AD-A080 897 NAVAL AIR DEVELOPMENT CENTER WARMINSTER PA SOFTWARE --ETC F/G 9/2
A SCREENING CRITERION FOR DELIVERED SOURCE IN MILITARY SOFTWARE--ETC(U)
NOV 79 R J PARISEAU
UNCLASSIFIED NADC-79163-50-VOL-1 NL

1 OF 1
AD
A080897

END
DATE
FILMED
3-80
DDC

REPORT NO. NADC-79163-50

4 LEVEL II
NW

AD A080897

VOLUME I

A SCREENING CRITERION FOR DELIVERED SOURCE IN MILITARY SOFTWARE

Richard J. Pariseau
Software and Computer Directorate
NAVAL AIR DEVELOPMENT CENTER
Warminster, Pennsylvania 18974

14 November 1979

TECHNICAL NOTE
AIRTASK NO. ZF61412001
Work Unit No. GC333

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Prepared for
NAVAL AIR SYSTEMS COMMAND
Department the Navy
Washington, DC 20361

80 2 21 017

NOTICES

REPORT NUMBERING SYSTEM - The numbering of technical project reports issued by the Naval Air Development Center is arranged for specific identification purposes. Each number consists of the Center acronym, the calendar year in which the number was assigned, the sequence number of the report within the specific calendar year, and the official 2-digit correspondence code of the Command Office or the Functional Directorate responsible for the report. For example: Report No. NADC-78015-20 indicates the fifteeth Center report for the year 1978, and prepared by the Systems Directorate. The numerical codes are as follows:

CODE	OFFICE OR DIRECTORATE
00	Commander, Naval Air Development Center
01	Technical Director, Naval Air Development Center
02	Comptroller
10	Directorate Command Projects
20	Systems Directorate
30	Sensors & Avionics Technology Directorate
40	Communication & Navigation Technology Directorate
50	Software Computer Directorate
60	Aircraft & Crew Systems Technology Directorate
70	Planning Assessment Resources
80	Engineering Support Group

PRODUCT ENDORSEMENT - The discussion or instructions concerning commercial products herein do not constitute an endorsement by the Government nor do they convey or imply the license or right to use such products.

APPROVED BY: [illegible] DATE: 14 [illegible] 1997

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14) NADC-79163-50-VOL-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6) A Screening Criterion for Delivered Source in Military Software. Volume I.		5. TYPE OF REPORT & PERIOD COVERED 9) Technical Note
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10) Richard J. Pariseau 12) 32		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Air Development Center Software and Computer Directorate Warminster, PA 18974		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AIRTASK NO. ZF61412001 Work Unit No. GC333
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Air Systems Command Department of the Navy Washington, DC 20361		12. REPORT DATE
		13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 16) F61412 17) ZF61412001		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. [illegible]

PE 62766N

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

11) 14 Nov 79

18. SUPPLEMENTARY NOTES

Volume II; Appendices A, B, C, D, E, F, G, H.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Reliability
Investigation
Repair
Improvement

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The goal of this study is to identify measurable characteristics of the program source code that indicate the likelihood of future changes to the program modules. These changes include both repair of software errors and improvement in software performance.

Source code data and module change data were analyzed to correlate the source code characteristics with the number of changes made to the modules.
(Continued on reverse) over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393538

The following conclusions were reached: (1) a pair of source code parameters (size/structure) in a linear relationship is a useful predictor of the number of changes, (2) a single parameter is not sufficient as a predictor of fugure change and, (3) multiple parameters and nonlinear relationships do not significantly improve the prediction over the two parameter linear case.

The results can be made an acceptance criterion for delivered source modules prior to module testing. This application would support present Navy efforts to improve software reliability and maintainability through engineering techniques applied as early as possible in the software development process.

VOLUME I

For information and discussion.
The opinions expressed do not necessarily
represent the views of the Naval Air Development Center.

TABLE OF CONTENTS

LIST OF TABLES

INTRODUCTION

This report discusses an investigation into aspects of software reliability that may be characterized by measurable properties of the program source code (static analysis). Through examination of a large data base, size, structural, and syntactic characteristics of software modules have been related to the module histories of change. These relationships can be used as specification criteria for module characteristics when contracting for software. Additionally, they can be used to estimate maintenance requirements, to compare alternate implementations, to estimate changes to the software, and as input to failure models in order to estimate software MTFB (Mean Time Between Failure). These applications are practical in a FASP (Facility for Automated Software Production) environment (reference (a)) because the program source code is stored in a data base and the tools to automatically measure its size, structural, and syntactic characteristics fit naturally into the FASP concept.

GENERAL DISCUSSION

GOALS

The goal of this study was to determine a relationship between measurable characteristics of the program source code and the number of changes that are made to the program modules. This relationship would then be evaluated as the basis for a program module acceptance criterion.

BACKGROUND

Correct operation of military software is critical. The rapidly increasing use of software in tactical systems and in major communication systems has resulted in initiatives by the DOD (Department of Defense) and the Navy to improve the reliability of its software and to control the cost of achieving that reliability (reference (b)). These initiatives include evaluation of software quality at the earliest possible moment in the software development cycle.

The cost of military software is high. DOD software costs are presently in the three billion dollars per year range; new DOD software functions are occurring at a rate of 67 percent per year (reference (c)). About 75 percent of the cost occurs during the operation and maintenance phase of the system life cycle (references (c), (d), and (e)). About 50 percent of the remainder goes into the various developmental activities categorized as testing (references (d), (f), (g), and (h)).

