ADLIF - A Structured Design Language for Metric Analysis

by

Calvin Lee Selig

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

APPROVED:

Dr. Sallie M. Henry, Chairperson

Dr. Dennis G. Kafura

Dr. James D. Arthur

Blacksburg, Virginia

ADLIF - A Structured Design Language for Metric Analysis

by

Calvin Lee Selig

Dr. Sallie M. Henry, Chairperson

Computer Science and Applications

(ABSTRACT)

Since the inception of software engineering, the major goal has been to control the development and maintenance of reliable software. To this end, many different design methodologies have been presented as a means to improve software quality through semantic clarity and syntactic accuracy during the specification and design phases of the software life cycle. On the other end of the life cycle, software quality metrics have been proposed to supply quantitative measures of the resultant software. This study is an attempt to unify the two concepts by providing a means to determine the quality of a design before its implementation.

Acknowledgements

I would like to thank my advisor, Dr. Sallie Henry, for her continued support throughout this research. Her assistance and friendship through the long process of developing this thesis have been greatly appreciated. I would also like to thank my other committee members Dr. Dennis Kafura and Dr. James (Sean) Arthur. A very heartfelt thanks goes out to Geoff Vining and Ann Jenson who work in the statistics consulting lab. The analysis portion of this study would have been incomplete, and the results questionable without their help.

A special thanks goes out to for his moral support in this undertaking. He was always there when I needed someone to bounce ideas off of, and when I needed to either yell or lose my sanity. His assistance with both SCRIPT and the analyzer was also invaluable. I would also like to thank for his time and dedication in proofing this work, as well as , , and whose help made debugging of the analyzer used in this research possible. Other people whose friendship was instrumental in the maintenance of my sanity are , , , and . I am also grateful to and for their support and understanding during the development of this research.

Last, but certainly not least, I would like to thank my family: , , , , and , and . Their encouragement and belief in me were, at times, all that

kept me going. Finally, thank you to all of my friends too numerous to mention. Even when you didn't know it, your friendship was necessary for the completion of this work.

Table of Contents

Chapter 1 Introduction to Software Quality Metrics 1

Introduction 1

Code Metrics 6

Structure Metrics 10

Hybrid Metrics 11

Conclusion 12

Chapter 2 Design Methodology Review 14

Introduction 14

General Design Techniques 15

Detailed Design 16

Existing PDLs 18

PDLs and Metrics 20

Conclusion 22

Chapter 3 Definition of ADLIF 23

Introduction to ADLIF 23

Comments 24
Built-in Functions and Procedures 25
Pointers 26
Parameter passage 27
FOR Loop 30
Packages 31
Conclusion 34

Chapter 4 Data Description 35
Introduction 35
Data Collection 36
Data Preparation 37
Conclusion 39

Chapter 5 Analysis of Project Data 40
Introduction 40
Statistical Analysis 40
Conclusion 44

Chapter 6 Analysis of Combined Data 46
Introduction 46
Analysis Methods 46
Comparison of all Procedure Complexities 47
Metric Comparison by Level of Refinement 52
Inter-metric Correlations 67
Conclusion 72

Chapter 7 Conclusions 75

Introduction 75
Research Conclusions 75
Future Work 76
Conclusion 77

Bibliography **78**

Appendix A. ADLIF Language Reference Manual **81**
Character Set 81
Names 82
Comments 82
Arithmetic Precedence 83
Built-In Functions and Procedures 84
General Format of a Program 88
Program Statement 88
Declaration Section 89
USE Clause 89
Constants 90
Character Types 90
Boolean Types 92
Integer Types 92
Real Types 93
Record Types 94
Array Types 95
Pointers 96
Variable Declarations 98
Subprograms 98
Procedures 98

Procedure Calls 100
Functions 100
Statements 104
Read and Write Statements 104
If Statements 107
Loop Statements 109
CASE Statement 111
PACKAGE USAGE 112

Appendix B. BNF for ADLIF 115

Appendix C. Project Data 124

Appendix D. Data Plots 171

Vita 199

List of Illustrations

Figure 1. Diagram of Currently Used Software Life Cycle 3
Figure 2. Diagram of Proposed Reduced Software Life Cycle 4
Figure 3. Diagram of Metric Generation Process 5
Figure 4. Parameter Modes 29
Figure 5. Example of Package Usage 33
Figure 6. Example of a Poorly Specified Procedure 45
Figure 7. Regression and 95% Confidence lines for LOC 53
Figure 8. Regression and 95% Confidence lines for INFO 54
Figure 9. Refinement Level Algorithm 56
Figure 10. Refinement Level Algorithm continued... 57
Figure 11. Refinement Level Algorithm continued 58
Figure 12. Regression and 95% Confidence lines for Highly Refined LOC 68
Figure 13. Regression and 95% Confidence lines for Least Refined LOC 69
Figure 14. Regression and 95% Confidence lines for Highly Refined INFO 70
Figure 15. Regression and 95% Confidence lines for Least Refined INFO 71
Figure 16. Example of Procedure Declaration 101
Figure 17. Example of Function Declaration 103
Figure 18. Example of Package Usage 114
Figure 19. Regression and 95% Confidence lines for LOC 172
Figure 20. Regression and 95% Confidence lines for N 173
Figure 21. Regression and 95% Confidence lines for V 174

Figure 22. Regression and 95% Confidence lines for E 175
Figure 23. Regression and 95% Confidence lines for CC 176
Figure 24. Regression and 95% Confidence lines for INFO 177
Figure 25. Regression and 95% Confidence lines for INFO-L 178
Figure 26. Regression and 95% Confidence lines for INFO-E 179
Figure 27. Regression and 95% Confidence lines for INFO-CC 180
Figure 28. Low Refinement Regression and 95% Confidence lines for LOC 181
Figure 29. Low Refinement Regression and 95% Confidence lines for N 182
Figure 30. Low Refinement Regression and 95% Confidence lines for V 183
Figure 31. Low Refinement Regression and 95% Confidence lines for E 184
Figure 32. Low Refinement Regression and 95% Confidence lines for CC 185
Figure 33. Low Refinement Regression and 95% Confidence lines for INFO 186
Figure 34. Mid Refinement Regression and 95% Confidence lines for LOC 187
Figure 35. Mid Refinement Regression and 95% Confidence lines for N 188
Figure 36. Mid Refinement Regression and 95% Confidence lines for V 189
Figure 37. Mid Refinement Regression and 95% Confidence lines for E 190
Figure 38. Mid Refinement Regression and 95% Confidence lines for CC 191
Figure 39. Mid Refinement Regression and 95% Confidence lines for INFO 192
Figure 40. High Refinement Regression and 95% Confidence lines for LOC 193
Figure 41. High Refinement Regression and 95% Confidence lines for N 194
Figure 42. High Refinement Regression and 95% Confidence lines for V 195
Figure 43. High Refinement Regression and 95% Confidence lines for E 196
Figure 44. High Refinement Regression and 95% Confidence lines for CC 197
Figure 45. High Refinement Regression and 95% Confidence lines for INFO 198

List of Tables

Table 1. Metric Abbreviations Used in Data Presentation 41
Table 2. Correlations for all Projects Using all Nine Metrics 43
Table 3. Correlations of all Design Procedures to all Source Procedures 48
Table 4. Regression Line Equations and Statistics 49
Table 5. Extra Statistics 51
Table 6. Correlations by Level of Refinement 60
Table 7. Low Refinement Regression Line Equations and Statistics 61
Table 8. Middle Refinement Regression Line Equations and Statistics 62
Table 9. High Refinement Regression Line Equations and Statistics 63
Table 10. Extra Statistics for Low Refinement Level 64
Table 11. Extra Statistics for Middle Refinement Level 65
Table 12. Extra Statistics for High Refinement Level 66
Table 13. Intermetric Correlations for the Pascal code 73
Table 14. Intermetric Correlations for ADLIF code 74
Table 15. Design and Source Metric Values for Project: AKS 125
Table 16. Design and Source Metric Values for Project: CASINO 127
Table 17. Design and Source Metric Values for Project: CHART 129
Table 18. Design and Source Metric Values for Project: CLUE 131
Table 19. Design and Source Metric Values for Project: DUNGEON 134
Table 20. Design and Source Metric Values for Project: FORM 136
Table 21. Design and Source Metric Values for Project: GAMMON 137

Table 22. Design and Source Metric Values for Project: GOLF 139
Table 23. Design and Source Metric Values for Project: MONOPOLY 140
Table 24. Design and Source Metric Values for Project: MONOPOLY2 142
Table 25. Design and Source Metric Values for Project: MORLOC 144
Table 26. Design and Source Metric Values for Project: MTCLIMB 145
Table 27. Design and Source Metric Values for Project: OTHELLO 146
Table 28. Design and Source Metric Values for Project: OTHELLO2 147
Table 29. Design and Source Metric Values for Project: PENTE 148
Table 30. Design and Source Metric Values for Project: POKER 151
Table 31. Design and Source Metric Values for Project: SCHEDULE 153
Table 32. Design and Source Metric Values for Project: SCORE4 155
Table 33. Design and Source Metric Values for Project: STARTREK 156
Table 34. Design and Source Metric Values for Project: STRATEGO 159
Table 35. Design and Source Metric Values for Project: SUBQUEST 160
Table 36. Design and Source Metric Values for Project: TALLY 161
Table 37. Design and Source Metric Values for Project: TWIXT 163
Table 38. Design and Source Metric Values for Project: VEGAS 165
Table 39. Design and Source Metric Values for Project: XWORD 167
Table 40. Design and Source Metric Values for Project: YAHTZEE 169
Table 41. Design and Source Metric Values for Project: YAHTZEE2 170

Chapter 1 Introduction to Software Quality Metrics

Introduction

In the last decade, the field of computer science has undergone a revolution. It has started the move from a mysterious art form to a detailed science. The vehicle for this progress has been the rising popularity of the field of software engineering. This innovative area of computer science has brought about a number of changes in the way we think of, and work with, the development of software. Due to this renovation, a field that started with little or no design techniques and unstructured, unreliable software has progressed to a point where a plethora of techniques exist to improve the quality of a program design as well as that of the resultant software. The popularity of structured design and coding techniques prove that there is widespread belief that the overall product produced using these ideas is somehow better. Statistics seem to indicate that this belief is true. Until recently, however, there existed no technique for quantitatively showing that one program is better than its functional equivalent. In the past few years, the use of software quality metrics seems to indicate that such a comparison is not only possible, but valid.

A typical software life cycle consists of requirements definitions, program design, implementation, testing, and finally maintenance [OVEC86]. The portion of the cycle that is of interest to this re-

search is that of design and implementation with the inclusion of software quality metrics. Figure 1 on page 3 contains a diagram of this part of the software life cycle using complexity metrics. First, a design is created and implemented in software. At that point, software quality metrics are generated for the source code. If necessary, as indicated by the metrics, the cycle returns to the design phase. Ideally, the software life cycle can be "reduced" to that in Figure 2 on page 4, where the metrics are generated during the design phase, before code implementation. This modified cycle will eliminate the generation of undesirable source code, since it is possible to use the metrics, exactly as before, only earlier. The goal of this study is to indicate the plausibility of using the "reduced" cycle to increase the efficiency of the software development process by implementing metric analysis as early as possible.

The goal of shortening the loop in the life cycle is highly dependent on the ability to perform the metrical measures on the design, along with the need for evidence that the metric values produced from the design reflect the quality of the resultant source code. To facilitate this ability, a software metric generator, which for purposes of this study may be considered a "black box," is provided that takes as input either the design or the source code and produces, as output, a number of complexity metric values. A diagram of the metric generation process is shown in Figure 3 on page 5.

Many software quality metrics have been developed in recent years [CONS86] [HALM77] [HENS79] [MCCC78] [MCCT76] [WOOM79] [YAUS80] to name just a few. Some of the existing metrics are qualitative and therefore non-automatable. These types of measures are not considered in this study, which focuses on metrics that are both quantitative and automatable. Metrics of this type can be put into three general categories: code or micro metrics, structure or macro metrics, and hybrid metrics. In general, code metrics are those that measure an attribute such as length, number of control statements, number of tokens, etc. That is, code metrics produce a "count" of some feature of the source program. Structure metrics attempt to capture the logic or semantics of the source program. Finally, Hybrid metrics consist of one or more code metrics combined with one or more structure metrics. Although both code and structure metrics result in a number that

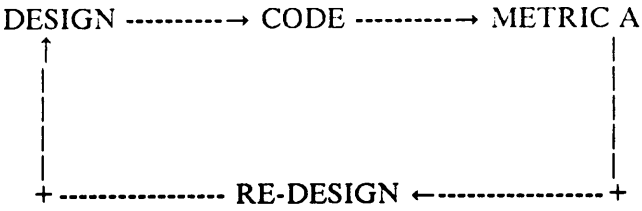

Figure 1. Diagram of Currently Used Software Life Cycle

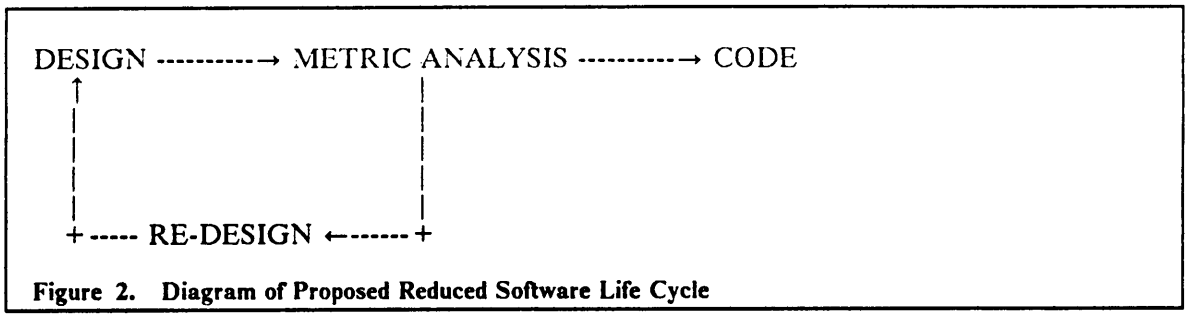

Figure 2. Diagram of Proposed Reduced Software Life Cycle

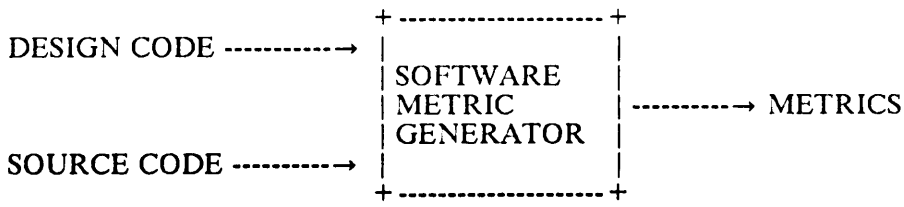

Figure 3. Diagram of Metric Generation Process

somehow represents the "goodness" of a program, it has been shown that the two types of metrics are measuring different features of the source systems [HENS81b]. In the following sections, some representative metrics from each category are discussed in some detail.

Code Metrics

Many code metrics have been proposed in the recent past. An effort has been made to limit this discussion to a few of the more popular ones that are typical of this type of measure. They include lines of code, parts of Halstead's Software Science, and McCabe's Cyclomatic Complexity. Each of these metrics is widely used and has been extensively validated [CANJ85] [ELSJ78] [REDG84].

Lines Of Code

The most familiar software measure is the count of the lines of code with a unit of *LOC* or, for large programs, *KLOC* (thousands of lines of code). Unfortunately, there is no consensus on exactly what constitutes a line of code. Most researchers agree that a blank line should not be counted but cannot agree on comments, declarations, null statements such as the Pascal "begin," etc. Another problem arises in free format languages which allow multiple statements on one textual line or one executable statement spread over more than one line of text.

For this study, the definition used is the following: A line of code is counted as the line or lines between semicolons, where intrinsic semicolons are assumed at both the beginning and the end of the source file. This specifically includes all lines containing executable and non-executable statements, program headers, and declarations.

Halstead's Software Science

A natural weighting scheme used by Halstead in his family of metrics (commonly called Software Science [HALM77]) is a count of the number of "tokens," which are units distinguishable by a compiler. All of Halstead's metrics are based on the following definitions:

n_1 = the number of unique operands.

n_2 = the number of unique operators.

N_1 = the total number of operands.

N_2 = the total number of operators.

Three of the software science metrics will be discussed, specifically N, V, and E .

The metric N is simply a count of the total number of tokens expressed as the number of operands plus the number of operators, i.e., $N = N_1 + N_2$. An operand is defined as a symbol used to represent data and an operator is any keyword or symbol used to express an action. The above definition sounds deceptively simple, but is in many cases ambiguous. For example, while the symbol "< =" should obviously be considered one operator, should the Pascal "BEGIN-END" surrounding a compound statement be considered one or two operators, or the keywords "WHILE-DO"? Should the Pascal reserved word "TYPE" be counted? Generally these ambiguities are resolved individually according to both language dependencies and convenience.

The second Halstead metric considered is volume (V). V represents the number of bits required to store the program in memory. Given n as the number of unique operators plus the number of unique operands, i.e., $n = n_1 + n_2$, then $\log_2(n)$ is the number of bits needed to encode every token in the program. Therefore, the number of bits necessary to store the entire program is:

$$V = N \times \log_2(n).$$

Obviously, since the measure V is dependent on N, it is subject to the same ambiguities as N. Studies have shown, however, that as long as there is consistency in choosing operators and operands, even a significant variation in classification rules has virtually no effect on results [ELSJ78].

The final Halstead metric examined is effort (E). The effort metric, which is used to indicate the effort of understanding, is dependent on the volume (V) and the difficulty (D). The difficulty is estimated as:

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2}.$$

Given V and D, the effort is calculated as:

$$E = V \times D.$$

The unit of measurement of E is elementary mental discriminations which represents the difficulty of making the mental comparisons required to implement the algorithm.

McCabe's Cyclomatic Complexity

McCabe's metric [MCCT76] is designed to indicate the testability and maintainability of a procedure by measuring the number of "linearly independent" paths through the program. To determine the paths, the procedure is represented as a strongly connected graph with one unique entry and exit point. The nodes are sequential blocks of code, and the edges are decisions causing a branch. The complexity is given by:

$$V(G) = E - N + 2 \quad \text{where}$$

E = the number of edges in the graph

N = the number of nodes in the graph.

According to McCabe, $V(G) = 10$ is a reasonable upper limit for the complexity of a single component of a program.

Constructing a tool for automatic metric generation is complicated if a graph needs to be generated. However, the method is greatly simplified if one realizes that since the same complexity may be generated by two completely different looking graphs, the measure is not dependent on how nodes are connected, but only on the number of edges and nodes in the graph. It is given that each node, except the final node, has at least one edge leading out of it. This can be represented by $E = N - 1 + R$, where the "-1" represents the fact that there is no edge leading out of the final node and R represents the extra edges created by the decision nodes in the graph. Then $E - N = R - 1$ by simple algebraic manipulation. This leads to the result:

$$V(G) = E - N + 2 = R + 1.$$

This result indicates that McCabe's Cyclomatic Complexity may be calculated by counting the number of simple decisions and adding one.

Counting decisions is a simple matter. One needs to merely count the occurrences of IF, DO, WHILE, CASE, and other conditional and loop control statements. This leaves only the problem of what to do with compound conditions (i.e., those using AND or OR). Since a statement of the form "IF A AND B THEN S" is functionally equivalent to "IF A THEN IF B THEN S," the AND and OR constructs should also be counted as decisions. Similarly, each predicate of a CASE statement should be counted since it is equivalent to multiple IF statements.

Therefore, the fairly complex problem of calculating McCabe's Cyclomatic Complexity using a graph has been reduced to the simple, easily automated process of counting the decisions in a component and adding one. The complexity of the entire program is simply the sum of the complexities for all the components.

Structure Metrics

It seems reasonable that a more complete measure will need to do more than simple counts of lines or tokens in order to fully capture the complexity of a module. This is due to the fact that within a program, there is a great deal of interaction between modules. Code metrics ignore these dependencies, implicitly assuming that each individual component of a program is a separate entity. Conversely, structure metrics attempt to quantify the module interactions using the assumption that the inter-dependencies involved contribute to the overall complexity of the program units, and ultimately to that of the entire program. In this study, the structure metric examined is Henry and Kafura's Information Flow metric.

Henry and Kafura's Information Flow Metric

Henry and Kafura [HENS79] [HENS81A] developed a metric based on the information flow connections between a procedure and its environment called "fan-in" and "fan-out" which are defined as:

fan-in the number of local flows into a procedure plus the number of global data structures from which a procedure retrieves information

fan-out the number of local flows from a procedure plus the number of global data structures which the procedure updates.

To calculate the fan-in and fan-out for a procedure, a set of relations is generated that reflects the flow of information through input parameters, global data structures and output parameters. From these relations, a flow structure is built that shows all possible program paths through which updates to each global data structure may propagate.

The complexity for a procedure is defined as:

$C_p = (fan\text{-}in \times fan\text{-}out)^2$ where

C_p is the complexity of procedure p.

fan-in is the number of fan-ins to procedure p.

fan-out is the number of fan-outs from procedure p.

The term "(*fan-in* × *fan-out*)" is squared to represent the idea that the complexity due to the inter-relationship between components is non-linear.

In addition to procedural complexity, the metric may be utilized for both a module and a level of the hierarchy of the system. A module with respect to a data structure D, consists of those procedures which either update D or retrieve information from D. Module complexity is defined as the sum of the complexities of the procedures in the module, and the level complexity is the sum of the complexities of the modules within the level.

Procedural complexity is used to find those procedures with heavy data traffic and those not adequately refined. Module complexity reveals overloaded data structures as well as improper modularization. The level complexity may be used to detect missing levels of abstraction or to compare alternate system designs.

Hybrid Metrics

Since, as stated above, code and structure metrics appear to be measuring different aspects of program complexity, it seems reasonable that a metric be comprised of both types of metrics in order to capture the complexity of the procedure as much as possible. This is what is termed a hybrid metric. More succinctly, a hybrid metric is composed of one or more micro metrics and one or more macro metrics. This study examines the hybrid Henry and Kafura's Information Flow metric.

