-.44283	.03491
CHAR13	.76106	-.04629	.02483
CHAR14	-.32250	.64722	.09411
CHAR15	-.14548	.18268	.89831

System Design

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3	FACTOR 4	FACTOR 5
DESIGN1	-.43282	-.03483	-.01116	.73281	-.11862
DESIGN2	.74770	.08623	.16588	-.11820	-.14066
DESIGN3	.20638	-.41677	.59183	.02187	-.31205
DESIGN4	.13069	-.47354	.65513	-.06930	-.00866
DESIGN5	-.03042	.24867	.75944	-.14114	-.02405
DESIGN6	-.28076	.06111	.66708	-.10501	.57445
DESIGN7	-.31649	.02159	.19085	-.21941	-.02941
DESIGN8	.19998	.60084	-.01660	-.23946	.17449
DESIGN9	.17915	-.28404	-.11230	.60385	.40984
DESIGN10	.61624	.16375	-.20097	-.15111	-.04332
DESIGN11	-.00776	-.01619	.21633	-.71115	.15051
DESIGN12	.32129	.41374	.21801	-.04258	.64593
DESIGN13	-.01740	.43001	-.17711	.34787	-.66006
DESIGN14	-.60566	.17235	-.14248	.05011	.11491
DESIGN15	-.13253	.03710	.05232	.18171	.74960
DESIGN16	-.10245	.17425	-.22110	-.44542	.62822
DESIGN17	.04388	.83495	-.06722	-.17229	.02879
DESIGN18	.03515	.76685	.07860	-.02580	-.05862
DESIGN19	-.07164	.74569	.04731	.22932	.01574
DESIGN20	-.37246	.13822	.48605	-.02191	.16794
DESIGN21	.69443	.03139	-.17744	-.10285	.18113
DESIGN22	.71095	.06567	-.23429	.40169	-.05159
DESIGN23	.58179	.09904	.22461	.47073	.11206

BEST COPY AVAILABLE

System Testing

FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3	FACTOR 4
TEST1	.65402	.24662	-.07002	.30880
TEST2	.69157	.15059	-.50114	.07755
TEST3	.75801	.06604	-.13370	.16354
TEST4	.37866	-.13508	.24976	.63362
TEST5	.56380	-.20941	-.14248	-.07292
TEST6	.41506	.05105	-.58582	-.13921
TEST7	.50253	-.41655	.19919	-.58325
TEST8	.51267	-.27148	.27751	.02965
TEST9	.24018	-.61911	.42928	.17955
TEST10	.48900	-.02090	.49491	-.07510
TEST11	.24646	-.54400	.30864	-.28406
TEST12	.01356	.75380	.34504	-.27303
TEST13	.49862	.31395	.36897	.21033
TEST14	.21568	.56077	.38014	.26044
TEST15	-.66493	.00126	.40870	.41924
TEST16	-.01553	.09268	.49619	-.43561
TEST17	.27532	.49590	.18068	-.11728
TEST18	.57430	.28668	.08626	-.38224
TEST19	.37301	-.52302	.02338	.19069

System Documentation

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3
DOCU1	.16579	-.06285	-.71028
DOCU2	.75081	-.16217	-.16901
DOCU3	.57085	.00693	-.01676
DOCU4	.75785	.08008	.51574
DOCU5	.78442	.30342	.00741
DOCU6	.68614	.33187	-.18967
DOCU7	.85776	.06177	-.17654
DOCU8	.63930	.16208	-.23727
DOCU9	.05428	.51812	-.62792
DOCU10	.28191	.74884	.22357
DOCU11	.28680	.63104	-.18537
DOCU12	.74420	-.01992	.14898
DOCU13	.05643	.74112	-.05021
DOCU14	-.13553	.36144	.48123
DOCU15	-.34241	.77805	.02702
DOCU16	.72493	-.08370	.35595