The use of testing to detect and repair software errors is time consuming and expensive. In 1972, the United States Air Force spent over 750 million dollars on various software testing activities (reference (d)). Additionally, the relative cost of repair for software errors increases exponentially with time into the software life cycle (reference (i)). Therefore, large cost leverage can be expected from any software technique that reduces the testing or maintenance effort.

By using measurable characteristics of the program source code as a basis for acceptance of the delivered program modules, poorly implemented software can be rejected prior to the testing or maintenance phases. Because these characteristics are measurable, the acceptance criterion can be specified as a contractual requirement. This measurement would not identify errors, would not guarantee the absence of errors, and would not eliminate the need for testing. Rather, it would provide the ability to reject the program source code until it meets a minimum standard for reliability.

SUMMARY OF APPROACH

The overall approach of this study has been to:

1. Identify measurable characteristics of the source program that are related to software reliability.

2. Measure these characteristics for existing military software and collect data for the corresponding history of change to the software.

3. Determine a relationship between the measurable characteristics and the changes.

Identification of Measurable Characteristics. Measurements may be performed upon the program source language in terms of its size, its structure, and its syntax. Size properties categorize the amount of the program through counts of fundamental elements. Examples are the number of source statements and number of modules. Structural properties categorize program organization and flow of control. Examples are program hierarchy, path information, and nesting level. Syntactic properties categorize the source language and its use in the program. Examples are statement types, use of terminal symbols, and use of operands. McCabe has proposed a new structural measure (reference (j)). Halstead has proposed measures that can be classified as either size or syntax (reference (k)).

The identification portion of the study began in 1974 with an attempt to determine structural properties of modules that could be used as measures of complexity. The properties examined were the number of nodes, arcs, paths, and source statements present in each module. The study consisted of simulating the process of error detection in a module as a function of exercising paths through the module. Control structures and error histories for 44 procedures from the NTDS (Navy Tactical Data System) program were used as input to the simulation. The results are reported in references (l) and (m)).

The general conclusion of this initial study was that the four structural factors had potential use as measures of complexity, but that no single factor stood out as being a major determinant. It was, therefore, decided to include syntax in the measurable properties and to determine if the program complexity could be measured as a function of two or more parameters.

Data Collection. There are a number of problems in securing size, structure syntax, and reliability data for large military software systems. One problem is that these software systems require years to develop and are expected to have a long operational life. Usage of the software is required to produce reliability data, and these systems rarely exist as numerically significant multiple copies operating simultaneously. It is, therefore, necessary to accept the long term cost associated with establishing and maintaining a data collection mechanism over both the development period and some portion of the maintenance period.

A second problem is technological. Military software is presently developed on equipment configurations and under operating systems that lag commercial practice by two to four generations. It is only very recently that environments such as FASP have been created. These environments allow application of automatic tools to measure the software characteristics, maintain the results, and control the software development/maintenance process. Over the DOD, this capability is limited. Therefore, on present projects, data collection efforts will be manual, clerical activities. This has cost and schedule impact which project managers must accept.

There are constraints on public release of the data that effectively act to limit its availability for studies. In those cases in which the collection of data represents an expenditure of the contractor's resources, the information may be proprietary. For certain military systems, the data may be classified since it reflects on the reliability of those systems.

The homogeneity of the collected data is affected by the change in the technological base during the interval of data collection. Over the software life cycle or even over the development period, military software efforts often have:

1. The target hardware change from a prototype system with one set of capabilities to a production system with a different set of capabilities.

2. The implementation language pass through a number of revisions or even be completely respecified.

3. The operating system environment for both development and integration pass through radical changes.

4. The applied software engineering methodologies change due to either selection of new contractors or changes in approach within a specific contractor.

These effects must be considered both in terms of the homogeneity of the data being analyzed and the applicability of the results to a specific software environment.

Another factor affecting data homogeneity is the protean nature of the source program. Over the life cycle of the software, the program constantly changes due to the repair of failures and to program improvements. Measured values of size, structure, and syntax will vary while the failure data is collected. These changes are not necessarily minor and present a difficulty in interpretation of the result.

The data collection portion of the study began in 1973 as part of an experiment in the application of structured programming to military software. This experiment consisted of applying various software engineering methodologies (e.g., egoless programming teams, top down implementation, structured code) to the development of a tactical display processing program for the CV-TSC (Carrier Based Tactical Support Center) system software. The effort was typical of military software developments in that the assembly language, the operating system, the computer, and the display system were all prototypes. The final software comprised three programs totaling 578 modules consisting of 70,434 words of instructions resulting from 126,755 lines of source. The results of this experiment have been reported in reference (n)).

As part of the development methodology, detailed documentation at the module level was required for all corrections or modifications. This documentation began with approval of module design and has continued through implementation, testing and operation. The combination of availability of the source program, detailed module information throughout the life cycle, and close association with the work performed resulted in the selection of this software system as the subject of the study. The raw data was collected and analyzed in 1977.

Additional data collection efforts have been started. In 1975, a tool was developed to measure the size, structural, and syntactic characteristics of programs written in the Navy's CMS-2 language. The present version of this tool, SPAR (Source Program Analyzer and Reporter), has been used to analyze approximately 800,000 lines of source code from Navy tactical programs. It is intended to use this information in concert with failure histories of those programs to evaluate the initial conclusions of this study.

Analysis. The module source language has been measured in terms of software complexity characteristics. The change and modification histories have been analyzed in an attempt to categorize the nature of the change as "errors" or "modifications." Descriptive statistics for the various quantities have been calculated. Linear regression techniques have been used to determine relationships between the measured source properties and the various values for change in the software. The relationships have been evaluated.