Henry and Kafura's Information Flow Metric

Henry and Kafura's Information Flow metric may also be used as a hybrid metric. This is the method used in an actual study by Henry and Kafura on the UNIX operating system [HENS81a]. The formula in its hybrid form is:

$C_p = C_{ip} \times (fan\text{-}in \times fan\text{-}out)^2$ where

C_p is the complexity of procedure p.

C_{ip} is the internal complexity of procedure p.

fan-in is the number of fan-ins to procedure p.

fan-out is the number of fan-outs from procedure p.

The metric used for the internal complexity C_{ip} may be any micro metric.

Conclusion

The use of software quality metrics allows a quantitative analysis of the complexity of a procedure or group of procedures. This analysis makes it possible to determine if a procedure is "better" than another procedure which is its functional equivalent, where "better" in this sense is defined as less complex. This should allow the development of more reliable software which is a major goal of software engineers.

Many software quality metrics have been proposed, but they fall into three general groups: Code metrics, Structure metrics, and Hybrid metrics. Code metrics are comprised of counts of some aspect of the code being analyzed. Structure metrics attempt to determine inter-relationships between program elements, and hybrid metrics are a combination of one or more code metrics and one or more structure metrics.

Chapter 2 gives a brief overview of several design methodologies and existing Program Design Languages. In chapter 3 is a partial description of the design specification language ADLIF, pointing out various interesting aspects of the language. The process of data collection is described in chapter 4 along with a brief explanation of some aspects of the data manipulation necessary for this study. In chapter 5, results for the study when using all of the data together are given. Results on a project-by-project basis are discussed in chapter 6 along with a comparison of the metrics to each other. Chapter 7 contains the conclusions and future work necessary with the use of metrics during design.

Chapter 2 Design Methodology Review

Introduction

In the area of software design, the basic problem is one of control. Without some form of control, the design process can degenerate into mass confusion consisting of disjoint ideas and goals. To help alleviate this problem, many varied methodologies have been formulated that allow, and even force, software design to be a constructive and orderly process [BASV75] [DEMT82] [SHES81] [YOUE79]. In general, the design process consists of two separate processes. The first is general design, which encompasses such techniques as functional decomposition and top-down design, and the second is detailed design which includes flowcharts and pseudo code. A detailed design technique developed in the last few years involves the use of a Program Design Language, often called simply a PDL.

In order to understand the practicality of a PDL, it is informative to review the evolution of the structured design process. Design methodologies have ranged from flowcharts to functional decomposition to pseudo code, including PDLs. Each of these techniques is discussed in some small detail; other common methodologies are discussed as well.

General Design Techniques

Functional Decomposition

A major step in the evolution of the design process was functional decomposition. This entailed the breaking down of a system into smaller, more manageable components, where each component is a functionally independent entity. In this way, a large, unwieldy project could be reduced to a series of simpler functions. Functional decomposition is accepted as a partial solution to the design of more complex programs. Due to the rise in system sophistication, it was no longer possible for one person to design an entire project. Functional decomposition allowed many people to design small parts of the overall project. It seemed that at least the beginnings of a truly workable design methodology had been found.

Top-Down Design

The concept of a top-down design is very similar to that of functional decomposition. The technique is to start with the most general structure of the program and leave the specific details until later in the design process. Each individual detail is then treated in the same manner until each has been fully expanded. In this way, a structured hierarchy is formed that separates the original problem into several smaller problems. In a procedural language, each incremental expansion generally results in one or more procedure calls. When used in this fashion (i.e., to generate the calling structure), the technique is also known as hierarchical decomposition [ARTJ86]. The top-down design technique differs from functional decomposition in that the program is not divided by function, but by level of detail, where the most detailed sections are the last to be defined. Quite often, the two techniques (functional decomposition and top-down design), are combined by first

partitioning the design into functional units and then applying the top-down design methodology to each of the individual functional units.

Stepwise refinement

Within a component determined by functional decomposition, top-down design, or any other method of decomposing a program into a series of smaller problems, a useful design technique is stepwise refinement. The methodology is defined as: "a system development methodology in which data definitions and processing steps are defined broadly at first and then with increasing detail" [ARTJ85]. This is similar to top-down design, but at this stage, specific steps are defined as opposed to general components of the program. The overall idea, however, is the same. Start with a general concept, and through a series of iterations refine it to the necessary level.

Detailed Design

Flowcharts

The first practical attempt at structured design was the flowchart. A controversy exists over the use of flowcharting. They have been called everything from "an essential tool in problem solving" [BOHM71] to "an obsolete nuisance" [BROF75]. In either case, flowcharts were a useful testing ground for the concept of having some sort of design methodology. They were the first technique used that forced a designer to supply more than a natural language description of the problem and possible solutions. Flowcharts required a design to show more detailed thought into problems that were often left for the programmer to solve such as implementation and efficiency. Many people feel, however, that even though flowcharts were better than nothing, they still had a great number

of deficiencies, some of the greatest of which were the lack of flexibility left to the programmer as well as the confusion often caused by the notation. Also, flowcharts tend to place emphasis on explicit flow of control as opposed to overall program structure [RAMH83]. Because of this, several different design methodologies were developed in an attempt to simplify the design process, clarify the resultant design, and place more emphasis on program structure.

Pseudo Code

To solve these problems, designers began to write code-like designs wherever possible to avoid the ambiguities of the flowcharts. This technique become known as writing pseudo code. This type of design is similar to a programming language without the syntactic restrictions. Designing in this style helped to reduce the communication gap between the programmers and the designers. Since there was no structured form for pseudo code, each individual used his own style and self-imposed syntax. Therefore, it quickly became apparent that there must be some sort of a standardization in order to most efficiently use this technique.

PDL

By devising the concept of pseudo code with a specific syntax, the program design language came into existence. A program design language is different from a programming language since "while a programmer communicates with the computer, a designer communicates with people: managers, programmers, other designers, and himself" [SUTS81]. Due to this difference, the requirements for a PDL are somewhat different. A well-designed PDL should have a rich typing environment as well as an adequate control structure set. It must, however, not be too restrictive. A designer using

the language should have the ability to move outside the bounds of a stringent syntax and express himself in a natural language medium.

Other features desirable in a well-constructed PDL should include: the ability to use the language at all levels of design from rough descriptions to detailed pseudo code, adaptability to the environment, familiarity to both designers and programmers, and susceptibility to the use of automated tools such as parsers, automatic document generators, and metric generators. A PDL with these features is indispensable as a design tool.

Existing PDLs

Due to the rapidly growing number of program design languages, it is not feasible to discuss them all. Instead, six different types of PDLs have been chosen that are representative of the types of ideas and goals that have been attempted with the use of these powerful design tools. All of the languages discussed below are automated to some degree and are currently in use.

The first PDL discussed is PSL/PSA, developed at the University of Michigan [FAIR85]. PSL/PSA stands for problem statement language/problem statement analyzer, is probably the best known PDL. PSL is a computer processable language that is used to describe a system during the requirements phase of a program. That description consists of formal language statements supplemented with narrative descriptions. PSA processes PSL statements and creates an interface between the object-oriented database of information stored as a PSL description and the users. The main goals of PSL/PSA are those of communication and adequate documentation through the use of a great variety of reports generated from queries to the data base. A few of the possible reports that may be generated are: a Data Base Summary, Name Selection, Contents, and Program Structure.

A second PDL, which is based on the work done by the ISDOS project at the University of Michigan which developed PSL, is the Requirements and Development Language of RDL

[HEAH79]. RDL is a non-procedural language which is intended to be used throughout the life of a system from the specification of the program to its maintenance. Like PSL, RDL creates a database of information that can later be used to generate both documentation and numerous reports. The language is currently in use as in internal tool at Sperry Univac Software Development.

Another system is the Flex Software Design System which was developed at the Naval Research Laboratory in Washington, D.C. [SUTS81]. Flex is different from most program design languages in that it is really a PDL generator. An installation is given the Flex basic language which is then structured and enhanced with supplied tools to get the final product that is used by designers. The system generates information useful to several classes of people. For designers and programmers, Flex has the ability to automatically create documentation; for management, report generators may be added to Flex that display progress through the use of simple counting metrics kept by the system. The Flex system has features in common with modern programming languages such as being modular, extensible, and strongly typed, which should make it both simple to learn and useful in the translation process from design to code.

The fourth PDL discussed is called Structured Analysis, abbreviated as SA [ROSD77]. This PDL uses the concept of top-down design in a graphical presentation. The overall program is based on the idea of a blueprint design to efficiently and succinctly represent ideas. Because of the graphical capabilities of SA, it is suitable for use with any natural or artificial language. "Its scope is universal and unrestricted" [ROSD77]. SA is easy to learn and useful in that pictures are usually more easily understood when trying to grasp how system parts will eventually create a coherent whole. A version of SA is currently being used by SofTech, Inc. in Waltham, MD.

The next type of program design language is not truly a PDL at all, but a programming language, possibly with some extensions. One of these adaptations is the Specification Language Anna which is currently under development at Stanford University [LUCD85]. Anna is based on Ada [ADA 80] and stands for ANNotated Ada. This, of course, gives the designer the full power and attributes of the programming language, along with the associated tools in the Ada environment. Many people

may criticize the use of a programming language for design, but if the designers can resist the urge to write code instead of specifications, Ada may become an acceptable program design language, since it overcomes the difficulties of creating tools strictly for design and fills the gap between design and programming capabilities.

The last PDL discussed is the one developed for and used in this study: ADLIF. The acronym stands for Ada-like Design Language based on Information Flow. The goals of the language are to provide flexibility and translatability while maintaining enough structure to allow quantitative metrics to be generated. Most PDLs adapt readily to the use of code metrics, but to allow structure metrics to be generated, there must be at least a superficial resemblance to a programming language. Ada was chosen as that language with the belief that the use of Ada will become prevalent. An environment, including several tools, exists for ADLIF that eases some of the design problems. They include: a parser for consistency checking, a hierarchy chart generator to determine the overall system structure, and an analyzer that generates both code and structure metrics for the design. It is hoped that through the use of these tools and the proper use of ADLIF, that most of the advantages of other PDLs will be obtained. ADLIF is currently in use for the undergraduate, project-oriented Software Engineering course at Virginia Polytechnic Institute and State University.

PDLs and Metrics

The concept of a program design language as a tool for specifications and requirements has become widely accepted, but it is not enough by itself. The need for design metrics is being acknowledged by some software organizations since PDLs do not provide any concrete, quantitative basis for making important design decisions. Metrical analysis of software is informative, but many times "bad" code, according to the metrics, is caused by faulty decisions during design. It will be costly both in time and money to correct the design flaw. Ideally, the error should have been found much

earlier in the program life-cycle. Design metrics appear to be the answer to some problems of this type.

Unfortunately, design metrics have not been extensively tested. In fact, most papers state that a specific metric could be used on their design methodology, but do not explain or validate the idea. In some cases where testing has been done, it is unclear if the metrics were automatically generated or calculated by hand. Finally, with the exception of the analyzer used in this study, all the proposed tools generate only code metrics. Several examples of attempts at design metric validation are discussed below.

For an unspecified design methodology, D.A. Troy and S.H. Zweben hand generated some counting metrics such as number of operators, number of operands, etc. to find the Coupling and Cohesiveness and an unknown complexity measure. Also included in their study were determinations of modularity and size [TROD81].

Paul Szulewski, Mark Whitworth, Philip Buchan, and J. Barton DeWolf developed an automated tool with funding by the National Bureau of Standards' Institute of Computer Science at Draper Laboratory, Inc. Their tool uses uninterpreted Design Digraphs to calculate Halstead's Software Science metrics, McCabe's Cyclomatic Complexity, and some simple counting metrics [SZUP81]. The claim is forwarded that the tool is also useful for the testing and debugging of program specifications.

The third experiment is the one done for this study. An analyzer has been constructed (as discussed in Chapter 1) that can quickly be adapted to any procedural language. Of particular importance is the ability to analyze a design written using ADLIF. The use of the analyzer allows the calculation of both code and structural metrics at the design phase of a system. The metrics generated include the code metrics (procedure length, McCabe's Cyclomatic Complexity, and Halstead's Software Science) and Henry and Kafura's Information Flow metric in both its structure and hybrid form. Validation of ADLIF using this tool comprises the bulk of this study.

Conclusion

Once the need for a structured design methodology was recognized, many different techniques evolved including flowcharts, functional decomposition, pseudo code, and program design languages. The sophistication of each method increased to handle the growing complexity of computer systems. There is a growing popularity for the use of PDLs. Their similarity to programming languages combined with flexibility make them a seemingly ideal design tool.

Many different design languages have been proposed, but they can be roughly categorized into those with a database (such as PSL/PSA), those used solely for communication and documentation, those like FLEX that can be customized to fit the environment, PDLs that use a graphical representation similar to SA, programming languages (such as Ada) as PDLs, and a purely textual language like ADLIF which is similar to a programming language. Each of the above representations has advantages as well as inherent limitations.

A fairly common goal is to gain the ability to obtain quantitative measures of a design. To accomplish this goal, several groups have attempted to measure software quality of various designs. One of these attempts, and probably the most ambitious involves the use of the analyzer that is used in this research [HENS88].

Chapter 3 Definition of ADLIF

Introduction to ADLIF

ADLIF is an acronym for Ada-like Design Language based on Information Flow. Its primary purpose is to create an intermediate step between the general design and coding phases in the life cycle of a system. It is NOT intended to be used as a programming language.

Using ADLIF allows complexity measures to be taken before the coding phase of a system. In this way, the quality of the software may be improved at an early stage, reducing time, man-hours, and cost invested in the system.

ADLIF is structured similar to Ada to allow an easy transition between ADLIF and Ada. This of course, includes predecessors of Ada such as Pascal. It seems desirable to have the ability to translate a design to Ada quickly, since, in the near future, the Department of Defense is requiring the use of an Ada environment. Other software contractors will probably follow this requirement. Most of the constructs (including both control and data structures) supplied in ADLIF are related to similar constructs in Ada and Pascal. This should facilitate ease or proficiency with ADLIF for users familiar with either language.

Since ADLIF is a design language and not a programming language, many actual statements in the system may be omitted, or included in comment form instead of explicitly indicated. Therefore, in the most general sense, an ADLIF program may be mostly comments, specifically showing only variable declarations, subprograms with their parameters, subprogram calls with parameters, and updates of data structures. In some situations, it is more informative for a design to consist mostly of English sentences. At other times, in the most detailed use of ADLIF, pseudo instructions that are close to actual code are more desirable. ADLIF allows the designer to choose from either of these options, or anywhere in between. It seems reasonable that a more detailed (or code-like) design will result in more meaningful metric measurements. However, in most cases, the choice to design somewhere between English and code will be most useful, since a more detailed design will not enhance understandability. This implies that an innately complex piece of software will require a more code-like design reflecting the fact that the software is not readily understandable to the programmer. ADLIF contains most of the basic programming statements and these may be used at the designer's discretion for specific algorithms or formulas.

This chapter uses portions of the reference manual for the language ADLIF to discuss the more interesting or complex portions of the language. Questions regarding specific syntax are referred to the reference manual contained in Appendix A and to the Backus-Naur form (BNF) for the language located in Appendix B.

Comments

ADLIF contains three different types of comments. The first is the general comment typical of most program languages to allow user documentation, which is a string of characters preceded by two hyphens that continues until the end of the line, where an automatic termination of the comment occurs. This type of comment is used for general documentation.

```
-- THIS IS A COMMENT.
C := -- Comments are ignored when
A + B -- occuring any place in a program.
```

The second type of comment is a required comment, and is enclosed in parentheses and followed by a semicolon. Comments are required following the program statement and after a subprogram declaration. The intent is to ensure that each subprogram, as well as the overall design, is documented.

The final type of comment in ADLIF is one enclosed in braces ("{}"). This type of comment may be used in an expression as a replacement of a mathematical expression. For example: ((3 + X) DIV Y) may be replaced by such expressions as ({add 3 to X} DIV Y) or ({first add X and 3 and then integer divide that sum by Y}), where the comment constitutes either some or all of the expression. It enables the designer to leave the details of expression generation up to the programmer by allowing "pseudo" expressions to be substituted for an actual expression, when use of the comment will simplify or clarify some aspect of the design.

Comments are descriptive and may be used anywhere in a program except before the program declaration and following the "END program name." statement.

Built-in Functions and Procedures

Some built-in functions have been provided in ADLIF and may be assumed to exist. Since ADLIF is not a programming language, it is only necessary that the syntactic analyzer recognizes the existence of these built-in routines. These functions constitute the normal set of supplied functions in most programming languages. Their inclusion in ADLIF is simply to allow more design flexibility. They include: ROUND, TRUNC, INTTOREAL, EOLN, EOF, NEW, ORD, CHR,

SUCC, and PRED. For details on the parameters to the above supplied functions, as well as details on the meaning of each, see Appendix A.

Pointers

Many programming languages have mechanisms to implement dynamic storage allocation. ADLIF supports this concept with the Ada implementation of pointers. Due to the complexity of pointers (or access types) they are discussed in detail in this section and demonstrated with examples.

Access types are types for which objects may be dynamically allocated. If POINT has the type ACCESS INTEGER, then the declaration PTR_VAR : POINT creates a pointer (memory address) to an integer object (memory cell). The built in procedure NEW must be used to create the INTEGER access object itself. The statement PTR_VAR := NEW (INTEGER(3)); creates an integer object with the integer value 3 and assigns a pointer to that object.

Object declarations cause an object to come into existence when they are elaborated and to be deleted on exit from the scope in which they are declared. Access objects are created using NEW and they continue to exist as long as there is a path to the object from an access variable.

```
TYPE POINT = ACCESS INTEGER;
VAR PT1, PT2 : POINT;
PT1 := NEW (INTEGER (3));
PT2 := PT1;  -- PT1, PT2 share accessed object
```

Access types are generally used to point to records (even though pointers may access any type) and to form linked lists by inserting an access type into the record that points to the record type. This allows the ability to form a list of dynamic length. An object of type ACCESS to a record may be declared before the record declaration, but if the record does not appear anywhere in the type declarations, an error will result.

```
TYPE LINK = ACCESS ELEMENT;
TYPE ELEMENT =
 RECORD
  VALUE : INTEGER;
  NEXT : LINK;  -- Points to element
 ENDRECORD;
VAR EL1, EL2 : LINK;
EL1 := NEW (ELEMENT);
EL2 := NEW (ELEMENT);
EL1.NEXT := EL2;
```

Record fields may be accessed using pointers in the same way as a field in a record is accessed (i.e., in the example above, EL1.NEXT.VALUE refers to the VALUE field in the record following the one pointed to by EL1). This allows for a great deal of flexibility with data structures.

Parameter passage

In a procedure, parameters may have one of three different modes. The modes available are those defined in Ada. They allow the designer to determine how a parameter may be used within a procedure, either allowing a value to be only passed in (pass by value), passed out only, or both. In a procedure, the mode of a parameter occurs between the variable name and the variable type. For an example of mode usage, see Figure 4 on page 29.

When using a function a mode is not allowed, and the assumed mode is IN. This forces functions to return a single value using the function name. If the designer feels that it is required to use a mode other than the default of IN, the function must be transformed into a procedure. This follows the mathematical definition of a function.

The three parameter modes available for procedures and their meanings are:

IN

The parameter acts as a local constant whose value is provided by the corresponding actual parameter. If a mode is not specified, IN is assumed.

OUT

The parameter acts as a local variable whose value is assigned to the corresponding actual parameter as a result of the execution of the subprogram.

INOUT

The parameter acts as a local variable and permits access and assignment to the corresponding actual parameter. A value must be provided by the corresponding actual parameter, and a value is assigned to the corresponding actual parameter as a result of the execution of the subprogram.

Given that the program in Figure 4 on page 29 was translated into Ada, compiled and executed, the expected output would be:

```
1    3
10   11   12
1    11   12
```

```
PROGRAM Show_Mode (INPUT, OUTPUT);
(This program demonstrates the use of the parameter modes
 available in ADLIF.);

VAR First, Second, Third : INTEGER;

PROCEDURE Example (Num1 : IN INTEGER;
                   Num2 : OUT INTEGER;
                   Num3 : INOUT INTEGER);
(This procedure writes out the values of the
 parameters, changes them, and writes out their
 new values.);

BEGIN Example
  WRITELN (Num1, Num3);  -- Num2 may not be
                         -- printed out yet,
                         -- since it is undefined.
  Num1 := 10;
  Num2 := 11;
  Num3 := 12;
  WRITELN (Num1, Num2, Num3);
END Example;

BEGIN Show_Mode
  First := 1;
  Second := 2;
  Third := 3;
  Example (First, Second, Third);
  WRITELN (First, Second, Third);
END Show_Mode.
```

Figure 4. Parameter Modes

FOR Loop

The FOR loop in ADLIF is similar to the one in Ada. It is an attempt at making the FOR loop a structured construct. In other implementations of this type of loop (Pascal, for example), the loop control variable may be modified, or accessed from outside of the loop. This allows the possibility of adverse side effects. Much of this problem is solved by making the loop control variable local to the block of code that constitutes the FOR loop. Several restrictions on the use of the variable remove the rest of the problem. First, the loop control variable may only appear as a variable subscript, and second, the variable may only be passed as a parameter using the IN mode. These restrictions guarantee that only the loop itself may modify the value of the loop control variable. See below for examples on the use of the FOR loop.