System Correctness

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3	FACTOR 4
CORECT1	.00945	-.26536	.87715	.11700
CORECT2	.12778	-.01932	.30808	.85266
CORECT3	.29996	-.36331	.67235	-.19507
CORECT4	.44412	.20626	.79171	.05938
CORECT5	.31572	.66383	.06618	.20885
CORECT6	.27935	-.30270	-.52551	-.06425
CORECT7	.61130	.18329	-.09592	-.09847
CORECT8	.41050	-.05453	.05814	.46509
CORECT9	.80668	.03476	-.00669	.16437
CORECT10	.90569	.06539	.19019	-.05224
CORECT11	.68483	.00669	.17624	.22241
CORECT12	.18614	-.41989	.16192	-.37428
CORECT13	-.00995	-.49621	.14408	.13608
CORECT14	.27380	.64655	.20442	.18444
CORECT15	.01654	.36516	.50372	-.28440
CORECT16	.18768	.86023	.09065	-.09370
CORECT17	.56146	.21238	.10677	-.03526
CORECT18	.02717	.06064	-.22859	.81536
CORECT19	.60276	-.00824	-.11505	.34079
CORECT20	.59991	.43457	.14648	-.04995

BEST COPY AVAILABLE

System Clarity

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3
CLAR1	.05136	-.75874	-.36846
CLAR2	.58995	-.36019	-.11812
CLAR3	.78412	.04869	-.17349
CLAR4	.54674	.39463	-.11504
CLAR5	.63232	.50065	-.00005
CLAR6	.09616	.81327	-.04049
CLAR7	-.14512	-.06990	.73963
CLAR8	.06933	.54882	.56674
CLAR9	.37322	.51472	-.05436
CLAR10	-.01660	.34703	.66003
CLAR11	.62855	-.00881	.56158
CLAR12	.05884	-.01795	.66103
CLAR13	-.29952	.52702	.03967
CLAR14	.56825	.04904	.02795
CLAR15	.74500	.03383	.02994
CLAR16	.44680	-.00544	.21095

Programming Style

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3	FACTOR 4
STYLE1	-.32687	.31784	-.70405	-.32558
STYLE2	-.00102	.86058	-.21733	.01739
STYLE3	-.19090	.83146	-.12494	.14365
STYLE4	-.14642	.02383	.84666	.02205
STYLE5	.61753	-.24255	.13456	.16407
STYLE6	.58687	-.01243	-.[illegible]295	.11146
STYLE7	.55423	.01458	-.[illegible]072	-.18305
STYLE8	.54315	.34861	.33483	-.17823
STYLE9	.41708	.22865	.02433	.27436
STYLE10	.87187	-.19134	.02304	.05527
STYLE11	.24798	-.24933	.60707	.22470
STYLE12	.48739	.37805	.28041	.21431
STYLE13	.43621	.25648	-.02634	.74984
STYLE14	.20857	.68506	.23500	.06009
STYLE15	.39010	.49848	-.00024	.09470
STYLE16	-.12152	.17378	.63895	.52970
STYLE17	.09366	-.13670	.03142	.87082
STYLE18	.05806	.26215	-.04537	.80412
STYLE19	-.40658	.07875	.01918	-.21753
STYLE20	-.12750	.38779	-.67019	.22642

System Management

ROTATED FACTOR MATRIX:

	FACTOR 1	FACTOR 2	FACTOR 3	FACTOR 4
MANAGE1	-.15153	.28580	-.26765	.79499
MANAGE2	.59563	.00032	.00504	.23343
MANAGE3	.17721	.44494	.45150	.37460
MANAGE4	.65539	.20701	.01117	.03237
MANAGE5	.31733	-.65511	.32798	.00050
MANAGE6	.20584	.09036	.21199	.74311
MANAGE7	.16346	-.35173	.07180	.55144
MANAGE8	.07265	.81564	.08892	.16047
MANAGE9	.32098	.59775	.13389	-.18628
MANAGE10	.46326	.64892	.27543	.20455
MANAGE11	.49979	.25069	.52684	.22639
MANAGE12	.90512	-.05333	-.07958	-.05246
MANAGE13	.05233	.06770	.88239	-.02109
MANAGE14	-.45037	-.08803	.75144	-.01843