DATA

Two steps were required to determine the specific values of the module source parameters and the module change history that are used in this study. The first involved collection of the program source images for the modules and collection of the documents that describe the changes to those modules during development and operation. The second consisted of analyzing the source images to determine the parameter values and analyzing the change documents to determine the category and quantity of change to the software modules.

COLLECTED DATA

The source program images and the corresponding change documentation were collected for that portion of the CV-TSC system software consisting of the tactical display processing program. Development of this software began in June 1973. Integration of the tactical display processing program had been completed in the system integration facility by September 1974. Initial delivery as part of the CV-TSC system software occurred in December 1974.

The tactical display processing program comprises 340 modules consisting of 35,018 words of instructions resulting from 75,692 lines of source. The source images were collected in April 1977 from the CV-TSC system library identified as: "Set J, 15 June 1977." These images were processed by various translators and system routines and now exist as files in the NAVAIRDEVCEN CCS (Central Computer System).

A requirement of the development methodology was that detailed documentation at the module level be provided for all corrections or modifications to approved design or code. These documents are called "change requests." They are hand-written and provide various amounts of information concerning the change. Appendix A is an example of a typical change request.

Approximately 800 change requests were available in April 1977. These spanned 3½ years from October 1973 to the end of March 1977. The last 2¼ years represented the period of operational usage and program maintenance. Only a small number of these 800 change requests fail to involve a change in the program source. The major problem in evaluating them lies in distinguishing between changes to correct an error and modifications to improve the program.

ANALYZED DATA

This data resulted from examination of the change requests and analysis of the source images of the tactical display processing program. The effort required to examine the change requests constrained the data to a portion of the program. This portion consisted of all tactical display processing software with the exception of those modules used to define the global data base and those modules that comprised the special purpose executive.

All of the source image data collected in April 1977 were processed by an analyzer that counted various size parameters and provided detailed data on the use of the syntax. Review of the listings for those modules corresponding to the examined change requests provided additional counts of structural parameters.

The resulting data were condensed into the final set of size, structural, and syntactic parameters and stored as a data base in the NAVAIRDEVCEN CCS. The stored data consist of parameter values and change values for 272 modules and 61 associated submodules comprising 16,915 words of instructions resulting from 53,614 lines of source. There are a total of 462 changes.

Modules and Submodules. The program development methodology specified the module to be the basic source unit building block for the program. A module was defined as a program unit that was discrete and identifiable with respect to assembly, combining with other units, and loading. Due to the development methodology, modules usually performed a single, specific function (i.e., had functional strength).

There were cases in which a parent module called various subordinate modules to perform related functions on a common data structure. In order to hide the data structure, the subordinate modules were physically located within the source image of the parent module, (i.e., the modules had informational strength). In these cases, the subordinate modules were termed "submodules." Although only modules existed as identifiable entities to the operating system, the submodules were treated as distinct and identifiable units with regard to the methodology of the development cycle.

For the purposes of this study, modules and submodules are treated identically. In the rest of this report, the term "module" will mean "module or submodule" unless specifically noted otherwise.

Change Request Data. Each change request was examined to determine which modules were affected. For each module, six items of information were recorded:

1. Change request identifying number.
2. Date of change request.
3. Implementing engineer.
4. Approving engineer.
5. Primary area of change (design, code, unknown).
6. Primary reason for change (error, modification, unknown).

A sample data sheet can be found in appendix B.

If the change was known to be a change in code but not a change in design, the area of change was designated as "code." If the change was known to be a design change as well as a change in code, it was designated as "design." Otherwise, the change area was designated as "unknown." This latter case occurred in three instances.

If the reason for the change was a known error as indicated on the change request or recalled by one of the engineers, it was designated as an "error."

If the change was a known modification (in the sense of an "improvement" to the original program) as indicated on the change request or recalled by one of the engineers it was designated as a "modification." Otherwise, the reason for change was designated as "unknown."

From this information, the following values were calculated and stored by module in a database in the CCS:

1. Total number of changes (Total Changes).
2. Total number of changes in design.
3. Total number of changes in code only.
4. Total number of unknown change areas.
5. Total number of known errors (Known Errors).
6. Total number of known modifications.
7. Total number of unknown reasons for change.
8. Possible Errors (Defined as: Total Changes - Known Modifications).

For the purposes of the study, Total Changes, Known Errors, and Possible Errors were selected as the measures of change to the program modules.

Source Code Data. The source code data resulted from the output of a program that counted size and syntactical parameters and from examination of the module listings for counts of structural parameters. Thirty parameters were collected for each module:

1. Number of cards.
2. Number of comment cards.
3. Number of processed cards.
4. Registers: Number of unique references.
5. Registers: Total number of references.
6. Op-Codes: Non-branching: Number of unique references.
7. Op-Codes: Non-branching: Total number of references.
8. Op-Codes: Branching: Number of unique references.
9. Op-Codes: Branching: Total number of references.

10. Op-Codes: Directives: Number of unique references.
11. Op-Codes: Directives: Total number of references.
12. Op-Codes: Total number of unique references.
13. Op-Codes: Total number of references.
14. Variables: Local names: Number of unique references.
15. Variables: Local names: Total number of references.
16. Variables: Global names: Number of unique references.
17. Variables: Global names: Total number of references.
18. Variables: Numbers: Number of unique references.
19. Variables: Numbers: Total number of references.
20. Variables: Local expressions: Number of unique references.
21. Variables: Local expressions: Total number of references.
22. Variables: Global expressions. Number of unique references.
23. Variables: Global expressions: Total number of references.
24. Variables: Total number of unique references.
25. Variables: Total number of references.
26. Total number of ITE's (IF-THEN-ELSE's).
27. Total number of DOW's (DOWHILE's).
28. Total number of DOU's (DOUNTIL's).
29. McCabe Number.
30. Program Clarity.