The formal definition for the syntax of the FOR loop is:

```
FOR VARIABLE {IN or INREVERSE} Expr1, Expr2 LOOP
  statement[s];
ENDFOR;
```

Expr1 is the initial value of VARIABLE and Expr2 is the final value of VARIABLE. In the FOR statement the test for VARIABLE exceeding Expr2 is made prior to the execution of the loop statements. The test for comparing VARIABLE and Expr2 is as follows:

```
If IN and VARIABLE > Expr2 then
  terminate the loop.
If INREVERSE and VARIABLE < Expr2 then
  terminate the loop.
```

The execution of the FOR loop starts by elaborating the iteration clause, which acts as the declaration of the loop parameter. The identifier of the loop parameter is first introduced and the discrete range is then evaluated: the loop parameter is declared as a variable local to the loop

statement, whose type is that of the elements in the discrete range and whose range constraint is given by the discrete range. In other words, in the statement FOR LCV = 1, 10 LOOP the variable LCV is local to the loop and because the constants 1 and 10 are integers, LCV must be an integer. Inside the loop, LCV may only be used as a variable subscript or an IN parameter in a procedure call.

```
FOR LCV IN 1, 10 LOOP
 SUM := SUM + 1;
ENDFOR;
-- This loop is executed once
FOR LCV INREVERSE Min, Min LOOP
 TOTAL := SUM + INDEX;
 SUM := SUM + 1;
ENDFOR;
```

Packages

A facility called "Packages" is supplied in ADLIF in order to have the capability of modularizing a program design. Using a package, subprograms may be designed and used that are completely separate from any specific ADLIF program, thus allowing the use of the concept of information hiding. The overall form of a package is similar to that of an ADLIF program, and is given formally below:

```
PACKAGE package-name IS     -- This is the package
                            -- specification, and is
Visible-entities;           -- the only part of the
                            -- package visible to the
END package-name;           -- outside world.
PACKAGE BODY package-name IS
```

```
(comment);
[Declarative part;]          -- This is the body of the
                             -- package and contains
BEGIN package-name           -- all code related to the package.
statement[s];
END package-name;
```

In the specification above, *package-name* is any valid unique ADLIF name. *Visible-entities* are similar to a normal declarative part, with the exception that only those constants, types, variables, and subprogram declarations that are to be visible to the calling entity should be specified. Anything that is to be hidden to the calling entity should be omitted from this section. Note that procedures and functions not shown in the visible-entities section may only be used internally in the package. The *declarative part* is identical to that in an ADLIF program or subprogram, as are *statement[s]*. Note that subprogram declarations shown in the visible-entities section must be complete and identical to the declarations repeated in the declarative section in the body of the package.

In order to make the package specification visible to a program or another package, the reserved word USE is needed, and is placed as the first statement in the declarative section, with the following syntax:

```
USE package-name [,package-name];
```

Each declaration in the visible-entities section of the package must then be duplicated in the declarative section of the program using the package, with each entity followed by a package reference which indicates where the entity may be found. See below for the specific syntax of a package reference.

```
Visible-entity; PACKAGE package-name;
```

Each of the constants, types, variables, and subprograms from the package may be used or called in the normal manner. For a complete example of a package, see Figure 5 on page 33.

```
PACKAGE Manipulate IS      -- Specification

  PROCEDURE Push (Stack : INOUT Stacktype;
                  Elem : IN INTEGER);
  (This procedure implements the normal
   PUSH.);

  PROCEDURE Pop (Stack : Stacktype);
  (This procedure implements the normal
   POP function.);

END Manipulate;

PACKAGE BODY Manipulate IS      -- Body
(This package is the implementation of the
 stack operations PUSH and POP.);

  CONST Max = 100;

  TYPE Intrange = (0..Max);
  TYPE Stacktype = Array (1..Max) OF INTEGER;

  VAR Stack : Stacktype;
  VAR Top : Intrange;

  PROCEDURE Push (Stack : INOUT Stacktype;
                  Elem : IN INTEGER);
  (This procedure implements the normal PUSH.);

  BEGIN Push
    Top := Top + 1;
    Stack[Top] := Elem;
  END Push;

  PROCEDURE Pop (Stack : INTEGER);
  (This procedure implements the normal POP function.);

  BEGIN Pop
    Top := Top - 1;
  END Pop;

BEGIN Manipulate          -- Initialization
  Top := 0;
END Manipulate;
```

Figure 5. Example of Package Usage

Conclusion

This chapter discusses some of the more interesting or hard to understand constructs in ADLIF including comments, built-in functions, access types, parameter modes, the FOR loop, and packages. A full language description is given in Appendix A, and specifics about syntax may be found both there and in Appendix B, which contains the BNF for ADLIF.

Due to the similarity to Ada and Pascal, ADLIF will be both simple to learn and relatively easy to use. The use of programming constructs allows the designer the flexibility to decide how much of the design needs to be in pseudo code and how much may be in English sentences. Both the ease of use and the flexibility, along with the ability to do complexity measures early in the program life cycle, should make ADLIF a valuable design tool.

Chapter 4 Data Description

Introduction

Research, such as that done for this study, is highly dependent on the availability of relevant data to analyze. Studies that are based on small test programs (less than one thousand lines of code) may be considered suspect in that the data itself is unrealistic. On the other end of the spectrum are large-scale programs (more than ten thousand lines of code) which can become difficult to handle, both in terms of availability of the design and the corresponding source, and in managing the analysis involved with large amounts of data. For this study, an attempt has been made to collect data that has been developed in a realistic environment that is both manageable and non-trivial. Finally, a large amount of effort has been made to see that the integrity of the data is not compromised when preparing the data for statistical analysis.

In general, programs produced by students and used as data are not very well accepted. It is felt that the programming environment is unrealistic. For this reason, studies using student generated data are not as quickly accepted as those using data collected outside of academia.

Due to the problem with obtaining "real world" data, the data used for this study is student generated, but there is a major difference between the data used here and that generally produced in academia. The programming is done in an environment that is as realistic as possible. Therefore, it is felt that even though the programs used are produced within an academic environment, they are substantially more realistic than that generally obtained under these conditions.

Data Collection

Over the past six years, ADLIF designs and the resultant Pascal source code has been collected from undergraduate senior-level software engineering courses at both Virginia Polytechnic University and the University of Wisconsin—LaCrosse. The class is designed to teach students the basics of software engineering including, but not limited to, Design Languages, Software Design Methodologies, Software Quality Complexity Metrics, and Testing Methodologies. Assignments include a project developed using techniques covered in the course [HENS83].

The goal of the project is to expose students to the experience of non-trivial program development in a "real world" environment where designing and implementing a project is not a single-person task. To this end, the class is divided into teams of three people. Each team is responsible for designing a system that upon completion will be from three thousand to five thousand lines of source code. This design includes hierarchy charts and module specifications. After a team has completed the specifications for their project, they "hire" classmates to implement the modules. Finally, the team must integrate the modules into the completed program.

The completed projects that have been furnished by the classes are varied. They range from two thousand to eight thousand lines of code, indicating the students' inability to accurately predict program size. Program function is also widespread. Due to the fact that the experience of a real-world environment is the important goal of the assignment and not final program function, many

groups have developed games such as chess, Othello, backgammon, and Monopoly. Not all of the teams, however, chose this route. Other types of projects have been developed including a hierarchy chart generator, a Pascal code formatter, a calendar system, and a bulletin board.

In an attempt to ensure that the completed designs and source are usable as data, students are given minimal design requirements. The requirements are that the students should include in their specifications:

1. Procedure and function calls.
2. Updates to global data structures.
3. Parameters necessary to each procedure.

Some tools have been developed to aid students in the specification portion of the project. One tool is an ADLIF hierarchy chart generator which allows the design teams and programmers to get an overall feel for the structure of the program and to more easily determine problems inherent with the calling hierarchy. Another tool that has been developed is an ADLIF syntax analyzer which is useful in eliminating common errors (such as typing errors), as well as determining missing components (such as parameters, variable declarations, etc). The syntax analyzer also ensures that the projects, which are used as data, are in an analyzable form. Since the completion of the software metric analyzer used for this study, students have been allowed to use it to generate design metrics which can be examined to indicate possible trouble spots in the design, such as inadequate refinement, poorly designed data structures, and missing levels of abstraction.

Data Preparation

After the ADLIF and Pascal code have been processed by the analyzer and metrics generated, it is necessary to manipulate the data to some extent. First, the metrics for system calls, as well as a few

other specialized procedures, must be removed and then procedures must be combined into modules. Justification for this manipulation and the methods used are explained in the following text.

Once metric generation is complete, the values attributed to system calls and certain specialized procedures that do not contribute useful information should be removed. In terms of system routines, this is due to the fact that the metric values for a procedure invoking a system routine are resolved during analysis, and the complexity for the system call itself is extraneous since the actual code for the routine cannot be considered. Procedure values not pertinent to the overall complexity are those of routines that are generally considered utilities, such as graphics routines, number conversion routines, etc. The reason that the complexity values for these procedures are not useful is that routines of this type are similar to operators. They are invoked from many different places in the program and therefore have extremely high complexity. Due to the large number of calls to these routines, the structure metric values in this case are not a true reflection of the actual complexity of the procedure. These procedures are similar to the "memoryless" procedures defined in [HENS81a].

The other manipulation required is that of combining the procedures into modules. In order to compare the ADLIF procedural complexity to that of the Pascal, there must be a one-to-one correspondence between the design and source. The method used is simple. A single ADLIF procedure may be a definition of the function performed by several Pascal procedures. In order to equate the design and source, it is necessary to add the complexities of each of the Pascal routines. In this way, the design complexity is directly comparable to that of the source complexity on a functional level. For example: Given a procedure A in ADLIF with a specific function, in the Pascal code the same function is accomplished by the procedure P, which also calls the routines P_1 and P_2. In order to compare the design and source complexities of the function, it is necessary to add the metric values of P_1 and P_2 to that of P. After the addition, it is possible to directly compare the metrics for procedures A and P, since they now represent the complexities of the same function in ADLIF and Pascal respectively.

Conclusion

When attempting to validate a new methodology, it is important to gather relevant data. Relevant in this sense means that the programs used should be non-trivial in length and developed in a realistic environment that emulates, as much as possible, a typical software development process in the "real world." Of course, the ideal situation is to actually obtain software produced outside of academia, but since this can be a difficult task, the next best choice is that of programs produced in the most realistic environment possible. Under all of the above criteria, the data for this study is indeed useful. It has been obtained over a six-year period from the project assignments in the undergraduate software engineering class. Program lengths range from three thousand to eight thousand lines. In addition, the programs have been developed in a somewhat realistic environment where design teams put together the specifications and then the programmers, represented by other students, implement the design. In addition, care is taken to ensure that pre-statistical manipulation does not impair the quality of the data. Complexity values are only removed from the raw data when their removal is indicated by strictly defined criteria. Finally, the routines from both ADLIF and Pascal are placed into modules that form a one-to-one correspondence between the design and source procedures.

Chapter 5 Analysis of Project Data

Introduction

This chapter presents the statistical analysis of the data on a project-by-project basis. The analysis consists of the correlation values for all metrics across the 27 projects used as data in the study. The raw data used may be found in Appendix C. Note that in the appendix, observations that are preceded by a "*" are the ones that remain after outlier elimination. The chapter concludes with a discussion of the results and interpretation of possible meanings. The abbreviations for the metrics used throughout this and the following chapters are presented in Table 1 on page 41.

Statistical Analysis

The first step in the statistical analysis of the data is the elimination of outliers. In order to effectively eliminate invalid data while maintaining the overall integrity of the data, there are two major considerations. First is the effect of a change in the design that is not reflected in the given specifications but does appear in the resultant source code, and second is when specifications are not

Table 1. Metric Abbreviations Used in Data Presentation

Metric	Abbreviation
Length	LOC
N	N
Volume	V
Effort	E
Cyclomatic Complexity	CC
Information Flow	INFO
Information Flow with Length	INFO-L
Information Flow with Effort	INFO-E
Information Flow with Cyclomatic	INFO-CC

developed at a high enough level to make measurements meaningful. The first is evident when the complexity of the specifications for a procedure is larger than that of the source. It seems obvious that the complexity of the design should always be less than or, at best, equal to that of the source. When this is not the case, a design modification must have been made that does not appear in the source code. So as to not eliminate cases where the specifications are only slightly larger, only cases where the Information Flow complexity of the design is more than twice that of the source are eliminated. The second case appears when the source complexity is substantially larger than that of the specifications. For purposes of this study, "substantially" is defined to be source Information Flow complexity more than one hundred times greater than the values for the design. After elimination of outliers using the above criteria, there are 981 procedures left to analyze from 27 projects. Table 2 on page 43 shows the correlations of ADLIF to Pascal for all of the projects and metrics after the outlier elimination.

It is informative to note that in general, the Information Flow Metric in both its structure and hybrid form performs better than the micro metrics. In fact, nine of the projects (2, 7, 11, 18, 19, 23, 24, 26, 27) have substantially lower correlations for the micro metrics. A reason for this phenomena is that the specifications are not well refined, but include those elements necessary to the Information Flow Metric including procedure declarations, calls, and updates to global variables. Project 23 is a good example of this general category of projects. The specifications for this project consist almost exclusively of those elements listed above. This type of design gives no clues as to the final form of the procedure. Therefore, it is impossible to predict the procedure complexity with the micro metrics. There simply is not enough information in the design.

Seven of the projects (3, 8, 10, 14, 15, 21, 22) have poor correlation values for all of the metrics. This is due to poor design. The designers did not consistently include any detail in most of the procedures. Not enough detail is provided to predict the micro metric values, and the elements necessary for Information Flow are not available. In fact, within the above named projects, most of the ADLIF procedures closely resemble the one shown in Figure 6 on page 45. Unfortunately, this is syntactically correct ADLIF code even though the semantic value is practically nonexistent.

Table 2. Correlations for all Projects Using all Nine Metrics

Project	*Correlations*								
NUM	**LOC**	**N**	**V**	**E**	**CC**	**INFO**	**INFO-L**	**INFO-E**	**INFO-CC**
1	0.781	0.828	0.891	0.753	0.670	0.940	0.992	0.981	0.829
2	0.325	0.335	0.412	0.117	0.524	1.000	1.000	0.999	0.999
3	0.364	0.387	0.357	0.019	0.390	0.523	0.254	-0.005	0.508
4	0.931	0.868	0.922	0.777	0.117	0.999	1.000	1.000	0.951
5	0.913	0.794	0.829	0.829	0.707	0.992	1.000	0.992	0.977
6	0.876	0.889	0.926	0.901	0.057	0.982	1.000	1.000	0.995
7	0.151	0.092	0.075	0.033	0.008	0.507	0.726	0.696	0.764
8	-0.059	-0.014	-0.015	-0.098	0.220	0.238	0.135	-0.035	-0.008
9	0.700	0.668	0.650	0.717	0.339	0.785	0.801	0.700	0.938
10	0.493	0.456	0.489	0.765	0.149	0.436	0.143	0.138	0.004
11	0.313	0.171	0.192	-0.040	0.631	0.954	0.827	0.106	0.959
12	0.593	0.511	0.557	0.007	0.421	1.000	1.000	1.000	1.000
13	-0.797	-0.376	-0.562	-0.998	-0.500	0.120	-0.183	0.646	-0.196
14	-0.075	-0.019	0.034	-0.056	0.053	0.306	-0.043	-0.088	-0.006
15	0.275	0.126	0.151	0.014	0.336	0.255	0.236	0.095	0.170
16	0.683	0.813	0.824	0.575	0.554	0.912	0.927	0.964	0.808
17	0.804	0.756	0.767	0.475	0.813	0.593	0.606	0.728	0.753
18	-0.135	-0.254	-0.162	-0.062	-0.120	0.639	0.718	0.848	0.952
19	0.324	0.329	0.301	0.091	0.218	0.766	0.326	0.696	0.562
20	0.799	0.681	0.722	0.635	NA	1.000	1.000	1.000	1.000
21	0.383	0.481	0.511	0.536	0.610	0.428	0.483	0.877	0.704
22	0.216	0.190	0.313	0.563	0.435	0.371	0.304	0.072	-0.017
23	0.621	0.548	0.615	0.202	0.573	0.997	0.998	0.984	0.907
24	0.348	0.595	0.764	0.548	0.574	0.837	0.789	0.687	0.855
25	0.800	0.897	0.918	0.946	0.857	0.717	0.741	0.841	0.828
26	0.177	0.340	0.341	0.370	0.392	0.704	0.839	0.892	0.907
27	-0.244	-0.281	-0.275	-0.172	-0.149	0.854	0.456	0.357	0.539

```
PROCEDURE Example;
(This is an example of a poorly specified procedure.);
BEGIN Example;
-- A comment explaining the function of the procedure.
END Example;
```

Figure 6. Example of a Poorly Specified Procedure

Two rows of Table 2 on page 43 that are of particular interest are rows 13 and 20. Row 13 contains values that appear to show an extremely high negative correlation. Obviously this is an undesirable result. Upon inspection, the reason for the unusual correlations becomes obvious. After outlier elimination, there are only three routines left to analyze. Project 13 is one of those that is not specified very well and really belongs to the group of procedures with all poor values. The small amount of useable data simply distorted the results. Row 20 is of interest because one of its entries in the table is "NA." This is due to the fact that after outlier elimination, all of the McCabe values in the design that are left have a complexity of 1. It is not possible to do correlations on a constant vector, and therefore that correlation is "Not Available."

Conclusion

The simple use of correlations on a project-by-project basis is not very informative as far as the ability to project source code complexity is concerned. Many of the design teams typify a problem in software development. Given a choice between doing a job well, and doing the same job with as little effort as possible on their part, they choose the latter. This problem reinforces the belief that the specification phase of a project must be closely monitored to ensure that a useful design is the end result.

Since the results of the above correlations give so little useful information, it is informative to look at the data as a single entity as opposed to 27 different segments. It is hoped that by doing so, the projects that are not well specified will be offset by those that are well done. Chapter 6 gives the results of this analysis.

Chapter 6 Analysis of Combined Data

Introduction

Since a goal of this study is to generate regression lines for the individual metrics, and each of the projects has slightly different regression values, it is informative to examine the data as a whole in order to develop a single predictor for each metric. This chapter examines the data in its entirety giving correlations and regressions as well as plots showing the confidence intervals of the predictors. At the completion of the chapter, inter-metric correlations are examined.

Analysis Methods

After outlier elimination, which is the same as that used in Chapter 5, the analysis consists of correlations to determine the overall trends of the data, regression analysis to develop predictors, and confidence interval plots to give a feel for the error associated with the predictors. The correlations, regressions, and plots use straight forward statistical methods.

The regression used in this portion of the study is of the form:

$Y = aX + b$ where

Y is the dependent variable or source complexity

X is the independent variable or design complexity

a is the slope for the regression line

b is the intercept for the regression line

Comparison of all Procedure Complexities

Table 3 on page 48 contains the correlations for all design procedures to all the source procedures after outlier removal, for all nine metrics. It is interesting to note that in all cases, the Information Flow metric in both its structure and hybrid forms has a correlation greater than 0.74, while the micro metric correlations are all less than 0.56. It may be surmised that a reason that the micro metrics do not perform as well is that they are more dependent on the level of refinement of the specifications. In addition, since the structure and hybrid metrics did so well, there is some indication that they are independent of the level of refinement. The high values for the structure and hybrid metrics indicate that they are probably going to be useful as design measures, and that the language ADLIF is a worthwhile design tool.

The regression lines generated using the form of regression discussed above are given in Table 4 on page 49. Predictor equations may be derived from this regression data in the following way using the *coef*'s from the table:

$Y = Slope \times X + Intercept$ where

Y is the dependent variable or source complexity

X is the independent variable or design complexity

Slope is the slope for the regression line

Intercept is the intercept for the regression line

Table 3. Correlations of all Design Procedures to all Source Procedures

LOC	N	V	E	CC	INFO	INFO-L	INFO-E	INFO-CC
0.551	0.473	0.516	0.329	0.344	0.748	0.805	0.750	0.757

Table 4. Regression Line Equations and Statistics

Regression Line Information				
		Coef	Std Err	t-Value
LOC	Intercept Slope	10.267 0.997	0.985 0.048	10.426 20.647
N	Intercept Slope	98.439 0.956	7.643 0.057	12.879 16.792
V	Intercept Slope	505.403 1.014	42.096 0.054	12.006 18.828
E	Intercept Slope	40740.293 0.817	5155.005 0.075	7.903 10.895
CC	Intercept Slope	4.476 0.722	0.331 0.063	13.538 11.461
INFO	Intercept Slope	-400652.310 34.614	450265.530 0.982	-0.890 35.264
INFO-L	Intercept Slope	15409537.590 14.158	13068221.200 0.339	1.179 41.789
INFO-E	Intercept Slope	4393813873 37.457	24583769195 1.056	0.179 35.455
INFO-CC	Intercept Slope	-1185738.600 87.070	3241771.460 2.349	-0.366 36.220

Therefore, a predictor equation for the LOC metric would be:

$$Y = 0.997 \times X + 10.267$$

Using this method, it is possible to measure the complexity of the design and predict what the source complexity will be with a 95% degree of accuracy for each of the metrics. The other columns of Table 4 on page 49 represent the standard error (*Std*Err) and the value from the Student t distribution (*t-Value*). The standard error term indicates that 95% of the data will fall within twice the standard error centered around the regression line. The *t-Value* indicates the level of confidence that the associated *Coef* is non-zero. In order to state that a coefficient is definitely non-zero, the *t-Value* must be greater than two for the data used in this study.