Various parameters are described below. Actual values for each module can be found in appendix C.

Card Counts (Items 1, 2, 3). The number of cards (item 1) includes all processed cards, all comment cards, all blank cards, and three cards related to job control (header, assembler on, assembler off). A processed card (item 3) is a source image containing an op-code or a directive. It is identical to the total number of op-code references (item 13).

Op-Code Counts (Items 6-13). Directives (items 10, 11) are assembler pseudo-ops. The total references (items 12, 13) are the sums of nonbranching op-codes (items 6, 7), branching op-codes (items 8, 9), and directives. The total number of op-code references (item 13) is identical to the number of processed cards.

Variable Counts (Items 14-25). Local names are variables defined internally to the module; global names are defined externally. Global names include the names of other modules when referenced.

Numbers have been counted as unique parameters when they were used as literals or as the operand in an equivalence statement. They were not counted when part of an expression defining addresses or constants.

Expressions in the operand field of the assembly statement result in values that represent single, unique items, whether they be addresses or constants. Therefore, expressions have been counted as single items and treated in the analysis as if they were an alternate form of spelling for a variable.

The total number of references (items 24, 25) are the sums of local variables, global variables, numbers, local expressions, and global expressions.

Structured Programming Constructs (Items 26, 27, 28). These are counts of the occurrences in the modules of the standard structured programming constructs.

McCabe's Number (Item 29). This is the cyclomatic complexity, V(G), proposed by McCabe in reference (j). It is a structural measure that may be viewed as the number of independent paths in the module. For a module with a directed graph, G, containing n vertices, e edges, and p connected components, it is defined as:

$$V(G) = e - n + p$$

McCabe shows that for structured programs the cyclomatic number is equal to one more than the number of predicates (i.e., decision points); therefore, this item was equal to the sum:

ITE (item 26) + DOW (item 27) + DOU (item 28) + 1

Program Clarity (Item 30). Halstead, in reference (k), suggests a number of measures that could be viewed as size or syntactic parameters. His suggested parameter, E, defined as "the total number of elementary mental discriminations required to generate a given program" was chosen as a possible measure of module clarity. It was calculated from the approximation:

$$E = \frac{[(N_1 + N_2) \log_2 (n_1 + n_2)] \, n_1 N_2}{2 n_2}$$

where: N_1 = Total Operator References.

= Total number of op-codes referenced (item 13).

N_2 = Total Operand References.

= Total number of register references (item 5) + total number of variable references (item 25).

n_1 = Number of Unique Operators.

= Total number of unique op-code references (item 12).

n_2 = Number of Unique Operands.

= Total number of unique register references (Item 4) + total number of unique variable references (item 24).

ANALYSIS

Analysis of the data consisted of calculating descriptive statistics for the data, calculating correlation coefficients between data items, and determining linear and nonlinear relationships between the data parameters (measures of size, syntax, and structure) and measures of changes to the software. The primary goal of the analysis was to determine a functional relationship that could be used to accept or reject source modules on the basis of predicted change to the module.

Analysis was performed on the NAVAIRDEVCEN's CCS. This is a large-scale, third generation commercial system consisting of two CDC (Control Data Corporation) 6600's and a CYBER 170/Model 175. Calculations were performed using the statistical library supplied as part of the operating system (reference (o)).

DESCRIPTIVE STATISTICS

There are 38 data items. Thirty are parameter values measured from the source image of the modules, and eight are counts of change to the modules as evaluated from the change requests. From the eight counts of change, TC (Total Changes), KE (Known Errors), and PE (Possible Errors) were selected as the measures of change.

The sum, the mean, the variance, and the standard deviation over the 333 data samples were calculated for each of the 38 data items. For each measure of change (TC, KE, and PE) the data items were grouped by value of that measure. Then the sum, the mean, the variance, and the standard deviation were calculated for each data item in each group.

The calculations were performed to provide a general understanding of the data, to aid in selection of significant parameters for the linear and nonlinear fits, and as a check on steps in the linear regression calculations. The results are listed in appendix D.

CORRELATION COEFFICIENTS

Correlation coefficients were calculated between each pair of the 38 data items. The calculations were performed to aid in identifying significant parameters and to aid in reducing the parameter sets for the multiparameter linear fits and the nonlinear fits. The correlation coefficients between the 38 data items and the measures of change (TC, KE, and PE) are presented in appendix E.

LINEAR RELATIONSHIPS

Two groups of linear fits were calculated. The first group involved selecting pairs of significant parameters from the set of 30 parameters and using multiple regression analysis to fit the pairs to each of the three measures of change (TC, KE, and PE). Two parameters were chosen to represent size or syntax and two parameters were chosen to represent structure. The four resulting size/structure pairs were:

1. Processed Source (PS))/McCabe Number (MN).

2. Processed Source (PS)/Total Branching Instructions (TBI).

3. Program Clarity (PC)/McCabe Number (MN).

4. Program Clarity (PC)/Total Branching Instructions (TBI).

The second group of linear fits involved using multiple regression analysis to fit about two-thirds of the data parameters to each of the measures of change. Elimination of parameters from the set to be fitted was subjective. Guidelines generally followed were to eliminate:

1. Parameters that essentially duplicated each other.

2. Parameters that represented a significant component of another parameter.

3. Parameters that were highly correlated with a parameter already included.

<u>Two Parameter Linear Fit</u>. The goal was to find a fit of the form:

$$C_i = K_1S_i + K_2G_i + K_3$$

where: C is a measure of change

S is a size/syntax parameter

G is a structure parameter

K's are constants.