In addition to the prediction of the source complexity, it would be convenient if an estimate of the amount of error involved in the prediction could be calculated. To this end, some extra statistics are given in Table 5 on page 51. Given a specific design complexity for a metric, these statistics may be used to calculate a 95% confidence interval for a predicted source complexity using the equation:

$$Slope \times X + Intercept \pm 2 \times \sqrt{MSE \times (1/n + (X - X_{Mean})/SS_x)}$$ where

X is the independent variable or design complexity

Slope is the slope for the regression line

Intercept is the intercept for the regression line

MSE is the Mean Squared Error from the model

n is the total number of data points employed in the regression

X_{Mean} is the average value of the design complexities

SS_X is the sum of squares over X and is given by:

$$SS_X = \sum_{i=1}^{n} (X_i - X_{Mean})^2$$

[WALR78]

Table 5. Extra Statistics

Metric	*Statistics*			
	n	*MSE*	SS_x	X_{Mean}
LOC	**981**	**2.468E+05**	**2.482E+05**	**12.754**
N	**981**	**1.037E+07**	**1.134E+07**	**80.379**
V	**981**	**4.500E+08**	**4.373E+08**	**405.837**
E	**981**	**2.973E+12**	**4.452E+12**	**13616.830**
CC	**981**	**9.312E+03**	**1.785E+04**	**3.053**
INFO	**981**	**2.455E+17**	**2.049E+14**	**39205.830**
INFO-L	**981**	**2.915E+20**	**1.454E+18**	**2.376E+6**
INFO-E	**981**	**7.412E+26**	**5.283E+23**	**1.725E+9**
INFO-CC	**981**	**1.340E+19**	**1.852E+15**	**1.316E+5**

Using the above equation and the prediction line for LOC as given above, the 95% confidence interval for a design complexity of 100 is given by :

$$0.997 \times 100 + 10.267 \pm 2 \times \sqrt{2.468E+5 \times (1/981 + (100 - 12.754)/2.482E+5)}$$

$$109.967 \pm 2 \times \sqrt{2.468E+5 \times (0.001 + 87.246/2.482E+5)}$$

$$109.967 \pm 2 \times \sqrt{2.468E+5 \times (0.001 + 3.515E-4)}$$

$$109.967 \pm 2 \times \sqrt{2.468E+5 \times 0.0013515}$$

$$109.967 \pm 2 \times \sqrt{333.550}$$

$$109.967 \pm 2 \times 18.263$$

$$109.967 \pm 36.526$$

In other words, the actual value associated with the complexity 100 for the metric LOC will be in the range 73.441 — 146.493 95% of the time.

Figure 7 on page 53 and Figure 8 on page 54 are examples of the plots where the middle line is the regression line, and the outer curves are 95% confidence intervals (i.e., 95% of the data will be in between the confidence curves). Plots for all of the metrics may be found in Appendix D. Note that because of the relative size of the Information Flow values, it is simpler to examine the log of the complexity values. All plots using the Information Flow metric use the $\log_{10}$ of both the design and source complexity values, but the regression analysis is done with the raw data values.

Metric Comparison by Level of Refinement

As mentioned above, it is informative to examine the metric values based on the level of refinement of the specifications. In order to accomplish this purpose, it is necessary to determine the refinement level for each routine to be examined. It seems reasonable that as the refinement level increases, the length of the procedure will also increase, but more importantly, the number of control structures will increase as well. Also, it appears necessary to consider *normalized* complexity values

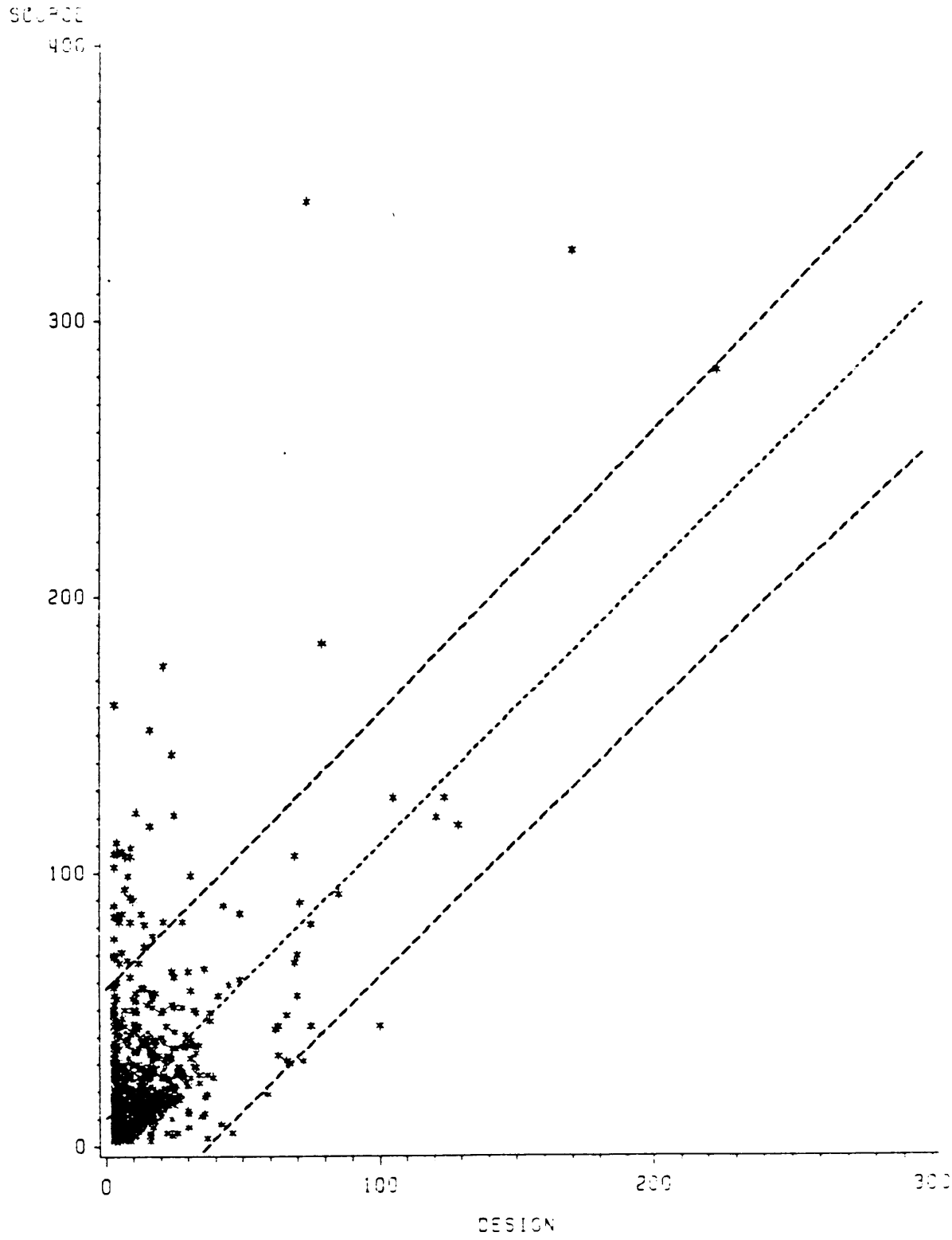

Figure 7. Regression and 95% Confidence lines for LOC

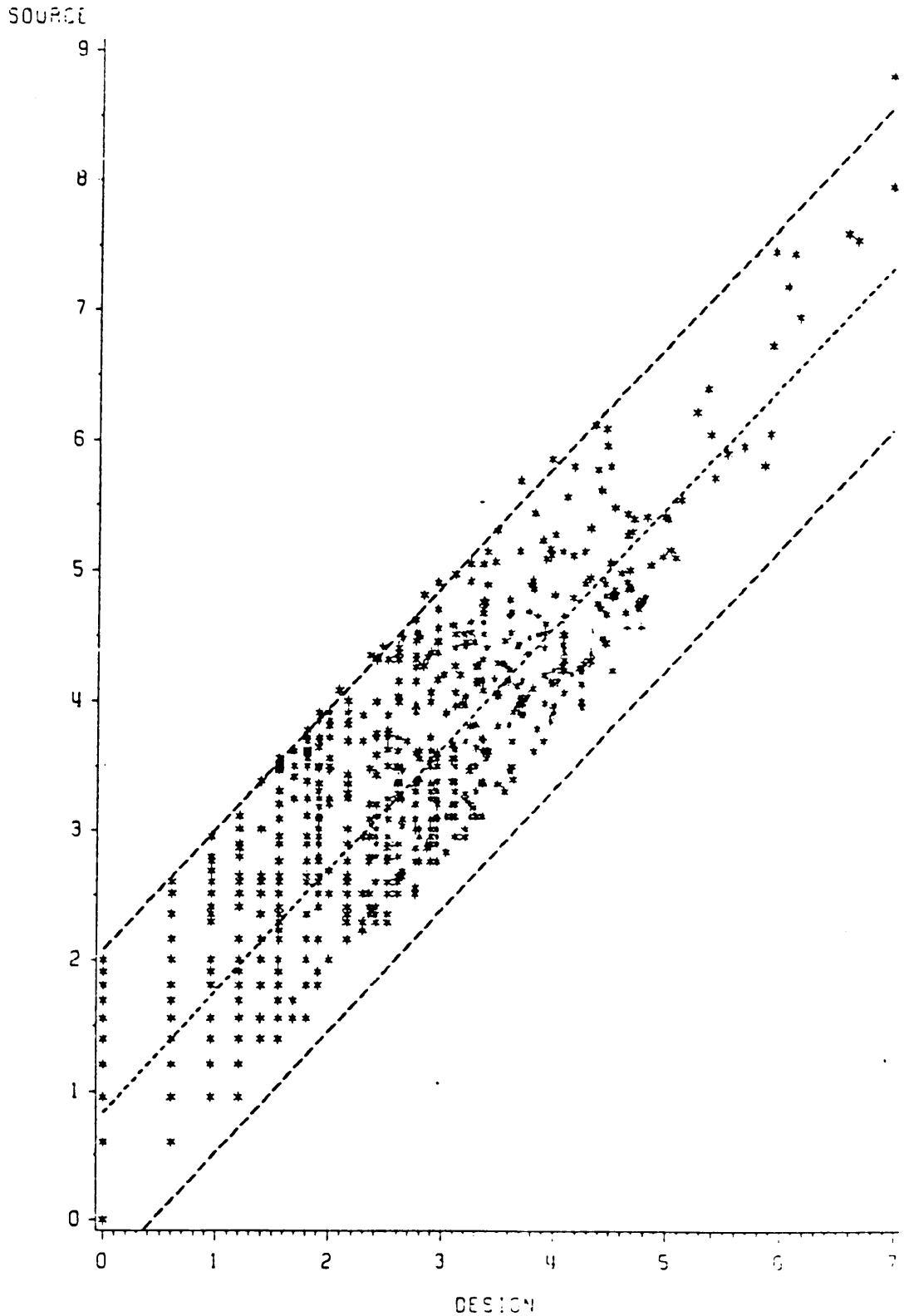

Figure 8. Regression and 95% Confidence lines for INFO

as opposed to examining the raw data, since, for example, the semantic difference between 1 and 5 is greater than that between 41 and 45. The normalizing function used in this study is:

$$Metric_{Norm} = \frac{(Metric_{Source} - Metric_{Design})}{(Metric_{Source} + Metric_{Design})}.$$

Note that the function results in a number between zero and one, and that the the value approaches zero as the design metric value approaches the source metric value.

Using these ideas, an algorithm to automatically determine the completeness of the design is given in Figure 9 on page 56 through Figure 11 on page 58. The code for the algorithm is written in S which is a UNIX-based statistical package [BECR84]. It is interesting to note that the algorithm is completely dependent on micro metric values, yet, as will be demonstrated, the Information Flow metric performs better than the micro metrics. This again indicates that structure metrics are probably independent of refinement level.

In English, the algorithm procedes as follows: First, in Figure 9 on page 56, a normalized value is calculated for the McCabe and N metrics as explained previously for each procedure. In the next step of the algorithm, shown in Figure 10 on page 57, the level of refinement is determined using the normalized values. If the CC_{Norm} value is 0.0, then it can not be used to determine level, and the metric N is the sole determiner. Otherwise, the CC_{Norm} value is checked first, and if it appears fully refined, the level is either High or Middle, depending on N_{Norm}. If the CC_{Norm} value is of Middle refinement, the level is either Middle or Low, again depending on the value of N_{Norm}. Finally, if CC_{Norm} indicates a low level of refinement, the procedure is determined to be Low. The last step in the algorithm, appearing in Figure 11 on page 58, performs the actual regression at each level of refinement.

In order to determine the validity of the results of the algorithm, seven projects were examined and procedural refinement determined by hand. Comparison of the results of the hand-generated levels and those determined by the algorithm indicate that this method of setting a level of refinement

```
MACRO allnorm (design, source, ccdesign, ccsource, ndesign, nsource,
          botref, midref, topref)
ccnorm <- NULL
nnorm <- NULL
#
# Normalize the CC and N values for all routines.
#
for (lcv in 1 to nrow (ccdesign)) {
    #
    # Normalize CC values.
    #
    ccnorm <- rbind (ccnorm, (round ( ((ccsource[lcv] - ccdesign[lcv]) /
                                (ccsource[lcv] + ccdesign[lcv])) ,3))
    #
    # Normalize N values.
    #
    nnorm <- rbind (nnorm, (round ( ((nsource[lcv] - ndesign[lcv]) /
                               (nsource[lcv] + ndesign[lcv])) ,3))
    }
```

Figure 9. Refinement Level Algorithm

```
topref <- NULL
midref <- NULL
botref <- NULL
#
# Examine the normalized values to determine level of
# refinement.
#
for (lcv in 1 to nrow (ccnorm))
    #
    # If the normalized CC value is zero, then the
    # metric is useless for determining level.  Use N.
    #
    if (ccnorm[lcv] = = 0.0)
        #
        # Using N to determine level.
        #
        if (nnorm[lcv] < = 0.333)
            topref <- rbind (topref, lcv) else
            if (nnorm[lcv] < = 0.667)
                midref <- rbind (midref, lcv) else
                botref <- rbind (botref, lcv) else
        #
        # CC is non-zero.  Give it priority by checking it first.
        #
        if (ccnorm[lcv] < = 0.333)
            #
            # CC indicates level greater than low.  Determine
            # if it is high or middle using N.
            #
            if (nnorm[lcv] < = 0.333)
                topref <- rbind (topref, lcv) else
                midref <- rbind (midref, lcv) else
            #
            # CC indicates level can't be high.  Determine
            # if it is middle or low using N.
            #
            if (ccnorm[lcv] < = 0.667)
                if (nnorm[lcv] < = 0.667)
                    midref <- rbind (midref, lcv) else
                    botref <- rbind (botref, lcv) else
                #
                # CC indicates level is low.
                #
                botref <- rbind (botref, lcv)
```

Figure 10. Refinement Level Algorithm continued...

```
#
# Examine regression for procedures at low level.
#
vec  <- botref
dtmp <- design [vec,]
atmp <- add2100[vec,]
stmp <- source [vec,]
#
# Do regression for all micro metrics and information flow.
#
for (lcv in 1 to 6) {
    tmpa <- cbind (dtmp[,lcv], atmp)
    regress (tmpa, stmp[,lcv])
    }
#
# Examine regression for procedures at middle level.
#
vec  <- midref
dtmp <- design [vec,]
atmp <- add2100[vec,]
stmp <- source [vec,]
#
# Do regression for all micro metrics and information flow.
#
for (lcv in 1 to 6) {
    tmpa <- cbind (dtmp[,lcv], atmp)
    regress (tmpa, stmp[,lcv])
    }
#
# Examine regression for procedures at high level.
#
vec  <- topref
dtmp <- design [vec,]
atmp <- add2100[vec,]
stmp <- source [vec,]
#
# Do regression for all micro metrics and information flow.
#
for (lcv in 1 to 6) {
    tmpa <- cbind (dtmp[,lcv], atmp)
    regress (tmpa, stmp[,lcv])
    }
END
```

Figure 11. Refinement Level Algorithm continued ...

was extremely accurate, being correct more than 99% of the time. Due to the strong results of the algorithm, it was used to generate the refinement level of the procedures for all twenty seven projects. The results of the algorithm indicate that of the 981 procedures, 422 have a High level of refinement, 283 have a Middle level of refinement, and 276 of the procedures have a Low level of refinement.

Due to the complexity values generated for all of the data, regardless of level of refinement, it was expected that as completeness of the specifications increased, the correlation values for the micro metrics would increase as well. Also expected was that the Information Flow metric correlations would be fairly consistent, since the high value when using all of the data together seemed to indicate that structure metrics are independent of refinement level.

The analysis results, shown in Table 6 on page 60, show that the above expectations are generally true. The Information Flow metric performed consistently well, with a low correlation value of 0.792. This result shows explicitly that the metric is independent of refinement level. The micro metrics reacted as predicted. They do not perform well at a low level of refinement, and their correlations increase as level of refinement becomes greater. An interesting result is that the metrics perform quite well at even a middle refinement level. This indicates that after a minimum amount of detail is included in the specifications, the micro metrics become useful measures.

Table 7 on page 61 through Table 9 on page 63show the regression line equations and statistics for each of the metrics. As discussed above, the information in these tables may be used to form a prediction equation given that the level of refinement can be determined, again with a 95% degree of confidence. Table 10 on page 64 through Table 12 on page 66 give the extra statistics needed to predict a confidence interval for a specific design complexity value and metric using the technique discussed above.

The predictors that result for the micro metrics are more useful than those using the combined data above since more information about the lines is known (i.e., the level of refinement). To reflect this

Table 6. Correlations by Level of Refinement

Low Level						
	LOC	**N**	**V**	**E**	**CC**	**INFO**
	0.610	0.512	0.552	0.211	0.405	0.904

Middle Level						
	LOC	**N**	**V**	**E**	**CC**	**INFO**
	0.701	0.731	0.756	0.577	0.867	0.904

High Level						
	LOC	**N**	**V**	**E**	**CC**	**INFO**
	0.810	0.830	0.807	0.706	0.902	0.792

Table 7. Low Refinement Regression Line Equations and Statistics

Regression Line Information				
		Coef	Std Err	t-Value
LOC	Intercept Slope	29.224 1.126	2.236 0.088	13.069 12.737
N	Intercept Slope	226.137 1.410	18.032 0.143	12.541 9.877
V	Intercept Slope	1235.605 1.396	104.910 0.127	11.778 10.961
E	Intercept Slope	90760.893 2.570	16309.020 0.719	5.565 3.573
CC	Intercept Slope	3.630 6.117	1.286 0.833	2.823 7.341
INFO	Intercept Slope	81740.175 10.022	105225.480 0.286	0.777 35.106

Table 8. Middle Refinement Regression Line Equations and Statistics

Regression Line Information				
		Coef	Std Err	t-Value
LOC	Intercept	4.900	1.241	3.948
	Slope	1.242	0.075	16.498
N	Intercept	49.247	8.553	5.758
	Slope	1.440	0.080	17.937
V	Intercept	285.608	43.100	6.627
	Slope	1.408	0.073	19.356
E	Intercept	24147.773	4525.805	5.336
	Slope	1.459	0.123	11.851
CC	Intercept	0.789	0.259	3.045
	Slope	1.932	0.066	29.219
INFO	Intercept	-7258.697	14693.092	-0.494
	Slope	5.231	0.148	35.447

Table 9. High Refinement Regression Line Equations and Statistics

Regression Line Information				
		Coef	Std Err	t-Value
LOC	Intercept Slope	1.284 0.810	0.547 0.029	2.348 28.304
N	Intercept Slope	20.210 0.842	4.262 0.028	4.741 30.488
V	Intercept Slope	110.247 0.830	25.482 0.030	4.327 28.002
E	Intercept Slope	9522.419 0.756	3647.289 0.037	2.611 20.458
CC	Intercept Slope	0.644 0.791	0.133 0.018	4.829 42.840
INFO	Intercept Slope	-297874.630 40.602	958329.640 1.527	-0.311 26.582

Table 10. Extra Statistics for Low Refinement Level

Metric	*Statistics*			
	n	*MSE*	SS_x	X_{Mean}
LOC	**276**	**1.785E+05**	**1.409E+05**	**11.388**
N	**276**	**7.024E+06**	**3.531E+06**	**56.127**
V	**276**	**3.227E+08**	**1.657E+08**	**280.500**
E	**276**	**8.709E+11**	**1.318E+11**	**6031.786**
CC	**276**	**7.690E+03**	**2.055E+02**	**1.279**
INFO	**276**	**3.710E+15**	**3.693E+13**	**45085.200**

Table 11. Extra Statistics for Middle Refinement Level

Metric	*Statistics*			
	n	MSE	SS_x	X_{Mean}
LOC	283	5.893E+04	3.820E+04	11.696
N	283	2.757E+06	1.812E+06	70.360
V	283	1.285E+08	6.481E+07	349.226
E	283	7.614E+11	3.576E+11	9348.640
CC	283	9.615E+03	2.577E+03	2.502
INFO	283	7.530E+13	2.752E+12	13747.350

Table 12. Extra Statistics for High Refinement Level

Metric	*Statistics*			
	n	*MSE*	SS_x	X_{Mean}
LOC	422	4.404E+04	6.717E+04	14.358
N	422	3.959E+06	5.590E+06	102.960
V	422	1.345E+08	1.954E+08	525.775
E	422	2.239E+12	3.916E+12	21439.980
CC	422	8.218E+03	1.313E+04	4.583
INFO	422	2.719E+17	1.650E+14	52433.400

increased accuracy it can be seen that the confidence intervals, in general, are "thinner" than those given above. The exception to this is at a low level of refinement where the intervals are actually wider for the micro metrics. For example, compare Figure 7 on page 53 with Figure 12 on page 68.

It is quickly apparent that the confidence interval in the plot with only those procedures that are highly refined is much more concise, but the interval in Figure 13 on page 69 is much wider. This reflects the fact that at low levels of refinement, the complexity of the resultant source is difficult to predict with any degree of accuracy using micro metrics.

The Information Flow metric is unaffected by level of refinement. This can clearly be seen by comparing the plot shown in Figure 8 on page 54 (using all of the data) with Figure 14 on page 70 (using only data from highly refined procedures) and Figure 15 on page 71 (using only data from least refined procedures). The confidence intervals in the three figures are almost identical indicating that the Information Flow metric is totally independent of refinement level. Plots for the five micro metrics and the Information Flow metric at all three levels of refinement are given in Appendix D.

Inter-metric Correlations

Table 13 on page 73 contains the inter-metric correlations for the Pascal code. This information serves to verify studies previously done by [BASV83][CANJ85][HENS81b] [LIH 87] and [REDG84]. The table indicates that between micro metrics there is, in general, a high degree of correlation. When comparing the micro metrics with the structure metric Information Flow however, it is seen that the results are quite close to zero. This result indicates that the code and structure metrics are measuring different aspects of the source code.

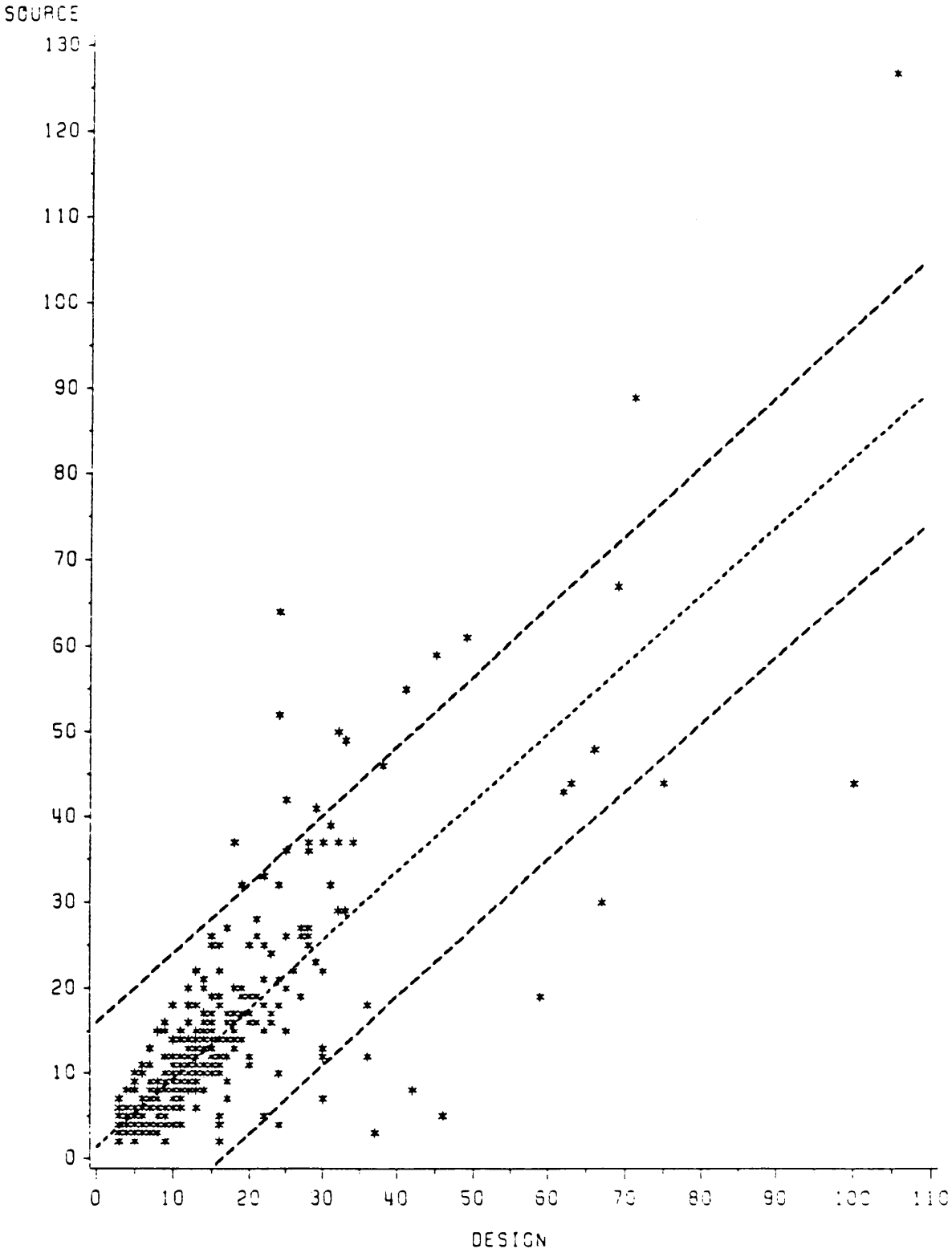

Figure 12. Regression and 95% Confidence lines for Highly Refined LOC

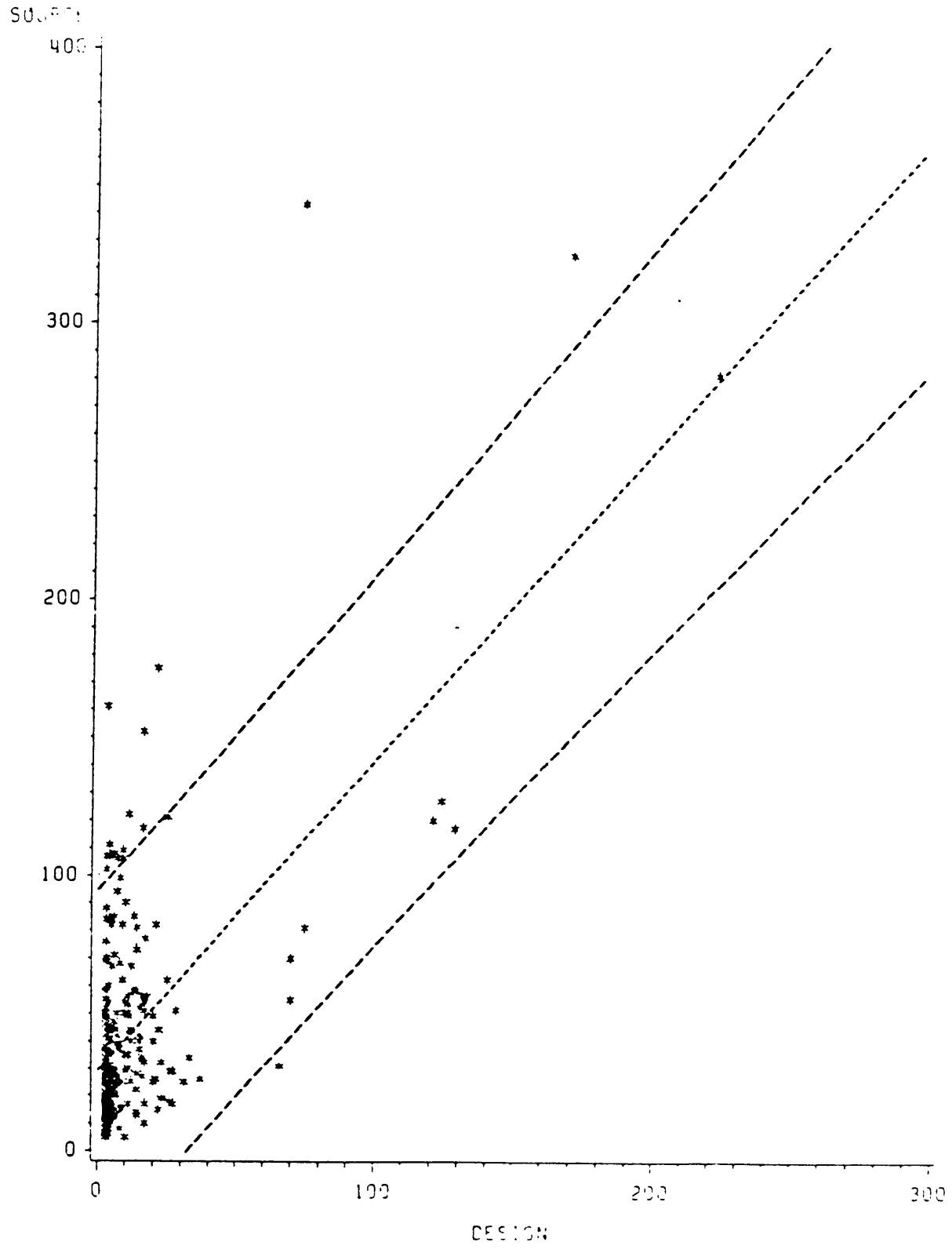

Figure 13. Regression and 95% Confidence lines for Least Refined LOC

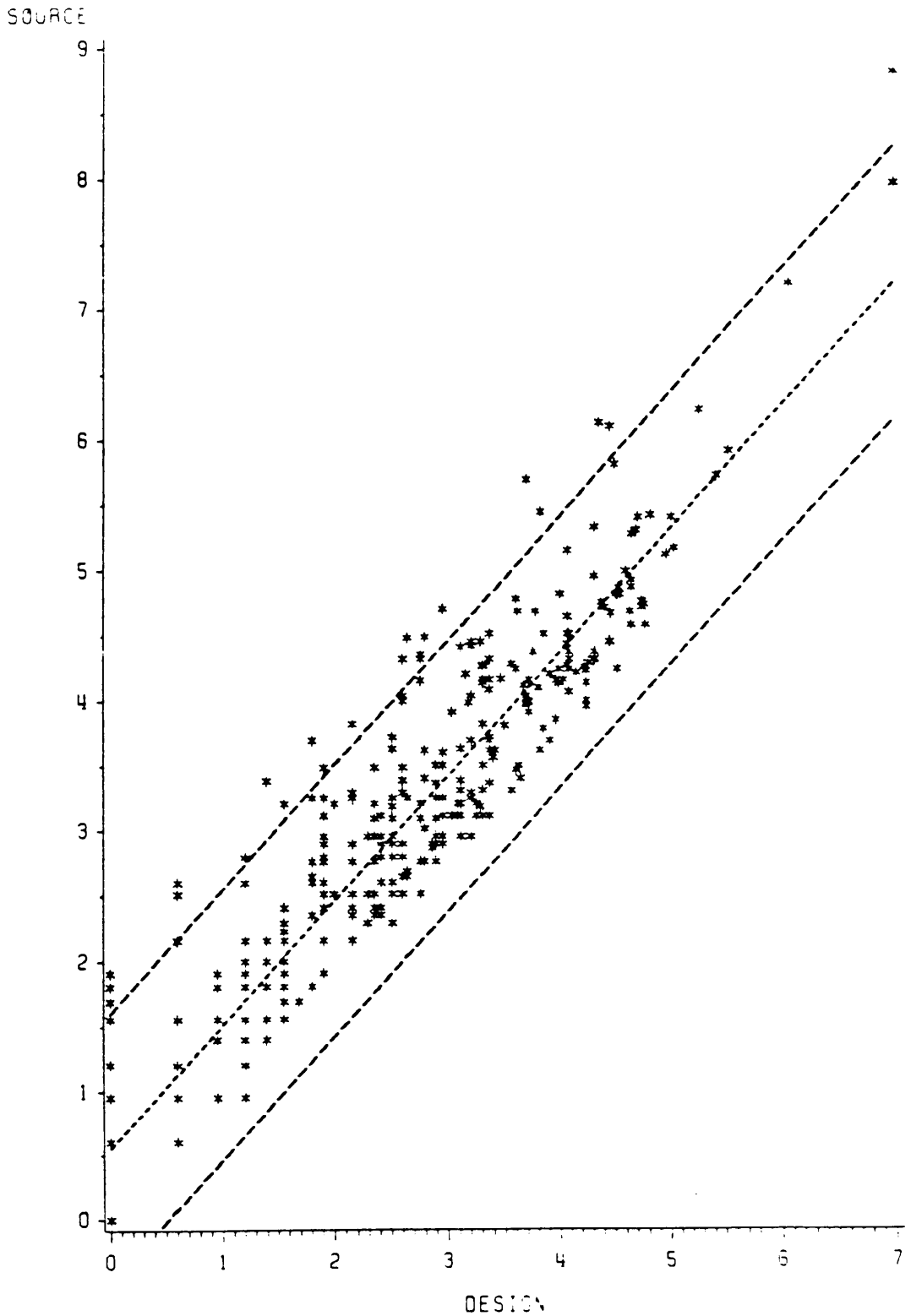

Figure 14. Regression and 95% Confidence lines for Highly Refined INFO

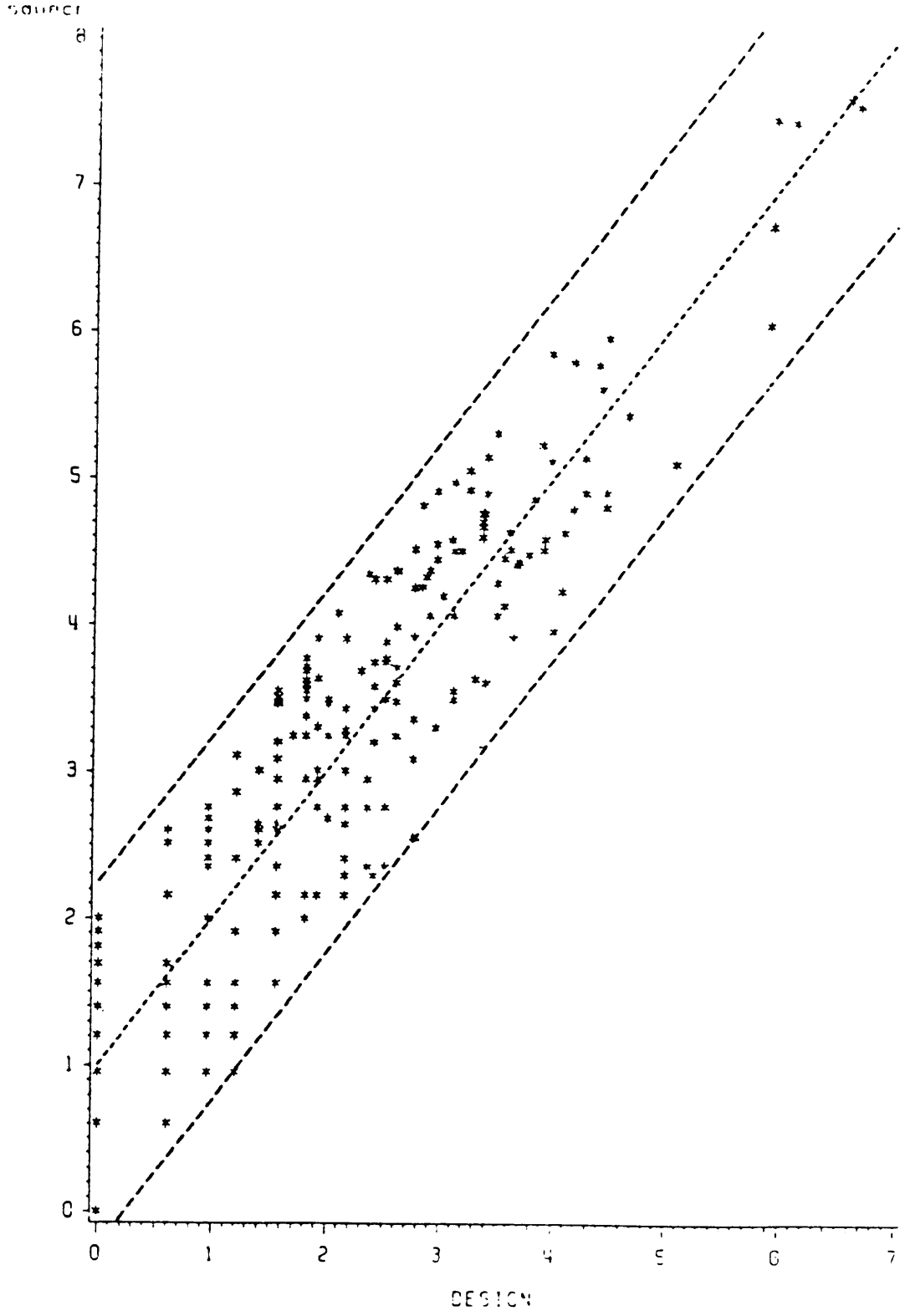

Figure 15. Regression and 95% Confidence lines for Least Refined INFO

The inter-metric correlations, with respect to the design, show the same relationships as those using Pascal, and are given in Table 14 on page 74. This is a desirable result since it indicates a consistency of measurement when comparing the designs to the resultant source code. It also lends credence to the usefulness of performing complexity measures at design time.

Conclusion

This chapter has presented the statistical analysis for both the data as a whole and based on levels of refinement. It shows that the structure and hybrid metrics do much better than the code metrics, and more importantly, proves that structure metrics are independent of level of refinement. Regression lines and confidence interval plots allow the calculation of predictor equations for the resultant source code complexity values to be predicted based on the design. It concludes by presenting the results of inter-metric correlations for both the ADLIF design code and the Pascal source code in support of previous studies.

Table 13. Intermetric Correlations for the Pascal code

Pascal Source Code						
Metric	**LOC**	**N**	**V**	**E**	**CC**	**INFO**
LOC						
N	0.893					
V	0.885	0.989				
E	0.521	0.749	0.711			
CC	0.629	0.776	0.781	0.492		
INFO	0.044	0.029	0.036	0.005	0.019	

Table 14. Intermetric Correlations for ADLIF code

ADLIF Design Code						
Metric	**LOC**	**N**	**V**	**E**	**CC**	**INFO**
LOC						
N	0.894					
V	0.894	0.988				
E	0.465	0.725	0.681			
CC	0.541	0.702	0.656	0.662		
INFO	0.260	0.208	0.249	0.039	0.039	

Chapter 7 Conclusions

Introduction

The use of software quality metrics has become much more common in the computer science community. Up to this point, however, their use has been mainly restricted to the measurement of production software or, in the case of design, limited to micro metrics. In many of the studies using micro metrics to help evaluate a design, the complexity values are hand generated. This research is an attempt to show that (1) Structure and Hybrid metrics are extremely useful at design time, and (2) automatic generation of metrics for design specifications is not only possible, but also desirable.

Research Conclusions

The results of this study clearly indicate several things:

- It is possible to generate meaningful complexity values at design time.

- A minimum amount of detail must be included in the specifications to make the measures useful.
- It is possible to predict the complexity values of the resultant source from the design measurements.
- Structure metrics are independent of level of refinement.
- Micro metrics are NOT independent of level of refinement. Due to this fact, the hybrid metrics are also somewhat dependent on level of refinement.

Some of these conclusions are to be expected, and this study simply served to verify them. It seems obvious that measures can be done if the design language is similar to an actual programming language, and that a minimum level of detail must be attained in the specifications in order to make the measurements meaningful. Another expected result is that the micro metrics are NOT independent of refinement level. This is due to the fact that they are simple counters. Without a minimum amount of specifications available, there is not enough information to get an overall picture of the source routine that will result from the implementation of the specifications.

The other conclusions are not at all obvious. The result that structure metrics are independent of the refinement level is an excellent example. This leads to the belief that given a choice of metrics to use to measure a design, a structure metric, or more specifically Henry and Kafura's Information Flow metric, would be a wise selection. Secondly, the fact that the design measures may be used to predict the relative complexity of the resultant source is, although not totally unexpected, a salient result of the study.

Future Work

Due to the fact that the language ADLIF so closely resembles a programming language, several modifications need to be made to make ADLIF more general, while at the same time facilitating

ease of use. Obviously, if a design tool is difficult to learn and use, people will be resistant to it. To this end, several suggestions for improvement are:

- Construct an editor specific to ADLIF that will help in the specification process.
- Structure the comments to allow both ease of construction and report generation possibly similar to PSL/PSA's.
- Develop an on-line project management system that uses both ADLIF and an analyzer similar to the one used in this study in order to encourage the use of both of these tools as well as metrics to evaluate the design.

There is still much work to do with the idea of analyzing designs. More data needs to be collected with ADLIF, hopefully incorporating the above suggestions, in order to verify and extend this study to incorporate more programs of radically different types such as operating systems, compilers, etc. Also, ADLIF should be directly compared against different types of design languages in order to help determine the more useful aspects of design languages, with the idea of developing a PDL that is both useful and simple to use.

Conclusion

The advent of Program Design Languages is an opportunity for the Software Engineering field to further improve the development of quality software. New tools, such as the analyzer used in this study, and PDL sensitive editors must continue to be studied in an attempt to make the design process both more structured and more reliable. The incorporation of existing tools, such as software quality metrics, can be of great assistance in this undertaking and should be examined to determine their role in the software development process. Hopefully, the research done for this study will be one of the first steps in the on-going search for better design methodologies.

Bibliography

[ADA 80] *Reference Manual for the ADA Programming Language*, United States Department of Defense, July, 1980.

[ARTJ85] Arthur, J.D., Henry, S.M., Nance, R.E., "Immediate Software Development Issues For Embedded Systems Applications In Surface Combatants," Virginia Polytechnic Institute Technical Report, TR-85-19, Mar. 1985.

[ARTJ86] Arthur, J.D., Nance, R.E., Henry, S.M., "A Procedural Approach to Evaluating Software Development Methodologies: The Foundation," Virginia Polytechnic Institute Technical Report, TR-86-24, 1986.

[BASV75] Basili, V.R., Turner, A.J., "Iterative Enhancement: A Practical Technique For Software Development," IEEE Transactions on Software Engineering, Vol. SE-1, No. 4, Dec. 1975, pp. 390-396.

[BASV83] Basili, V.R., Selby, R.W., Phillips, T.Y., "Metric Analysis and Data Validation Across Fortran Projects," IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, Nov. 1983, pp. 652-663.

[BECR84] Becker, R.A., Chambers, J.M., *S An Interactive Environment for Data Analysis and Graphics*, Belmont, CA., The Wadsworth Statistics/Probability Series, 1984.

[BOHM71] Bohl, M., *Flowcharting Techniques*, Palo Alto, CA., Science Research Associates, 1971.

[BROF75] Brooks, F.P., *The Mythical Man-Month*, Reading, MA., Addison-Wesley, 1975.

[BROF86] Brooks, F.P., "No Silver Bullet: Essence and Accidents of Software Engineering," Information Processing '86, Elsevier Science Publishers B.V., 1986.

[CANJ85] Canning, J.T., *The Application of Software Metrics to Large-Scale Systems*, Ph.D. Thesis, Computer Science Department, Virginia Polytechnic Institute, April 1985.

[CONS86] Conte, S.D., Dunsmore, H.E., Shen, V.Y., *Software Engineering Metrics and Models*, The Benjamin/Cummings Publishing Company, Inc., 1986.

[DEMT82] DeMarco, T., *Controlling Software Projects*, Elsevier North-Holland Publishing Company, 1982.

[ELSJ78] Elshoff, J.L., "An Investigation Into The Effect of The Counting Method Used on Software Science Measurements," ACM SIGPLAN Notices, Vol. 13, No. 2, Feb. 1978, pp. 30-45.