The fit was then used to predict values of change, and the predictions were compared with the actual values in order to evaluate the linear function as an acceptance/rejection criterion for the modules.

Appendix F contains the 12 sets of results. Each set includes the defining function, a quantized scatter diagram of predicted values versus actual values, and an evaluation of the prediction. The evaluation indicates the percent of correct acceptance/rejection and the percent of incorrect acceptance/rejection for various levels of rejection.

Multiparameter Linear Fit. The data parameters were fitted to the three measures of change to achieve functions of the form:

$$C_i = \sum_{j=1}^{m} K_j P_{ij}$$

where: The m values of P_i represent the set of selected data parameters as described above.

P_1 is one of the size/syntax parameters (PS or PC).

P_2 is one of the structure parameters (MN or TBI).

The K's are constants.

As the number of independent variables in the regression formula is increased, the residual error for the predicted change value of the specific data set decreases. Thus, the resulting multiparameter function can be used to:

1. Indicate an upper bound to the quality that can be expected for a linear fit.

2. Indicate how well the parameters S and G (two parameter linear fit) represent the characteristics of the larger set of parameters.

Appendix G contains the 12 sets of results. Each set includes the defining function, a quantized scatter diagram of predicted values versus actual values, and an evaluation of the prediction. The evaluation indicates the percent of correct acceptance/rejection and the percent of incorrect acceptance/rejection for various levels of rejection.

NONLINEAR RELATIONSHIPS

The goal was to determine if a nonlinear function would improve the prediction of the measures of change. The functional form selected was:

$$C_i = S_i^{\alpha} G_i^{\beta} \sum_j K_j P_{ij}$$

where: C is the measure of change.

S and G are the significant size or structure parameters as described above.

The P_i's are the remaining data parameters selected as described above and include the constant, "1."

The K's, α, and β are constants.

The values for α and β were determined through multiple regression analysis on the data:

$$\log_{10}(C_i + 1) = \alpha\log_{10}S_i + \beta\log_{10}G_i + \gamma$$

Appendix H contains the 12 sets of results. Each set includes the defining function, a quantized scatter diagram of predicted values versus actual values, and an evaluation of the prediction. The evaluation indicates the percent of correct acceptance/rejection and the percent of incorrect acceptance/rejection for various levels of rejection.

RESULTS

Results are discussed for the areas of descriptive statistics of the data, correlation coefficients, and functional relationships resulting from the multiple regression analysis. It should be emphasized that these results are based upon data representing a unique software development environment (e.g., language, support facilities, methodologies, personnel). Their general applicability must be further considered in terms of the similarity of other software development environments to the environment that has been studied.

DESCRIPTIVE STATISTICS

Appendix D contains four sets of descriptive statistics of the data. The first set consists of the sum, mean, variance, and standard deviation of the data items over all samples. In the second set, the data have been grouped by value of the Total Change data item. The sum, mean, variance and standard deviation of each data item in each group has been calculated. The third and fourth sets have been processed similarly in terms of values of the Known Errors and Possible Errors data items.

All Samples. Table I presents descriptive statistics for an average module. These are rounded values taken from the complete set in appendix D. Table I shows that the development methodology resulted in small, uncomplicated modules that required little change. The main points of interest are:

1. An average module is small and heavily commented.

2. Decision branching is largely due to IF-THEN-ELSES as opposed to DOWHILES or DOUNTILS.

3. Branching in the module is largely due to the two branches per structured construct plus the single return to the calling module.

4. The mean value of McCabe's Number is low.

5. The mean value of Program Clarity is low.

6. The mean values for the measures of change are low.

TABLE I

Descriptive Statistics for an Average Module

Data Item	Mean	Standard Deviation
Processed Source Cards	51	48
Comment Cards	105	97
Total Branching Instructions	11	12
McCabe Number	5.5	5.8
Program Clarity	42K	84K
IF-THEN-ELSES	4.0	5.4
DOWHILES	0.1	0.3
DOUNTILS	0.4	0.8
Total Changes	1.4	2.1
Known Errors	0.7	1.4
Possible Errors	1.0	1.7

Grouped Samples. Tables II, III, and IV present the means and standard deviations for selected data items grouped according to the number of occurrences of Total Changes, Known Errors, and Possible Errors respectively. The data items presented are those used in the two parameter linear fits (Processed Source, Total Branching Instructions, McCabe Number, and Program Clarity). The values have been rounded from the results in appendix D.

In general, the mean values of the grouped data increase nonlinearly with measure of change. Specific points do not conform. For instance, the mean values at 6 Known Errors (2 samples) and the mean values at 6 and 7 Possible Errors (3 samples) are low compared with adjacent groups. However, the overall results indicate that larger, more complicated modules will be changed more frequently. At the same time, tables II, III, and IV indicate that no single source code parameter is better than the others and that no single measure of change is superior to the others.

TABLE II

Means (M) and Standard Deviations (SD) for Selected Data as a Function of Occurrences of Total Changes

	TOTAL CHANGES											
	0	1	2	3	4	5	6	7	8	10	13	15
Occurrences	152	77	45	20	12	10	5	5	3	2	1	1
Processed Source Cards (M)	35	46	59	70	83	72	103	77	176	174	183	327
(SD)	35	41	39	41	47	31	45	61	159	47		
Total Branching Instructions (M)	8	10	13	15	19	15	25	10	35	35	32	104
(SD)	8	9	11	10	11	7	18	4	34	21		
McCabe Number (M)	4.2	4.8	5.9	7.3	8.3	7.7	9.8	4.2	19.7	13.5	15	51
(SD)	4.2	4.8	4.9	5.2	5.1	4.2	8.9	0.8	21.1	7.8		
Program Clarity (M)	24K	34K	45K	52K	85K	54K	108K	42K	355K	238K	303K	541K
(SD)	53K	53K	55K	56K	108K	39K	111K	38K	50K	125K		

TABLE III

Means (M) and Standard Deviations (SD) for Selected Data as a Function of Occurrences of Known Errors

	KNOWN ERRORS								
	0	1	2	3	4	5	6	7	11
Occurrences	232	49	23	15	5	4	2	2	1
Processed Source Cards (M)	40	63	67	84	73	128	49	283	327
(SD)	38	41	43	48	25	57	28	107	
Total Branching Instructions (M)	9	14	15	17	15	25	8	61	104
(SD)	9	11	11	10	7	14	6	17	
McCabe Number (M)	4.4	6.2	7.0	8.5	6.6	10.5	4.0	31.0	51.0
(SD)	4.4	5.1	5.3	5.0	4.6	8.4	1.4	17.0	
Program Clarity (M)	28K	46K	51K	85K	57K	181K	22K	629K	541K
(SD)	54K	50K	56K	100K	35K	142K	16K	429K	

TABLE IV

Means (M) and Standard Deviations (SD) for Selected Data as a Function of Occurrences of Possible Errors

		POSSIBLE ERRORS										
		0	1	2	3	4	5	6	7	8	10	15
Occurrences		180	78	29	19	14	5	2	1	3	1	1
Processed Source Cards	(M)	37	47	72	69	99	98	49	30	225	183	327
	(SD)	37	34	46	41	44	43	28		125		
Total Branching Instructions	(M)	8	10	16	15	23	19	8	11	50	32	104
	(SD)	9	8	12	11	12	13	6		23		
McCabe Number	(M)	4.3	4.7	7.1	7.3	10.3	7.8	4.0	5.0	25.3	15.0	51.0
	(SD)	4.5	3.8	5.2	5.3	5.5	7.3	1.4		15.5		
Program Clarity	(M)	26K	32K	58K	51K	103K	105K	22K	8K	458K	303K	541K
	(SD)	53K	46K	62K	55K	96K	112K	16K		424K		

Another way of looking at these results is that they partition into three regions which can be viewed as low, medium, and high measures of the particular type of change (Total Changes, Known Errors, Possible Errors). Table V presents these regions and the associated mean values for Processed Source, Total Branching Instructions, McCabe Number, and Program Clarity of the data within each region.

Table V indicates that for this software development environment it would have been possible to use measurements of size/structure parameters as a basis for accepting modules that would have low occurrences of change and rejecting modules that would have high occurrences of change. It can also be seen from table V that the results would be similar regardless of which measure of change is selected.

TABLE V

Mean Values of Parameters for Regions of Change

	Low Change	Medium Change	High Change
Total Changes	(0, 1)	(2-7)	(8-15)
Processed Source	39	69	198
Total Branching Instructions	9	15	44
McCabe Number	4.4	6.8	21.7
Program Clarity	27K	55K	241K
Known Errors	(0)	(1-4)	(5-11)
Processed Source	40	69	167
Total Branching Instructions	9	15	41
McCabe Number	4.4	6.9	18.1
Program Clarity	28K	55K	285K
Possible Errors	(0, 1)	(2-7)	(8-15)
Processed Source	40	77	237
Total Branching Instructions	9	17	57
McCabe Number	4.4	7.7	28.4
Program Clarity	28K	67K	444K

CORRELATION COEFFICIENTS

Table VI presents the values of the correlation coefficients between selected data items and the three measures of change (Total Changes, Known Errors, Possible Errors). The table includes Processed Source, Total Branching Instructions, McCabe Number and Program Clarity. No single data item correlates well with any of the measures of change.

TABLE VI

Correlation Coefficients Between Selected Data Items and Measures of Change

Data Item	Total Changes	Known Errors	Possible Errors
Processed Source Cards	0.53	0.51	0.55
Comments	0.49	0.49	0.54
Total Branching Instructions	0.48	0.49	0.55
McCabe Number	0.43	0.46	0.51
Program Clarity	0.46	0.51	0.52
IF-THEN-ELSES	0.44	0.47	0.51
DOWHILES	0.15	0.19	0.20
DOUNTILS	0.15	0.14	0.17

FUNCTIONAL RELATIONSHIPS

An evaluation of the various functional relationships as a source program module acceptance criterion is presented in table VII. This data has been excerpted from the results in appendices F, G, and H. Data values have been rounded.

The functional relationships resulting from the linear and nonlinear fits have been used to predict the amount of change (Total Changes, Known Errors, Possible Errors) for each module. The low/medium change boundary indicated by the grouped statistics of table V has been used as the module acceptance/rejection criterion. The results as predicted from the functional relationships have been compared with the actual changes indicated by the data. The evaluation in table VII is expressed as the percent of modules correctly accepted/rejected and the percent of modules incorrectly accepted/rejected.

Table VII indicates that for this software development environment any of the 36 developed equations could have been successfully used as a screening criterion. Correct acceptance and rejection of source modules would have occurred in approximately 70 percent of the cases. The remaining cases are approximately split between incorrect acceptance and incorrect rejection of the source modules.