[FAIR85] Fairley, R., *Software Engineering Concepts*, McGraw-Hill, Inc., 1985.

[HALM77] Halstead, M.H., *Elements of Software Science*, New York, Elsevier North-Holland, 1977.

[HEAH79] Heacox, H.C., "RDL: A Language For Software Development," ACM Sigplan Notices, Vol. 14 No. 12, Dec. 1979, pp. 71-79.

[HENS79] Henry, S.M., *Information Flow Metrics for the Evaluation of Operating Systems' Structure*, Ph.D. Thesis, Computer Science Department, Iowa State University, Ames, Iowa, 1979.

[HENS81a] Henry, S.M., Kafura, D., "Software Structure Metrics Based on Information Flow," IEEE Transactions on Software Engineering, Vol. SE-7, No. 5, Sept. 1981, pp. 510-518.

[HENS81b] Henry, S.M., Kafura, D., Harris, K., "On the Relationships Among Three Software Metrics," Performance Evaluation Review, Vol. 10, No. 1, Spring, 1981, pp. 81-88.

[HENS83] Henry, S.M., "A Project Oriented Course On Software Engineering," ACM SIGCSE Bulletin, Vol. 15, No. 1, Feb. 1983, pp. 57-61.

[HENS88] Henry, S.M., "A Technique for Hiding Proprietary Details While Providing Sufficient Information for Research," Journal of Systems and Software, 1988 (to appear).

[KAFD82] Kafura, D., Henry, S.M., "Software Quality Metrics Based on Interconnectivity," Journal of Systems and Software, Vol. 2, 1982, pp. 121-131.

[KAFD84] Kafura, D., Canning, J., Reddy, G., "The Independence of Software Metrics Taken at Different Life-Cycle Stages," Proceedings: Ninth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Nov. 1984.

[KAFD85] Kafura, D., Canning, J., "A Validation of Software Metrics Using Many Metrics and Two Resources," Prodeedings: International Conference on Software Engineering, London, England, Aug. 1985.

[LIH 87] Li, H.F., Cheung, W.K., "An Empirical Study of Software Metrics," IEEE Transactions on Software Engineering, Vol. SE-13, No. 6, June 1987, pp. 697-708.

[LUCD85] Luckham, D., Von Henke, F., "An Overview of Anna, a Specification Language for Ada," IEEE Software, Mar. 1985, pp. 9-22.

[MCCC78] McClure, C., "A Model for Program Complexity Analysis," Proceedings: 3rd International Conference on Software Engineering, May 1978, pp. 149-157.

[MCCT76] McCabe, T.J., "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec. 1976, pp. 308-320.

[OVEC86] Overstreet, C.M., Nance, R.E., Balci, O., Barger, L.F., "Specification Languages: Understanding Their Role in Simulation Model Development," Virginia Polytechnic Institute Technical Report, SRC-87-001, Dec. 1986.

[RAMI83] Ramsey, H.R., Atwood, M.E., Van Doren, J.R., "Flowcharts Versus Program Design Languages: An Experimental Comparison," Communications of the ACM, Vol. 26, No. 6, June 1983, pp. 445-449.

[REDG84] Reddy, G., *Analysis of a DataBase Management System Using Software Metrics*, M.S. Thesis, Computer Science Department, Virginia Polytechnic Institute, June 1984.

[ROSD77] Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 16-34.

[ROSD85] Rosenblum, D.S., "A Methodology for the Design of Ada Transformation Tools in a DIANA Environment," IEEE Software, Mar. 1985, pp. 24-33.

[SHES81] Sheppard, S.B., Kruesi, E., Curtis, B., "The Effects of Symbology and Spatial Arrangement On The Comprehension of Software Specifications," IEEE, 1981, pp. 207-214.

[SNEH81] Sneed, H., "SOFTDOC - A System for Automated Software Static Analysis and Documentation," ACM, March 1981, pp. 173.

[SUTS81] Sutton, S.A., Basili, V.R., "The Flex Software Design System: Designers Need Languages, Too," IEEE Computer, Nov. 1981, pp. 95-102.

[SZUP81] Szulewski, P.A., Whitworth, M.H., Buchan, P., DeWolf, B., "The Measurement of Software Science Parameters in Software Design," ACM, March 1981, pp. 89.

[TEID77] Teichroew, D., Hershey III, E.A., "PSL/PSA: A Computer_Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, Jan. 1977, pp. 282-289.

[TROD81] Troy, D.A., Zweben, S.H., "Measuring the Quality of Structured Design," ACM SIGSOFT sponsored Software Engineering Symposiom, June 1981.

[WALR78] Walpole, R.E., Myers, R.H., *Probability and Statistics for Engineers and Scientists, 2 Edition*, McMillan : New York, 1978.

[WOOM79] Woodard, M.R., Hennell, M.A., Hedley, D., "A Measure of Control Flow Complexity in Program Text," IEEE Transactions on Software Engineering, Vol SE-5, No. 1, Jan. 1979, pp. 45.

[YAUS80] Yau, S.S., Collofello, J.S., "Some Stability Measures for Software Maintenance," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, Nov. 1980, pp. 545-552.

[YOUE79] Yourdan, E., Constantine, L., *Structured Design*, Prentice Hall, 1979.

Appendix A. ADLIF Language Reference Manual

Character Set

All language constructs may be represented with the ASCII character set, which is subdivided as follows:

1. Upper Case Letters
2. Lower Case Letters

Note: A lower case letter is not equivalent to the corresponding upper case letter.

3. Digits
4. The Space Character
5. Special Characters as Follows:

' " # % & ( ) * + , - . / : ; = < > _ | ! $? @ \ [ ] { }

Names

Names can denote declared entities. These are types, procedures, functions, constants, scalar variables, and array variables.

A name may consist of one or more alphanumeric ASCII characters. The first character must be alphabetic (A-Z, a-z). Following characters may be alphanumeric or underscores. A name may not end with an underscore, or have two consecutive underscores. Note that names may be any length greater than zero, but must be unique within the first twelve (12) characters. Examples are shown below.

```
PI
INCREMENT
LOOP_COUNTER1
LOOP_COUNTER2
-- the variable LOOP_COUNTER2 is not unique in the first twelve characters
-- so it will be treated as the same variable as LOOP_COUNTER1.
```

Comments

Comments are strings of characters preceded by two hyphens that continues until the end of the line, where an automatic termination of the comment occurs.

```
--THIS IS A COMMENT.
C :=          -- Comments are ignored when
A + B;        -- occuring any place in a program.
              -- The above two statements are equivalent to C := A + B;
```

Comments are descriptive and may be used anywhere in a program except before the program declaration and following the "END program name" statement.

Some comments in ADLIF are required and are enclosed in parentheses and followed by a semicolon. Comments are required following the program statement, and after a subprogram declaration.

The final type of comment in ADLIF is one enclosed in braces ("{}"). This type of comment may be used in an expression as a replacement of a mathematical expression. For example: ((X + 3) DIV Y) may be replaced by such expressions as ({add X and 3} DIV Y) or ({first add X and 3 and then integer divide that sum by Y}).

Arithmetic Precedence

The arithmetic precedence in ADLIF is as follows:

Parenthesis	()
Braces	{}
Function Calls	
NOT	
Multiplication	*, /, DIV, REM, AND
Addition	+, -, OR
Relational Operators	>, <, =, > =, < >, < =

Therefore, multiplication is done before subtraction and after function calls. In the case of two operators of the same type in an expression, the operations are done from left to right. Some examples are shown below. The derivations are greatly increased in order to clarify the order of operations.

```
(3 + 8 / 2 * (5 + 7)) + 5      = (3 + 8 / 2 * 12) + 5
                               = (3 + 4 * 12) + 5      -- Notice that the division
                                                       -- is done first since *
                                                       -- and / have the same
                                                       -- precedence.
                               = (3 + 48) + 5
                               = (51) + 5
                               = 51 + 5
                               = 56
```

```
NOT ((TRUE OR TRUE AND FALSE) OR TRUE)
                               = NOT ((TRUE OR FALSE) OR TRUE)
                               = NOT ((TRUE) OR TRUE)
                               = NOT (TRUE OR TRUE)
                               = NOT (TRUE)
                               = NOT TRUE
                               = FALSE
```

Built-In Functions and Procedures

Some built-in functions have been provided in ADLIF and may be assumed to exist. The functions and their required parameters and return values are as follows:

ROUND

Parameter is an expression which resolves to a real number The returned value is an integer. If the fractional part of the real number is less than 0.5, the ROUND function returns the integer portion of the real. If the fractional portion of the real number is greater than or equal to 0.5, the function returns the absolute value of the integer portion of the real plus 1 and the correct sign is added. Examples are shown below.

```
Intnum := ROUND (1.332 + 0.1);          -- Intnum = 1
Intnum := ROUND (45.5);                 -- Intnum = 46
Intnum := ROUND (-3.78);                -- Intnum = -4
Sum := 7.2;
Intnum := ROUND (Sum);                  -- Intnum = 7
```

TRUNC

Parameter is an expression which resolves to a real number. The returned value is an integer. The integer returned is simply the integer portion of the real. Examples are shown below.

```
Intnum := TRUNC (1.332);                -- Intnum = 1
Intnum := TRUNC (45.5 - 2.3);           -- Intnum = 43
Intnum := TRUNC (-3.78);                -- Intnum = -3
Sum = 7.2;
Intnum := TRUNC (Sum + 4);              -- Intnum = 11
```

INTTOREAL

Parameter consists of an expression which resolves to an integer. The returned value is a real number. The real value returned by the function is Integernum.0. Examples are shown below.

```
Realnum := INTTOREAL (3);                -- Realnum = 3.0
Realnum := INTTOREAL (-715);             -- Realnum = -715.0
Realnum := INTTOREAL ((3 + 7) DIV 5);    -- Realnum = 2.0
```

EOLN

Parameter is a filename and the returned value of EOLN is a boolean. If the pointer to the filename is at the end of a line, the function returns TRUE, otherwise the function returns FALSE. If no filename is supplied, filename defaults to INPUT.

EOF

Parameter is a filename and the returned value of EOF is a boolean. If the pointer to the filename is at the end of the file, the function returns TRUE, otherwise the function returns FALSE. If no filename is supplied, filename defaults to INPUT.

NEW

Parameter is a variable of type ACCESS. The function creates a node in memory (that is, memory is allocated for the data structure accessed by the pointer) and the function returns the address of the new node.

ORD

Parameter is an expression which resolves to a character. The returned value is an integer which is the ASCII value of the character. Examples are shown below.

```
    Integernum := ORD ('a');                    -- Integernum = 96
let Charac be a constant with value 'b' then
    Integernum := ORD (Charac);                 -- Integernum = 97
let Charact be a variable with value 'e' then
    Integernum := ORD (Charact);                -- Integernum = 100
```

CHR

Parameter is an expression that resolves to an integer. The returned value is the character represented by the integer value (if any) using ASCII. If no such character exists, the function returns ' '. Examples are shown below.

```
    Charval := CHR (94 + 3);                    -- Charval = 'b'
Let Temp be a variable with value 99 then
    Charval := CHR (Temp);                      -- Charval = 'd'
```

SUCC

Parameter is an expression which resolves to a character. The returned value is a character such that ORD(returnchar) = ORD(parameter) plus 1. Examples are shown below.

```
    Charval := SUCC ('A');                      -- Charval = 'B'
    Charval := SUCC (CHR (96));                 -- CHR (96) = 'a' so Charval = 'b'
```

PRED

Parameter is an expression which resolves to a character. The returned value is a character such that ORD(returnchar) = ORD(parameter) minus 1. Examples are shown below.

```
Charval := PRED ('B');                    -- Charval = 'A'
Charval := PRED (CHR (97));               -- CHR (97) = 'b' so Charval = 'a'
```

Note that PRED(SUCC(expression)) = SUCC(PRED(expression)) = (expression).

General Format of a Program

The general format of a program is as follows:

```
PROGRAM STATEMENT;
[DECLARATION SEGMENT;]
BEGIN program-name
   STATEMENT[S];
END program-name.
```

Program-name is the identifier used directly after the reserved word PROGRAM in the PROGRAM STATEMENT. Note that anything except array subscripts enclosed in brackets is optional. Therefore, the declaration segment of a program may be omitted.

Program Statement

The program statement is always the first statement in every program. Its general format is:

```
PROGRAM program-name [(filename[,filename...])]; (comment);
```

Program-name is any valid identifier and *filename* is one of the following: Input, Output, or Identifier. *Input* defaults to system input, *Output* defaults to system output and *Identifier* is the name of a data file for input or output. The identifier must be declared to be of type INFILE, OUTFILE, or INOUTFILE in the declaration section of the mainline code. An INFILE can only be used for input, an OUTFILE can only be used for output, and in INOUTFILE can be used for both input and output. *Comment* is a required comment explaining the purpose of the program.

Declaration Section

The declaration section consists of zero or more of the following in this order:

A use clause,

Constant declarations,

Type declarations,

Variable declarations,

Subprogram declarations.

Any or all of the above may be omitted, but the relative order must remain the same. Each of the entities above is explained in greater detail below.

USE Clause

Please see PACKAGE USAGE for details on the use of the USE clause.

Constants

Constants may be declared in the declaration sections of the mainline code, procedures, or functions. They must appear before the type declarations. A constant is a way of naming a number or a string that is used frequently and might be changed sometime in the future. Thus to change the value used, only the constant declaration must be modified, not any of the actual code.

The general format is:

```
CONST[ANT] Identifier = {Number} ;
                        {String}
```

where *identifier* is a valid name, *number* is a real or integer, and *string* is a literal string enclosed in single quotes. Examples are shown below.

```
CONST Increment = 1;
CONSTANT Blank = ' ';
CONST Rate = 0.5;
```

The constant name may be used anywhere the number or string may be used. For example:

```
Count := Count + Increment; -- is the same as Count := Count + 1;
```

Character Types

Predefined

A variable that is of the type CHAR[ACTER] may contain the value of any one ASCII character. See the following for an example.

```
TYPE MIDDLE_INITIAL = CHAR;
    FIRST_INITIAL = CHARACTER;
VAR PERSON : MIDDLE_INITIAL;
    FIRST_NAME : FIRST_INITIAL;
    LAST_INITIAL : CHAR;
```

The variables PERSON, FIRST_NAME and LAST_INITIAL may take on any legal ASCII character such as PERSON := 'A';

User-defined

Each of the 95 printable characters of the ASCII character set can be denoted by a character literal or a range. See the following for an example.

```
TYPE  ROMAN_DIGIT = ('I', 'V', 'X', 'L', 'C', 'D', 'M');
VAR ROMAN_VAR : ROMAN_DIGIT;
```

The variable ROMAN_VAR may take on any one character contained in the type ROMAN_DIGIT.

```
TYPE CAPITAL = ('A'..'Z');
VAR MIDDLE_INITIAL : CAPITAL;
```

The variable MIDDLE_INITIAL may take on the value of any one character contained in the range from capital A to capital Z.

Boolean Types

There is a predefined type named BOOL[EAN]. It may contain the two values FALSE and TRUE, ordered with the relation FALSE < TRUE. The evaluation of a condition must deliver a result of this predefined type. For example:

```
VAR FLAG : BOOL;
```

The variable FLAG may take on either TRUE or FALSE (e.g., FLAG := TRUE;).

Integer Types

Predefined

A variable that is of the type INT[EGER] may contain the value of any integer from MIN_INT to MAX_INT, where these predefined constants are machine dependent. For example:

```
VAR COUNT : INTEGER;
```

In the example, the variable COUNT may take on any integer value from MIN_INT to MAX_INT.

User-defined

A type may be declared to be a subset of the full integer range using the following format:

```
TYPE IDENTIFIER = (Int..Int);
```

A variable of the user defined type may take on only values within the specified range. All arithmetic operations are equivalent within the subrange. *Int* may be a constant or *Int..Int* may be a previously defined type. Examples are given below.

```
TYPE COUNTER = (0..100);
VAR COUNT : COUNTER;
```

In the above example, the variable COUNT may take on any integer value from 0 to 100.

```
TYPE COUNTING = COUNTER; VAR COUNT2 : COUNTING;
```

COUNT2 has the same restrictions as COUNT in the previous example.

Real Types

Predefined

A variable of the type REAL can contain the value of a real number from MIN_REAL to MAX_REAL, inclusive. (Like MAX_INT and MIN_ INT, MAX_REAL and MIN_REAL are machine dependent.)

User-defined

A type may be declared that is a subrange of the reals. A variable of such a type may contain any real value which is contained in the declared subrange, inclusive.

All mathematical operations can be applied to variables of type real including: Addition (+), Subtraction (-), Multiplication (*), Real Division (/), Relational Operations (including <, >, =, > =, < =, < >). See below for some examples.

```
TYPE    PORTION = (-1.0..1.0);
VAR     GREAT : REAL;          -- May contain any real value.
        SMALL : PORTION;       -- May contain any real value between -1.0 and
                               -- 1.0 inclusive.
```

Record Types

A record is of the form:

```
TYPE IDENTIFIER =
        RECORD
            COMPONENT_LIST;
        ENDRECORD;
```

where the IDENTIFIER is any valid identifier name, and COMPONENT_LIST consists of a variable number of identifiers and their corresponding types. There must be at least one identifier-type pair in the COMPONENT_LIST. Some examples are:

```
TYPE CALENDER =
        RECORD
          DAY : (1..31);
          MONTH : (1..12);
          YEAR : (0..4000);
        ENDRECORD;
TYPE COMPLEX =
        RECORD
          RE : REAL;
          IM : REAL;
        ENDRECORD;
```

```
TYPE DATE =
        RECORD
          DAY : ('SUN','MON','TUES','WED','THURS','FRI','SAT');
          MONTH : (1..12);
          DAY_NUM : (1..31);
          YEAR : (0..99);
        ENDRECORD;
```

In order to access a field of a record, use the following format:

```
Identifier.Fieldname[.Fieldname]
```

where *Identifier* is the name of the variable that is of the record type, and *Fieldname* is a field in the record declaration.

Array Types

An array is of the form:

```
TYPE Identifier = ARRAY (INDEX[,INDEX...]) OF TYPE2;
```

where *Identifier* is any valid name, INDEX is a range of type integer or INDEX is a user defined type with explicit values, and TYPE2 is a predefined or user defined type. See below for some examples.

```
TYPE
    TABLE = ARRAY (1..10) OF INTEGER;
TYPE
    LINE = ARRAY (1..MAX_LINE_SIZE) OF CHARACTER;
                                        --where MAX_LINE_SIZE
                                        --is a predefined constant
```

```
TYPE
    PAGE = ARRAY (A..Z) OF LINE;          -- an array of an array
TYPE
    DAY = ('SUN','MON','TUES','WED','THURS','FRI','SAT');
TYPE
    SCHEDULE = ARRAY (DAY) OF BOOLEAN;
TYPE
    PAGE2 = ARRAY (1..50, 1..80) OF CHARACTER;
                                          -- an array with two dimensions
```

NOTE: An array may have any number of indices greater than 0.

To access an element of an array, use the following format:

```
Arrayname[Indices]
```

where *Arrayname* is the name of the variable declared as type ARRAY, and *Indices* is the correct number of index numbers into the array. For example, if the variable PAPER is declared to be of type PAGE2, in order to access the element with the first index equaling 30 and the second index equaling 3, the format is: PAPER[30,3].

Pointers

Access types are types for which objects may be dynamically allocated. If POINT has the type ACCESS INTEGER, then the declaration X : POINT creates a pointer to an integer object (memory cell). The built in procedure NEW must be used to create the INTEGER access object itself. The statement X := NEW (INTEGER(3)); creates an integer object with the integer value 3 and assigns a pointer to that object.

Object declarations cause an object to come into existence when they are elaborated and to be deleted on exit from the scope in which they are declared. Access objects are created using NEW and they continue to exist as long as there is a path to the object from an access variable. An example is:

```
TYPE POINT = ACCESS INTEGER;
VAR PT1, PT2 : POINT;

PT1 := NEW (INTEGER (3));
PT2 := PT1; -- PT1, PT2 share accessed object
```

Access types are generally used to point to records (even though pointers may access any type). In this way, linked lists may be formed by inserting an access type into the record that points to the type record. Using this method, a list may be formed that is of dynamic length. An object that is of type ACCESS to a record may be declared before the record declaration, but if the record does not appear anywhere in the type declarations, an error will result. For example:

```
TYPE LINK = ACCESS ELEMENT;
TYPE ELEMENT =
        RECORD
          VALUE : INTEGER;
          NEXT : LINK; -- Points to element
        ENDRECORD;
VAR EL1, EL2 : LINK;
EL1 := NEW (ELEMENT);
EL2 := NEW (ELEMENT);
EL1.NEXT := EL2;
```

Record fields may be accessed using pointers in the same way as a field in a record is accessed (i.e., in the example above, EL1.NEXT.VALUE refers to the VALUE field in the record following the one pointed to by EL1).

Variable Declarations

A variable declaration is in the following general format:

VAR[IABLE] Identifier [,Identifier...] : Type;

where *Identifier* is a valid unique name and *Type* is the desired variable type. For example:

VAR Count : INTEGER; VARIABLE Done,Flag : BOOLEAN;

The type used may be a built-in type such as CHAR, INTEGER, etc. or a previously user-defined type. Variable declarations appear in the declaration section of the program or a subprogram following user defined type declarations (if any).

Subprograms

A subprogram is in one of two categories: a procedure which is called as a statement and does not return a result, and a function which is part of an expression and does return a result. A subprogram may be declared inside the declaration segment of the mainline code or another subprogram. If a subprogram is declared from inside a procedure or function, it is local and may be called only within the scope of that procedure or function. Note: A subprogram may never be called by a procedure or function outside of the scope of the subprogram or by a procedure or function declared earlier in the program body.