Table VII also indicates that there is no significant improvement in acceptance and rejection if a larger number of parameters (multiparameter

TABLE VII

Evaluation of Acceptance Criterion for Low Change Areas

	Two Parameter Linear Fit				Multi-Parameter Linear Fit				Non-Linear Fit			
	1	2	3	4	1	2	3	4	1	2	3	4
TOTAL CHANGES <2												
Correct Acceptance (%)	52	52	57	54	54	55	55	55	57	57	57	57
Correct Rejection (%)	19	18	12	15	23	23	23	23	20	21	20	20
Incorrect Acceptance (%)	12	14	19	16	9	9	9	9	11	10	11	11
Incorrect Rejection (%)	17	17	12	15	14	14	14	14	12	11	12	11
KNOWN ERRORS <1												
Correct Acceptance (%)	45	44	49	46	46	44	46	46	49	50	50	50
Correct Rejection (%)	21	22	19	19	25	24	24	24	23	23	23	23
Incorrect Acceptance (%)	9	9	11	11	5	6	6	6	7	7	8	8
Incorrect Rejection (%)	25	25	21	23	24	25	24	23	21	20	20	20
POSSIBLE ERRORS <2												
Correct Acceptance (%)	67	67	70	68	68	68	67	67	68	69	68	68
Correct Rejection (%)	9	10	7	8	13	13	13	13	12	12	12	12
Incorrect Acceptance (%)	13	12	16	14	9	9	10	10	11	11	10	10
Incorrect Rejection (%)	11	11	8	10	10	10	11	11	10	9	10	10

linear fit) or a more complicated relationship (nonlinear fit) are used. This is significant in that it simplifies the practical problem of measuring the source code by reducing the number of parameters that must be measured.

The coefficient values for the twelve equations resulting from the two parameter linear fits are presented in table VIII. The coefficient values for the multiparameter linear fits and the nonlinear fits can be found in appendices G and H.

CONCLUSIONS

The goal of this study was to determine a functional relationship between various parameters of the module source code and various measures of change to the module. This relationship would be used as the basis of an acceptance criterion for delivered source code. The results of tables II, III, IV, V, and VI show that a single source code parameter does not provide sufficient information for an acceptance criterion. The results of table VII show that a linear equation in a size parameter and a structural parameter provides a sufficient basis for a practical acceptance criterion.

Table VII indicates that a successful prediction is made by any of the functional relationships in about 70 percent of the cases. In about 15 percent of the cases, the source code is incorrectly rejected. Incorrect rejection represents a prediction of more changes to the module than would actually result. The implementor is required to recreate software modules that actually would have had an acceptable number of errors/changes because they are evaluated as "too complicated" by the screening criterion. They will be accepted when they are simplified. Over the life cycle of the software system, this clarification will have benefits as the software is modified to adapt to changing requirements. The re-doing of the software is being performed in early development when it costs the least.

In about 15 percent of the cases, the source code is incorrectly accepted. Incorrect acceptance represents a prediction of fewer changes to the module than will actually occur. These errors/changes to the module will have to be detected during testing or operation. This is no worse than the situation presently accepted in software development.

Incorrect acceptance is the least desired case since it involves changes to the software during the testing, integration, or maintenance phases. The lowest values of incorrect acceptance occur in table VII when the measure of change is Known Errors. For these cases, incorrect acceptance of modules that are to have less than one known error will occur about 10 percent of the time regardless of which of the two parameter linear equations is selected. From table VIII, the four equations are:

(1) $KE = 0.014\ (PS) + 0.00042\ (MN) - 0.072$

(2) $KE = 0.011\ (PS) + 0.015\ (TB) - 0.074$

TABLE VIII

Coefficient Values for the Two Parameter Linear Fits

	Processed Source (PS)	Program Clarity (PC)	McCabe Number (MN)	Total Branching Instructions (TB)	Constants
TOTAL CHANGES					
1	$0.3459X10^{-1}$		-0.1056		0.2067
2	$0.2449X10^{-1}$			$-0.6526X10^{-2}$	0.2171
3		$0.8856X10^{-5}$	$0.4498X10^{-1}$		0.7696
4		$0.4978X10^{-5}$		$0.5699X10^{-1}$	0.5325
KNOWN ERRORS					
1	$0.1421X10^{-1}$		$0.4184X10^{-3}$		$-0.7243X10^{-1}$
2	$0.1086X10^{-1}$			$0.1538X10^{-1}$	-0.7436X10-1
3		$0.6697X10^{-5}$	$0.2404X10^{-1}$		0.2389
4		$0.5174X10^{-5}$		$0.2568X10^{-1}$	0.1431
POSSIBLE ERRORS					
1	$0.1763X10^{-1}$		$0.1913X10^{-1}$		$0.2732X10^{-1}$
2	$0.1048X10^{-1}$			$0.4191X10^{-1}$	$0.2019X10^{-1}$
3		$0.6566X10^{-5}$	$0.7023X10^{-1}$		0.3679
4		$0.4344X10^{-5}$		$0.5573X10^{-1}$	0.2131

(3) $KE = 0.0000067\ (PC) + 0.024\ (MN) + 0.24$

(4) $KE = 0.0000052\ (PC) + 0.026\ (TB) + 0.14$

where: KE is Known Errors.

PS is Processed Source.

MN is McCabe's Number.

PC is Program Clarity.

TB is Total Branching Instructions.