Procedures

The general form of a procedure is:

PROCEDURE identifier[(parameter:mode type[;parameter:mode type])];

(comment)

[Declarative part;]

BEGIN identifier

  statement[s];

END identifier;

In the general format, *identifier* represents a valid name which is used to identify the procedure in a call to it, and the BEGIN and END statements in the procedure body. *Parameter* is a valid name which designates a one-one correspondence between the parameter and a variable in the procedure call. The *mode* may be one of three different types: IN, OUT, or INOUT. The effect of the three modes is as follows:

IN

The parameter acts as a local constant whose value is provided by the corresponding actual parameter. If a mode is not specified, IN is assumed.

OUT

The parameter acts as a local variable whose value is assigned to the corresponding actual parameter as a result of the execution of the subprogram.

INOUT

The parameter acts as a local variable and permits access and assignment to the corresponding actual parameter.

Note: Mode effects are from the ADA Reference Manual (Section 6.2).

Type must be of the same type as the corresponding actual parameter. The comment is required as part of the subprogram and should be a few lines describing the purpose of the subprogram and the variables used. The declarative part contains the Constants, Types, Variables and other Sub-programs used in the subprogram. Statements must be one or more valid program statements. See Figure 16 on page 101 for an example of a procedure declaration.

Procedure Calls

A procedure call is a statement of the form:

Identifier [(parameter[,parameter...])];

In the call, *Identifier* is the name of the desired procedure and (*parameter*[,*parameter*...]) is(are) the variable(s) needed by the subprogram. Note that the parameters must be in the same order as they appear in the procedure declaration. For example:

TOTAL (COUNT);

FLEX;

FIND (SUM,TOTAL,NEWSUM);

Functions

The general form of a function is:

FUNCTION identifier [(parameter : type [;parameter : type...])] : returntype;

(comment);

[declarative part;]

```
PROCEDURE Simple (Num1: IN INTEGER; Num2 : INTEGER; Num3,Num4 : INOUT
        INTEGER; Result : OUT INTEGER);

(This procedure doesn't do much of anything.  It adds, subtracts, and multiplies
numbers in a haphazard manner as nothing more than an example);

CONST Increment = 1;

VAR Count : INTEGER;

BEGIN Simple
  Result := Num1 + Num2;
  Result := Result * Num3 * Num4;        -- Result may be used on the right side
                                         -- only because it was previously
                                         -- assigned a value.
  Num3 := Result + Increment;
  Count := Num1;
  WHILE (Count < Num3) LOOP
    Num4 := Num4 + Increment;
    Count := Count + Increment;
  ENDWHILE;
END Simple;
```

Figure 16. Example of Procedure Declaration

```
BEGIN identifier
   statement[s];
   identifier := value;
   statement[s];
END identifier;
```

In the general format, *identifier*, *parameter*, and *type* are identical to those in procedures, but it should be noted that a mode is not used. All parameters in a function are, by default, of type IN. *Returntype* is the type of the value returned by the function. The comment and declarative part are identical to those in a procedure. *Value* is any valid right-hand side of an assignment statement. Note: There may be zero or more statements preceding and following the assignment statement to the function name. See Figure 17 on page 103 for an example of a function declaration.

Function Calls

A function call is a statement of the form:

```
Variable := [value operator] Identifier [(parameter[,parameter...])] [operator value];
```

In the call, *variable* is any valid declared variable that is of the same type as that of the right hand side of the assignment statement. *Value* is a declared variable or a valid mathematical expression, and *operator* may be one of the following: +,-,*,/,DIV,REM,AND,OR, >, <, =, >=, <>, <=. *Identifier* is the name of the desired function and (*parameter*[, *parameter*...]) is (are) the variable(s) needed by the function. For example:

```
COUNT := CALCULATE (OLDCOUNT);
STATE := (7*COUNT) + NEWVALUE (SUM,TOTAL);
STATE2 := NEWVALUE (STATE,COUNT) - 45 * (TOTAL + SUM);
```

```
FUNCTION Fraction (Number : Real) : Real;

(This function returns the fraction portion of a real number.);

VARIABLE Temp1 : Integer;
VAR Temp2 : Real;

BEGIN Fraction
  Temp1 := TRUNC (Number);          -- TRUNC is a built in library function.
  Temp2 := INTTOREAL (Temp1);       -- INTTOREAL is also a built in function
                                    -- which converts an integer to a real.
  Fraction := Number - Temp2;
END Fraction;
```

Figure 17. Example of Function Declaration

Statements

The execution of statements causes actions to be performed. All statements must terminate with a semicolon, except the final statement ("END program-name") and BEGIN statements. It should be noted that a statement may consist of a single semi-colon. For example:

```
;  -- is a valid ADLIF statement.
```

Assignment Statement

An assignment statement is of the form:

```
                          Expression
VARIABLE :=               VARIABLE NAME        ;
                          Constant
                          Function Call
                          -- may be any one of the above or a combination of them.
```

For example:

```
SUM := SUM + 1;
NEWVALUE := OLDVALUE;
COUNT := 10;
VALUE[3] := 15;
```

Read and Write Statements

A read may be in one of two forms:

```
READ ([filename,] variable [,variable...]);
READLN [([filename,] variable [,variable...])];
```

where *filename* is the name of a file of type INFILE or INOUTFILE. If no filename is given, the default is INPUT. L*variable* is any valid identifier name. The only difference between the two forms is the type of read.

In the first case, the next read encountered will continue on the same line in the data file until EOLN is encountered when a READLN must be used to reset the END-OF-LINE boolean condition to FALSE. If, when reading numbers, a number is not found, then a read error occurs. In the second case, after the read the equivalent of a carriage return is implemented and the rest of the line of data (if any) is disregarded. Any type of variable may be read except for pointers. If a boolean value is read, the entire word TRUE or FALSE may be read or abbreviated to just T or F.

A write may also be in one of two forms:

WRITE ([filename,] expression [,expression...]);

WRITELN [([filename,] [expression [,expression...]])];

where *filename* is the name of a file of type OUTFILE or INOUTFILE. If the filename is not given, the default is OUTPUT, and *expression* is either a constant, a variable, or a string expression encloded in single quotes. The difference between the two forms is that in form one, a carriage return is not performed; the next WRITE or WRITELN encountered will continue on the same line of the output file In the second form, a carriage return is performed after the output. Note: The contents of a variable or constant to be output may be only a number or a character. Therefore, even though a variable may be assigned the literal 'MONDAY', it may not be output using the variable name, but must be output by using a string expression or be output one character at a time. The exception is a boolean variable that may be used to output TRUE or FALSE.

Examples of both reads and writes are shown below.

If the input contains the two lines:

ABC 123

and the variables to be used are:

```
Num1, Num2 : INTEGER;
Char1, Char2 : CHARACTER;

READ (Char1, Char2);            -- Char1 = 'A', Char2 = 'B'
READLN (Char2, Char1, Num1);    -- Char2 = 'C', Char1 = ' ', Num1 = 123
READLN (INPUT, Num2);           -- Num2 = 12300 from INPUT
                                -- Note the 'A' following 12300 is
                                -- disregarded. Also note that Char1 is
                                -- a blank which is a valid character.
```

If the statements following are issued ..

```
WRITE (Char1, 'Char2 = ', Char2);
WRITELN (OUTPUT, ',',Num2);
WRITE ('Num1 contains the value ', Num1);
WRITELN (OUTPUT);
```

the OUTPUT file contains ..

```
 Char2 = C,12300
Num1 contains the value 123     -- Note that outputing Char1 produced a
                                -- blank.
```

If Statements

In all types of IF statements, the number of ENDIF's must equal the number of occurrences of the reserved word IF.

IF-THEN

```
IF (expression) THEN statement[s] ENDIF;
```

The IF-THEN checks the expression. When the expression is TRUE, the statements (1 or more statements) inside the THEN clause are executed and then control goes to the statements following the IF-THEN. When the expression is FALSE, control immediately goes to the statement following the IF-THEN statement.

For example:

```
IF (COUNT = 0)
   THEN TOTAL := SUM;
ENDIF;                  -- last statement followed by a semi-colon.

IF (TOTAL <= 10)
   THEN
      SUM := SUM + 1;
      COUNT := 10; -- This statement doesn't require a semicolon.
   ENDIF;

IF (LETTER = 'A')
   THEN FLAG := TRUE
```

```
ENDIF;
```

IF-THEN-ELSE

```
IF (expression) THEN statement[s] ELSE statement[s] ENDIF;
```

The expression is checked and when it is TRUE, the statement(s) inside the THEN portion are executed and control passes to the statement following the ENDIF. When the expression is FALSE, the statements(s) inside the ELSE clause are executed and control is passed to the statement following the ENDIF. For example:

```
IF (COUNT < = 0)
    THEN
        TOTAL := SUM;
    ELSE
        COUNT := 4;
        SUM := 0;
    ENDIF;
```

Nested IF

```
IF (expression) THEN statement[s] ELSEIF (expression)
  THEN statement[s] [ELSE statement[s]] ENDIF;
```

The IF-THEN portion acts the same as a typical IF-THEN. The ELSEIF portion is used to nest two or more IF-THEN-ELSE's and the separate portions of each act the same as their respective portions of a typical IF statement. For example:

```
IF (COUNT < 0) THEN
```

```
        TOTAL := SUM;
    ELSEIF (COUNT = 0) THEN
        TOTAL := SUM + 1;
    ENDIF;
```

Loop Statements

WHILE-LOOP

```
    WHILE (expression) LOOP
        statement[s];
    ENDWHILE;
```

The expression is checked and when TRUE, the statement(s) are executed and control passes back to the beginning of the WHILE statement. When the expression is FALSE, control immediately passes to the statement following the ENDWHILE. Examples are shown below.

```
    WHILE (SUM < = 10) LOOP
        SUM := SUM + 1;
    ENDWHILE;
    WHILE (COUNT > = 0) LOOP
        TOTAL := TOTAL + 1;
        COUNT := COUNT - 1;
    ENDWHILE;
```

FOR-LOOP

```
FOR VARIABLE {IN or INREVERSE} expression1, expression2 LOOP
   statement[s];
ENDFOR;
```

*Expression*1 is the initial value of VARIABLE and *Expression*2 is the final value of VARIABLE. In the FOR statement, the test for VARIABLE exceeding Expression2 is made prior to the execution of the loop statements. The test for comparing VARIABLE and Expression2 is as follows:

```
If IN and (VARIABLE > Expression2) then
    terminate the loop.
If INREVERSE and (VARIABLE < Expression2) then
    terminate the loop.
```

Examples:

```
FOR I IN 1, 10 LOOP
   SUM := SUM + 1;
ENDFOR;

-- This loop is executed once
FOR I INREVERSE J, J LOOP
   TOTAL := SUM + INDEX;
   SUM := SUM + 1;
ENDFOR;
```

The execution of the FOR loop starts by elaborating the iteration clause, which acts as the declaration of the loop parameter. The identifier of the loop parameter is first introduced and the discrete range is then evaluated: the loop parameter is declared as a variable local to the loop statement, whose type is that of the elements in the discrete range and whose range constraint is

given by the discrete range. In other words, in the statement FOR I = 1, 10 LOOP, the variable I is local to the loop and because the constants 1 and 10 are integers, I must be an integer. Inside the loop, I may only be used as a variable subscript or as an IN variable in a procedure call.

CASE Statement

The general format of the case statement is:

```
CASE variable IS
  WHEN choice = >     {statement[s]}   ;
                      {NULL    }
 [WHEN choice = >     {statement[s]}   ;...]
                      {NULL    }
ENDCASE;
```

Where *variable* is any valid variable name and *choice* is either a simple expression, a discrete range, or the reserved word OTHERS. All possible values of an expression must be provided in order to guard against accidental omissions. Sometimes a default expression is needed for all values not explicitly stated; this is provided by the reserved word OTHERS. NULL is used when no action is wanted but its presence indicates that the choice has been accounted for in the CASE statement. For example:

```
CASE Today IS
  WHEN 'Mon'..'Thu'     = >Work;
  WHEN 'Fri'            = >Work;
                            Party;
  WHEN OTHERS           = >NULL;
ENDCASE;
```

PACKAGE USAGE

A facility called "packages" is supplied in ADLIF in order to have the capability of modularizing a program design. Using a package, subprograms may be designed and used that are completely separate from any specific ADLIF program, thus allowing the use of the concept of information hiding. The overall form of a package is similar to that of an ADLIF program, and is given formally below.

```
PACKAGE package-name IS      -- This is the package
                             -- specification, and is
Visible-entities;            -- the only part of the
                             -- package visible to the
END package-name;            -- outside world.
PACKAGE BODY package-name IS
(comment);
[Declarative part;]          -- This is the body of the
                             -- package and contains
BEGIN package-name           -- all code related to the package.
statement[s];
END package-name;
```

In the specification above, *package-name* is any valid unique ADLIF name. *Visible-entities* are similar to a normal declarative part, with the exception that only those constants, types, variables, and subprogram declarations that are to be visible to the calling entity should be specified. Anything that is to be hidden to the calling entity should be omitted from this section. Note that procedures and functions not shown in the visible-entities section may only be used internally in the package. The *declarativepart* is identical to that in an ADLIF program or subprogram, as are *statement[s]*. Note that subprogram declarations shown in the visible-entities section must be

complete and identical to the declarations repeated in the declarative section in the body of the package.

In order to make the package specification visible to a program or another package, the reserved word USE is needed, and is placed as the first statement in the declarative section, with the following syntax:

USE package-name [,package-name];

Each declaration in the visible-entities section of the package must then be duplicated in the declarative section of the program using the package, with each entity followed by a package reference which indicates where the entity may be found. See below for the specific syntax of a package reference.

Visible-entity; PACKAGE package-name;

Each of the constants, types, variables, and subprograms from the package may be used or called in the normal manner. For a complete example of a package, see Figure 18 on page 114.

```
PACKAGE Manipulate IS      -- Specification

  PROCEDURE Push (Stack : INOUT Stacktype;
                  Elem : IN INTEGER);
  (This procedure implements the normal
   PUSH.);

  PROCEDURE Pop (Stack : Stacktype);
  (This procedure implements the normal
   POP function.);

END Manipulate;

PACKAGE BODY Manipulate IS      -- Body
(This package is the implementation of the
 stack operations PUSH and POP.);

  CONST Max = 100;

  TYPE Intrange = (0..Max);
  TYPE Stacktype = Array (1..Max) OF INTEGER;

  VAR Stack : Stacktype;
  VAR Top : Intrange;

  PROCEDURE Push (Stack : INOUT Stacktype;
                  Elem : IN INTEGER);
  (This procedure implements the normal PUSH.);

  BEGIN Push
    Top := Top + 1;
    Stack[Top] := Elem;
  END Push;

  PROCEDURE Pop (Stack : INTEGER);
  (This procedure implements the normal POP function.);

  BEGIN Pop
    Top := Top - 1;
  END Pop;

BEGIN Manipulate          -- Initialization
  Top := 0;
END Manipulate;
```

Figure 18. Example of Package Usage

Appendix B. BNF for ADLIF

```
< prog >          ::= < progstate > < decsec > BEGIN < identifier > < stmts > < morestmts >
                          END < identifier > . |
                      < packspec > < packbody >
< progstate >     ::= PROGRAM < identifier > ( < files > );( < comment > ); |
                      PROGRAM < identifier > ; ( < comment > );
< packspec >      ::= PACKAGE < identifier > IS < useclause > < consts > < types >
                          < variables > < subdec > END < identifier > ;
< subdec >        ::= PROCEDURE < identifier > < parameters > ; ( < comment > );
                          < package > < subdec > |
                      FUNCTION < identifier > < paramaters > : < type > ; ( < comment > );
                          < package > < subdec > |
                      < empty >
< packbody >      ::= PACKAGE BODY < identifier > IS ( < comment > ); < decsec > BEGIN
                          < identifier > < stmts > < morestmts > END < identifier > ;
< decsec >        ::= < useclause > < consts > < types > < variables > < subprograms > |
                      < empty >
< identifier >    ::= < letter > < identcont >
```

```
< letter >          ::= A | B | .. | Z | a | b | .. | z
< identcont >       ::= < alphanum > < identcont > |
                        < empty >
< alphanum >        ::= < letter > | _ | < digit >
< stmts >           ::= < assignment > ; |
                        < readstate > ; |
                        < writestate > ; |
                        < ifstate > ; |
                        < whileloop > ; |
                        < forloop > ; |
                        < procedurecall > ; |
                        < casestate > ; |
                        < emptystate > |
                        < empty >
< morestmts >       ::= < stmts > < morestmts > |
                        < empty >
< files >           ::= INPUT < morefiles > |
                        OUTPUT < morefiles > |
                        < identifier > < morefiles >
< morefiles >       ::= ,INPUT < morefiles > |
                        ,OUTPUT < morefiles > |
                        , < identifier > < morefiles > |
                        < empty >
< comment >         ::= < ascii > < comment > |
                        < ascii >
< ascii >           ::= < < x|x is any ASCII character other than ) > >
< useclause >       ::= USE < identifier > < moreident > ; |
```

```
                    <empty>
<consts>        ::= CONST <constdec> <consts> |
                    CONSTANT <constdec> <consts> |
                    <empty>
<constdec>      ::= <identifier> = <contassign>; <package>
<contassign>    ::= <number> |
                    <string>
<package>       ::= PACKAGE <identifier>; |
                    <empty>
<number>        ::= <pos or neg> <digits>.<digits> |
                    <pos or neg> <digits>
<pos or neg>    ::= + | - | <empty>
<digits>        ::= <digit> <digits> |
                    <digit>
<empty>         ::=
<digit>         ::= 0 | 1 | 2 | .. | 9
<string>        ::= '<ascii2> <morestring>'
<ascii2>        ::= << x|x is any ASCII character other than ' >>
<morestring>    ::= <ascii2> <morestring> |
                    <empty>
<types>         ::= TYPE <typedec> <types> |
                    <empty>
<typedec>       ::= <identifier> = <type>; <package> |
                    <identifier> = <filetype>;
<type>          ::= <chartype> |
                    <booltype> |
                    <inttype> |
                    <realtype> |
```

< rectype > |

< arraytype > |

< pointertype >

< filetype > ::= < filemode > < identifier >

< filemode ::= INFILE |

OUTFILE |

INOUTFILE

< chartype > ::= CHAR |

CHARACTER |

(< chrliteral >) |

(< chrsub >)

< chrliteral > ::= < string > < morechrs >

< morechrs > ::= , < chrliteral > |

< empty >

< chrsub > ::= < chrconst > .. < chrconts >

< chrconst > ::= < string > |

< identifier >

< booltype > ::= BOOL |

BOOLEAN

< inttype > ::= INT |

INTEGER |

(< subrange >)

< subrange > ::= < integernum > .. < integernum >

< integernum > ::= < pos or neg > < digits > |

< pos or neg > < identifier >

< realtype > ::= REAL |

(< realnum > .. < realnum >)

< realnum > ::= < pos or neg > < digits > . < digits > |

```
                    < pos or neg >  < identifier >
< rectype >         ::= RECORD  < varlist >  ENDRECORD
< varlist >         ::= < vardecl > ;  < morevardecl >
< vardecl >         ::= < identifier >  < moreident >  :  < type >  |
                        < identifier >  < moreident >  :  < identifier >
< moreident >       ::= , < identifier >  < moreident >  |
                        < empty >
< morevardecl >     ::= < varlist >  |
                        < empty >
< arraytype >       ::= ARRAY ( < arraysize > ) OF  < type of array > ;
< arraysize >       ::= < expr >  < morearray >  |
                        < expr > .. < expr >  < morearray >
< morearray >       ::= , < arraysize >  |
                        < empty >
< pointertype >     ::= ACCESS  < pointtype >
< pointtype >       ::= CHAR |
                        CHARACTER |
                        BOOL |
                        BOOLEAN |
                        INT |
                        INTEGER |
                        REAL |
                        < identifier >
< variables >       ::= VAR  < vardecl > ;  < package >  < variables >  |
                        VARIABLE  < vardecl > ;  < package >  < variables >  |
                        < empty >
< subprograms >     ::= < procedure >  < subprograms >  |
                        < function >  < subprograms >  |
```

```
                    <empty>
<procedure>     ::= PROCEDURE <identifier> <parameters>; <procdec>
<procdec>       ::= (<comment>); <decsec> BEGIN <identifier> <stmts>
                        <morestmts> END <identifier>; |
                    PACKAGE <identifier>;
<parameters>    ::= (<variable> <morevar> : <mode> <type>; <moreparams>) |
                    <empty>
                    <variable> ::= <identifier> |
                    <identifier> [<expr> <moreexpr>] |
                    <identifier>.<recvar>
<moreexpr>      ::= ,<expr> <moreexpr> |
                    <empty>
<recvar>        ::= <identifier> <morerecvar>
<morerecvar>    ::= .<recvar> |
                    <empty>
<morevar>       ::= ,<variable> <morevar> |
                    <empty>
<mode>          ::= IN |
                    OUT |
                    INOUT
<moreparams>    ::= <variable> <morevar> : <mode> <type>; <moreparams> |
                    <empty>
<function>      ::= FUNCTION <identifier> <funcparams> : <type>; <funcdec>
<funcparms>     ::= (<variable> <morevar> : <type>; <morefparams>) |
                    <empty>
<morefparams>   ::= <variable> <morevar> : <type>; <morefparams> |
                    <empty>
<funcdec>       ::= (<comment>); <decsec> BEGIN <identifier> <stmts>
```

< morestmts > END < identifier > ; |

PACKAGE < identifier > ;