Choice of an equation should be made on the basis of the difficulty of measuring a particular pair of parameters. For example, McCabe's Number is easily measured for structured code. When the code is not structured, Total Branching Instructions is easier to measure. As shown by table VII, it does not matter which equation is selected.

Table VII also shows that there is little gain in extending the functional relationships to many parameters or to more complicated functions. There is small improvement in predicting ability between the two parameter linear fit and the multiparameter and nonlinear fits. This is encouraging since the simpler relationship is more practical both in terms of specifying software contracts and in terms of application when implementing the software.

While the developed functional relationships are specific only for this set of data, the results indicate that static analysis can produce a practical screening criterion applicable at a very early point in the software development process. It is to be expected that the coefficient values and screening boundary will vary with such development environment characteristics as programming language, personnel, support facility, and methodology. However, the results of this study indicate that a simple relationship of size, structural, and syntactic parameters can be developed as a practical screening criterion. If this type of analysis were performed consistently over many software developments, a collection of relationships corresponding to the different environments would result. The software purchaser could then pick the criterion corresponding to his development environment. For the present, the developed relationships can be applied to those environments similar to that of reference (n).

RECOMMENDATIONS

It is recommended that the results of this study be verified by expanding the analysis to other software development environments. These environments should reflect present DOD policy with regard to HOL's (high order languages) and modern software development methodologies. At the NAVAIRDEVCEN, this is embodied in FASP. FASP has the following advantages for the purposes of this analysis:

1. It is utilized by large software development and maintenance projects. Large developments are necessary in order to generate sufficient data for analysis. Large developments also represent the most valuable point of application for the screening criterion.

2. Tools to measure the size, structure, and syntax of software modules and to calculate the predicted change to those modules may be implemented in FASP as preprocessors to the language translators. This assures that measurement will be made each time a module is translated.

3. FASP Software Management Reports already track size and change at the module level. Expansion of this tracking to include structure and syntax, predicated change, and to differentiate between error correction and module modification is feasible.

It is therefore recommended that measuring tools be implemented in FASP for the following military computer languages:

COMPUTER	HOL	AL	NAVAIRDEVCEN PROJECTS
AN/UYK-7	CMS-2Y	ULTRA-32	CV-TSC, S-3A
AN/UYK-20	CMS-2M SPL/I(M)	MACRO-20/14	CV-TSC, LAMPS
AN/UYS-1	SPL/I	SPL	ASP, P-3C, LAMPS

It is also recommended that the FASP Software Management Reports be modified to allow tracking of module structure and syntax, tracking of predicted change, and to allow user specification of change category (error correction or module modification).

Following implementation of these changes, it is recommended that a three year study be performed for those projects developing and/or maintaining software in the above languages on FASP. It is expected that this study will result in a family of screening criteria applicable to specific languages utilizing the FASP software development environment.

It is further recommended that those projects at NAVAIRDEVCEN utilizing a software development environment similar to the one of the present study be required to:

1. Utilize the indicated screening criterion as an element of their software module acceptance procedure.

2. Historically track both the measures of the modules and their changes.

This data shall be used to verify or improve the screening criterion.

REFERENCES

(a) Naval Air Development Center FASP (Facility for Automated Software Production) Brochure, Nov 1977.

(b) Dennis W. Farrell, "Navy Airborne Weapon System Software Acquisition," Defense Systems Management Review, Vol. 1, No. 6.

(c) W.Myers, "The Need for Software Engineering," Computer, Feb 1978, pp. 12-26.

(d) Glenford J. Myers, "Software Reliability, Principles and Practices," John Wiley and Sons, 1976.

(e) G. Schick and R. Wolverton, "An Analysis of Competing Software Reliability Models," IEEE Transactions on Software Engineering, Vol. SE-4, No. 2, Mar 1978, pp. 104-120.

(f) B.W. Boehm, "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973.

(g) Honeywell, Incorporated, "Software Management Seminar" Brochure, Jan 1978.

(h) R. Wolverton, "The Cost of Developing Large-Scale Software," IEEE Transactions on Computers, Vol. C-23, No. 6, Jun 1974, pp. 615-636.

(i) B. W. Boehm, "Software Requirements and Design Aids," in "Software Reliability: Infotech State of the Art Report; 2: Invited Papers," Infotech International, 1977.

(j) T.J.McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec 1976, pp. 308-320.

(k) Maurice H. Halstead, "Elements of Software Science," Elsevier North-Holland, Inc. 1977.

(l) T.F.Green, G.T.Howard, R.J.Pariseau, N.F.Schneidewind, "Program Structures, Complexity, and Error Characteristics," presented at the Symposium on Computer Software Engineering, Polytechnic Institute of New York, 20-22 Apr 1976.

(m) G.T. Howard, M.Kirchgaessner, N.F.Schneidewind, Report No. NPS52SS76111, "Software Error Detection: Models, Validation Tests, and Program Complexity Measures," Naval Postgraduate School, No. 1976.

(n) R.J.Pariseau, Report No. NADC-76044-50, "Improved Software Productivity for Military Computer Systems Through Structured Programming," Naval Air Development Center, 12 Mar 1976.

(o) "Control Data 6000 Series Computer Systems Statistical Subroutines Reference Manual," Publication No. 60135300, Control Data Corporation, June 1966.

D I S T R I B U T I O N   L I S T

REPORT NO. NADC-79163-50

AIRTASK NO. ZF61412001
Work Unit No. GC333

	No. of Copies
NAVAIRSYSCOM, AIR-950D (2 for retention) (2 for AIR-530)	4
DDC	12