< assignment > ::= < variable > := < expr >

< expr > ::= < simpexp > |

< simpexp > < relational op > < simpexp >

< simpexp > ::= < pos or neg > < term > |

< simpexp > < adding op > < term >

< term > ::= < factor > |

< term > < multiply op > < factor >

< factor > ::= < variable > |

< constant > |

(< expr >) |

< function call > |

NOT < factor > |

{ < comment2 > }

< comment2 > ::= < ascii3 > < comment2 > |

< ascii3 >

< ascii3 > ::= < < x|x is any ASCII character other than '}' > >

< relational op > ::= < > | < | < = | > = | > | =

< adding op > ::= + | - | OR

< multiply op > ::= * | / | AND | DIV | REM

< constant > ::= < number > |

< identifier > |

NIL |

< string >

< function call > ::= < identifier > (< actual param >) |

```
                    <identifier>
<actual param>      ::= <expr> <moreparam>
<moreparam>         ::= , <actual param> |
                        <empty>
<readstate>         ::= READ ( <readfile> <variable> <morevar> ) |
                        READLN <readcont>
<readfile>          ::= <identifier> , |
                        <empty>
<readcont>          ::= ( <readfile> <variable> <morevar> ) |
                        <empty>
<writestate>        ::= WRITE ( <writefile> <expr> <moreexpr> ) |
                        WRITELN <writecont>
<writefile>         ::= <identifier> , |
                        <empty>
<writecont>         ::= ( <writefile> <expr> <moreexpr> ) |
                        <empty>
<ifstate>           ::= IF ( <expr> ) THEN <stmts> <morestmts> <else> ENDIF
<else>              ::= ELSE <stmts> <morestmts> |
                        ELSEIF ( <expr> ) THEN <stmts> <morestmts> <else> |
                        <empty>
<whileloop>         ::= WHILE ( <expr> ) LOOP <stmts> <morestmts> ENDWHILE
<forloop>           ::= FOR <identifier> IN <expr> , <expr> LOOP <stmts>
                              <morestmts> ENDFOR |
                        FOR <identifier> INREVERSE <expr> , <expr> LOOP <stmts>
                              <morestmts> ENDFOR
<procedurecall>     ::= <identifier> |
                        <identifier> ( <actual param> )
<casestate>         ::= CASE <variable> IS <casebody> ENDCASE
```

< casebody > ::= WHEN < caselabel > = > < caseline > ; < morecase >

< caselabel > ::= < pos or neg > < number > |
< pos or neg > < realnum > |
< pos or neg > < variable > |
OTHER |
< string > |
< subrange > |
< realnum > .. < realnum > |
< chrsub >

< caseline > ::= < stmts > < morestmts > |
NULL

< morecase > ::= < casebody > |
< empty >

< emptystate > ::= ;

Appendix C. Project Data

Table 15. Design and Source Metric Values for Project: AKS

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	85	92	507	496	53104	72755	1	2	129600	357604
*	2	25	26	175	190	22625	42269	4	3	9216	14161
*	3	4	3	45	40	1296	1510	1	1	576	576
*	4	7	18	47	114	1032	12344	1	4	400	784
*	5	9	16	60	95	1408	6393	1	1	900	1764
*	6	5	10	43	68	1104	4019	1	1	784	576
*	7	3	7	12	20	202	407	1	1	16	25
	8	15	9	117	125	22699	29925	8	6	1600	324
*	9	7	4	56	52	4334	4570	4	3	2025	1296
*	10	9	5	72	68	8549	8437	5	4	9801	17424
	11	11	11	67	58	4949	4786	2	2	400	144
	12	14	15	116	111	10662	11764	2	2	625	256
*	13	8	7	49	47	3438	3301	1	3	1	1
*	14	5	9	28	43	435	849	1	1	9	36
*	15	11	8	60	74	3209	7526	1	2	400	25600
	16	4	8	19	45	286	1993	1	1	4	1
*	17	4	10	24	67	548	4046	1	1	4	16
*	18	11	11	71	69	3626	3745	1	1	9	36
*	19	7	4	50	46	5510	5566	4	3	36	36
*	20	22	33	149	201	22030	46541	7	8	17424	13689
	21	19	16	149	141	17483	17963	3	3	16900	3136
*	22	23	19	108	122	5570	17762	1	9	8100	33124
*	23	7	28	47	223	1032	60621	1	10	18496	81225
*	24	30	12	154	72	19781	4096	2	1	441	30276
*	25	3	37	19	195	468	24524	1	8	10000	9216
*	26	4	11	18	63	224	9885	1	3	36	576
	27	9	11	50	65	5270	7761	3	4	36	1
*	28	12	18	68	104	2681	6652	1	1	9	64
	29	21	16	153	171	28523	46988	9	7	3969	900
*	30	9	5	80	77	8204	10159	5	4	3969	2916
*	31	9	5	74	71	7474	6781	5	4	81	81
	32	17	16	107	101	17684	17425	6	5	144	64
*	33	4	7	19	37	340	3198	1	2	3136	2304
*	34	3	31	14	128	162	1649	1	3	64	5184
*	35	3	7	12	20	202	71	1	1	16	25
*	36	3	7	9	20	83	71	1	1	1	25
*	37	8	8	50	43	1337	1486	1	1	33124	17424
*	38	3	7	9	20	46	71	1	1	36	25
*	39	3	7	12	20	202	71	1	1	16	25
*	40	8	7	49	47	3438	3301	1	3	1	1
*	41	8	8	45	39	2365	2045	1	1	16	36
*	42	6	9	35	61	1046	5014	1	2	144	225
*	43	6	8	41	66	1417	4340	1	2	144	225
*	44	10	42	40	188	673	612	1	3	16	1024

Design and Source Metric Values for Project: AKS continued ...

Metric Values

	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	10	9	75	71	7459	7674	2	2	256	400
*	46	24	22	118	124	10867	24805	2	7	3969	14400
*	47	8	7	49	47	3438	3301	1	3	1	1
*	48	24	21	156	209	17862	33901	7	10	90000	131769
	49	6	6	35	30	1226	1054	1	1	256	36
*	50	8	6	45	41	2374	1860	1	1	256	225
*	51	8	6	45	45	2374	2042	1	1	81	400
*	52	8	6	45	39	2374	1769	1	1	81	144
*	53	10	7	54	55	3197	3782	1	2	16	81
*	54	6	6	35	30	1376	1054	1	1	12100	20736
*	55	8	6	45	41	2374	1860	1	1	81	400
*	56	10	7	54	47	3197	2989	1	2	81	400
*	57	10	6	54	47	3197	2132	1	1	100	324
*	58	8	6	45	41	2374	1860	1	1	81	324
*	59	8	6	45	45	2374	2042	1	1	225	256
*	60	10	7	54	47	3197	2989	1	2	81	400
*	61	8	5	51	52	4032	5393	4	3	16	16
*	62	8	6	54	53	5173	5497	4	3	36	36
*	63	10	10	74	70	2506	3628	1	1	576	576
*	64	10	12	76	75	3247	3490	1	1	576	576
	65	14	54	75	302	3168	63166	4	12	12100	5184
*	66	17	27	124	166	9110	29213	4	5	4900	10816
*	67	6	10	31	51	903	2330	1	1	81	625
*	68	6	10	31	51	903	2330	1	1	256	900
*	69	6	11	31	55	903	2383	1	1	225	900
*	70	6	10	31	51	743	2330	1	1	144	576
*	71	6	10	31	51	903	2330	1	1	64	576
*	72	6	11	31	55	903	2383	1	1	144	784
*	73	72	31	368	195	62823	33438	4	12	5776	10816
*	74	8	8	46	39	1889	1661	1	1	144	144
*	75	6	6	35	29	1226	1019	1	1	4225	3136
*	76	3	5	14	22	177	331	1	1	256	400
	77	5	13	24	100	328	6250	1	2	256	16

Table 16. Design and Source Metric Values for Project: CASINO

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	75	81	448	496	48395	51056	4	21	1254400	28206722
	2	27	287	166	1809	12883	348299	4	55	441	12320100
*	3	6	11	27	69	480	9016	1	1	64	1296
*	4	19	20	85	147	3006	25811	1	1	36	144
	5	3	29	9	143	47	10347	1	3	1	19600
	6	3	28	9	128	47	4875	1	1	1	400
*	7	11	34	50	167	1717	42175	1	4	81	4900
*	8	4	16	17	86	169	6924	1	3	1	36
*	9	3	18	9	84	84	14014	1	4	1	100
*	10	3	58	14	358	178	25343	1	20	36	2916
	11	5	61	23	379	412	91984	1	14	1	19044
	12	16	67	82	366	3655	154006	1	10	100	20736
	13	10	69	51	432	1361	172377	1	25	16	5929
	14	3	16	13	114	103	15568	1	2	1	196
*	15	69	106	376	722	49100	160596	7	24	9025	148225
*	16	3	16	15	118	149	20837	1	8	400	9801
*	17	4	9	19	38	295	1302	1	1	4	9
*	18	13	13	72	75	7032	6486	3	3	4	16
	19	35	34	127	138	7304	7386	1	11	100	49
*	20	3	5	9	20	84	537	1	1	16	64
*	21	3	5	9	20	84	537	1	1	1	4
*	22	23	24	166	185	17944	26377	1	1	400	784
*	23	3	3	15	24	190	864	1	2	25	100
*	24	4	9	15	46	250	1380	1	1	36	1225
*	25	16	14	126	122	12143	13251	2	2	1296	2025
*	26	5	28	18	100	332	2474	1	1	1	9
*	27	6	5	29	22	442	293	1	1	729	784
	28	3	4	9	19	84	1123	1	2	4	1
	29	5	4	24	17	430	197	1	1	400	2082249
	30	10	11	36	52	2056	3708	2	2	36	14400
*	31	6	11	30	66	660	6266	1	2	1764	2916
*	32	5	36	18	186	478	24261	1	3	1	25
*	33	3	161	14	1269	178	573837	1	1	1	9
*	34	34	37	167	191	29797	40319	5	5	324	1521
*	35	25	20	158	163	18461	31614	6	6	11664	32400
*	36	14	20	60	108	5903	19770	2	6	81	784
*	37	31	32	179	243	32681	71483	5	7	1600	10816
*	38	3	4	14	26	178	1208	1	2	1	64
*	39	3	6	14	36	178	1951	1	2	4	36
*	40	18	20	96	161	9000	45419	6	9	324	1225
*	41	42	8	145	38	23743	2435	2	3	16	100
*	42	3	31	14	196	233	37291	1	5	81	900
	43	68	327	712	2381	451028	410695	45	69	3600	101606400
	44	3	114	9	1392	84	1653826	1	50	1	12100

Design and Source Metric Values for Project: CASINO continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
45	26	9	143	40	22652	1167	6	1	2304	16
46	23	119	91	726	8939	137638	3	23	576	215296
47	3	26	9	119	47	19577	1	6	4	4356
* 48	3	6	9	33	84	1144	1	2	1	36
49	11	6	37	33	2113	1144	2	2	64	4
50	5	6	18	28	332	1531	1	1	1	1600
* 51	6	12	28	74	800	1459	1	1	9	16
52	3	7	9	42	84	959	1	1	1	441
* 53	5	67	24	424	430	114891	1	9	64	5929
* 54	3	17	14	82	233	9517	1	3	1	36
55	5	5	18	16	478	478	1	1	1	144
56	15	8	81	30	8305	742	5	1	144	36
57	3	26	14	128	178	26435	1	7	4	1764
58	19	8	131	69	15458	5297	7	1	1764	225
* 59	18	16	110	118	10927	15786	11	9	400	1936
* 60	6	47	32	285	844	127304	1	2	100	484
61	3	27	9	196	84	23724	1	12	16	18225
* 62	6	15	32	74	714	7625	1	2	100	8100
63	6	5	29	22	442	293	1	1	1	784
64	5	101	24	682	301	498089	1	10	16	7744
65	17	18	78	106	5580	12295	3	4	3969	25
* 66	4	107	10	384	52	24667	1	1	64	3600
67	3	56	14	300	178	33236	1	17	36	50176
* 68	28	82	129	482	12964	185713	9	22	33856	308025
* 69	25	36	131	214	14779	23377	9	5	5625	23716
* 70	24	32	103	165	11366	38434	4	4	400	20736
* 71	27	29	151	192	16360	35096	1	8	144	1936
72	12	917	77	6100	5479	2323096	3	203	2401	609448000
73	11	87	54	531	4503	96286	3	13	100	61009
74	4	19	19	87	295	10009	1	2	1	576
* 75	3	42	9	325	84	50106	1	1	1	36
76	3	264	14	1677	233	316359	1	98	1	87385104
* 77	21	26	100	162	10203	38530	1	8	36	3600

Table 17. Design and Source Metric Values for Project: CHART

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	75	44	375	221	45590	23143	7	2	3969	60025
*	2	70	55	410	513	41588	433984	1	9	1764	82944
	3	7	48	43	394	1373	132918	3	16	256	142884
*	4	10	17	59	142	3976	27092	2	5	81	3136
*	5	6	28	48	214	3897	51042	3	7	144	10000
	6	11	56	100	375	15000	44766	4	9	144	58564
	7	10	38	87	309	10900	86141	4	14	16	9604
	8	5	8	24	93	497	11494	1	5	3600	400
*	9	10	90	76	468	3782	57344	1	5	1225	38025
*	10	30	13	214	124	37924	9721	7	5	2304	5184
*	11	10	9	54	48	4269	4423	2	2	81	256
*	12	10	4	54	45	4269	4583	2	2	25	36
*	13	7	10	34	105	607	18025	1	4	64	144
	14	7	9	34	85	607	10912	1	4	64	16
*	15	11	8	73	57	3503	6664	1	4	1225	4624
*	16	13	19	110	156	9625	27331	3	8	784	3600
*	17	22	25	172	206	19272	50293	3	6	900	1225
	18	24	40	183	283	23302	105475	7	8	900	144
*	19	12	40	88	281	6384	109382	1	8	64	100
*	20	12	44	88	307	6384	53154	1	8	81	576
*	21	7	44	54	307	2798	53154	2	8	324	576
*	22	100	44	1171	346	760462	23586	27	13	44100	38025
	23	22	11	133	106	17837	11713	7	3	10000	784
	24	16	22	96	187	10800	28105	3	2	8100	900
	25	22	21	132	195	18686	34021	7	5	1936	784
	26	7	12	34	103	517	7563	1	1	11025	81
	27	7	12	34	103	517	7563	1	1	7225	64
*	28	10	19	73	147	6291	29202	4	7	900	576
*	29	37	19	441	147	124260	29202	3	7	900	576
	30	10	6	81	50	7413	2991	4	2	900	324
	31	8	94	55	1684	2248	3382523	1	41	144	65536
*	32	59	19	635	211	277637	59847	15	8	5184	8100
	33	7	15	34	107	517	12874	1	3	16384	3600
*	34	7	20	34	129	517	17789	1	6	11664	17424
*	35	36	18	242	134	50864	22475	12	5	28561	28224
*	36	24	18	172	134	27105	22475	8	5	3600	19044
	37	24	10	171	90	26128	13291	8	4	33124	900
	38	7	37	34	456	607	156937	1	2	4	900
	39	7	10	34	100	607	15256	1	4	4	900
*	40	7	8	34	47	607	2234	1	1	36	144
	41	7	43	34	631	607	380301	1	7	36	5184
	42	4	18	19	134	345	22475	1	5	64	42849
*	43	10	18	74	134	5158	22475	4	5	625	30625
*	44	8	32	78	286	4499	94025	5	10	400	14161

Design and Source Metric Values for Project: CHART continued ...

Metric Values

	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	45	6	26	29	247	441	60844	1	8	16	2304
	46	6	27	56	221	2952	44696	3	7	81	28224
	47	8	12	78	151	4499	49224	5	8	400	64
	48	6	19	29	147	441	29202	1	7	16	15876
	49	6	19	56	147	2952	27579	3	7	81	15876
	50	4	21	19	177	272	21085	1	6	1	6561
*	51	7	4	45	78	2889	5841	3	3	16	625
	52	5	53	24	503	426	224454	1	13	8100	625
	53	8	4	59	54	3990	3293	2	2	39204	81
*	54	8	3	58	76	4489	5691	2	3	81	576
	55	8	53	59	503	3990	224454	2	13	81225	625
*	56	8	4	58	54	4489	3293	2	2	81	81
	57	4	3	19	20	236	416	1	1	16	1
	58	5	13	28	85	643	11692	1	3	144	16
	59	3	11	14	73	177	9063	1	5	1	10000
	60	4	7	24	55	657	4432	1	2	16	29241
*	61	5	8	32	56	1726	4922	2	2	1	81
*	62	30	7	207	55	17445	4432	6	2	33124	65536
	63	6	8	40	56	2184	4922	2	2	324	81
*	64	9	41	69	312	6203	72644	3	10	81	64
	65	15	72	109	588	12384	124094	3	33	3025	357604
	66	8	11	61	92	4169	11431	4	4	4900	225
*	67	5	13	35	86	1490	8204	2	2	36	441
	68	5	4	35	28	1490	1512	2	2	36	1
	69	29	11	181	59	23760	2912	5	1	12100	2304
*	70	3	6	14	39	177	2319	1	2	25	225
	71	8	7	62	67	6485	7273	3	3	900	81
*	72	6	5	57	57	6346	4808	3	3	324	4225
	73	6	5	53	42	4707	2809	3	2	900	36
	74	6	5	53	42	5311	2809	3	2	576	36
*	75	4	15	24	107	548	18271	1	2	225	625
*	76	10	35	71	295	6143	135959	2	12	81	8100
*	77	8	8	52	53	3774	4947	2	2	16	16
*	78	8	8	56	61	4973	6129	3	3	16	625
*	79	11	6	93	59	8693	6898	2	3	36	256
	80	8	5	49	51	3833	4568	3	3	1	576
*	81	10	5	95	51	8856	5261	1	3	144	441
	82	28	8	184	46	18996	2692	7	2	20449	144
	83	103	10	584	110	66547	12778	43	2	127449	256
*	84	12	22	77	241	6407	69728	4	8	36	900
*	85	6	6	29	52	671	4657	1	3	36	100
	86	12	10	73	80	7253	5888	5	2	144	36
	87	9	7	44	44	1144	3237	1	3	100	1
*	88	13	10	60	105	4759	13355	6	2	25	100

Table 18. Design and Source Metric Values for Project: CLUE

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	223	282	1193	1361	115594	193061	1	14	4494400	36000000
	2	3	5	14	39	233	2097	1	1	64	9801
*	3	3	4	14	31	233	1334	1	1	36	324
*	4	20	11	105	61	11686	6954	5	4	254016	525625
*	5	27	17	116	93	9480	7146	1	9	324	225
*	6	10	6	60	65	2829	5254	5	5	784	576
	7	6	5	35	29	817	662	1	1	1600	144
	8	35	9	160	58	13847	3847	1	3	100	4
*	9	7	13	53	87	3052	7318	1	7	9	25
	10	35	9	160	58	13847	3847	1	3	256	81
*	11	16	12	115	108	14951	20157	5	5	1600	900
	12	21	10	120	107	19187	18279	7	11	400	144
	13	11	15	72	93	1797	4105	1	1	6400	225
	14	10	10	50	44	1206	1511	1	1	4356	900
	15	3	9	14	54	178	3664	1	3	4	1
	16	3	9	14	78	178	7260	1	2	4	1
	17	3	11	14	101	178	10524	1	3	4	1
	18	3	7	14	68	178	5225	1	2	4	1
	19	14	152	69	947	1807	104967	1	61	8100	3111696
	20	30	29	206	208	32719	41324	4	5	5184	900
	21	12	12	98	101	14840	19319	3	3	81	36
	22	13	13	81	76	7553	7058	2	2	625	144
*	23	13	13	104	103	13884	13778	2	4	64	36
	24	8	10	50	52	2174	2365	1	1	484	36
	25	12	12	92	89	15394	13097	4	4	64	1
	26	11	12	62	64	2791	4745	1	3	100	4
	27	3	27	14	151	178	42156	1	6	4	1
	28	17	19	134	136	12332	22996	3	4	11025	1936
	29	8	7	71	66	4733	5296	2	2	729	144
	30	8	10	50	52	2174	2365	1	1	324	16
	31	12	12	92	89	15394	13097	4	4	64	1
	32	8	13	71	118	4733	15825	2	4	1024	324
	33	8	10	50	52	2174	2365	1	1	324	16
	34	12	12	92	89	15394	13097	4	4	64	1
*	35	10	14	86	141	8066	21175	2	2	16	9
	36	10	9	70	69	3361	3960	3	2	144	30976
*	37	9	9	71	81	3923	6480	2	2	900	1296
*	38	7	9	50	105	2390	11752	1	3	36	36
*	39	14	13	141	128	12313	10884	1	1	81	81
*	40	23	17	175	162	27425	28811	5	5	16	16
*	41	9	5	60	82	3257	6203	5	7	1	1
*	42	10	11	75	73	3803	4178	1	1	256	256
*	43	15	16	168	205	31254	55268	6	9	2304	2304
*	44	14	11	135	136	25273	35859	6	5	576	576

Design and Source Metric Values for Project: CLUE continued ...

Metric Values

	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	7	13	53	87	3052	7318	1	7	64	144
*	46	14	11	135	136	25273	35859	6	5	441	441
*	47	7	13	53	87	3052	7318	1	7	64	144
	48	6	19	29	147	441	29202	1	7	16	15876
*	49	15	10	126	107	20975	16029	5	4	9	9
	50	30	108	226	972	50278	175269	7	44	2916	4000000
*	51	13	25	98	209	11655	27566	4	11	100	7056
*	52	12	13	85	100	5722	8382	1	3	441	1089
*	53	16	18	169	205	26137	34971	15	18	576	1600
	54	19	16	160	177	27997	34878	6	7	4	1
*	55	5	15	24	127	363	16304	1	1	810000	5517801
*	56	3	3	14	23	233	742	1	1	36	196
	57	31	34	226	263	34565	66566	5	7	89401	17424
*	58	20	16	129	118	18964	20838	7	6	6400	12544
*	59	11	13	57	112	3267	17743	3	6	64	64
*	60	9	9	63	56	3466	2958	1	1	81	784
*	61	7	25	34	272	704	63592	1	18	16	256
*	62	17	51	84	337	5888	31050	2	41	400	1764
*	63	8	3	54	46	4644	3517	6	4	9	9
*	64	6	3	35	26	1873	1219	3	1	16	16
*	65	10	8	48	44	1981	2322	1	3	9	9
*	66	10	8	60	52	4531	3694	3	2	256	784
*	67	21	82	106	602	8556	126533	2	49	324	20736
	68	81	64	400	448	30578	59821	4	36	48841	2304
	69	11	6	57	51	1983	2211	5	3	8100	3969
	70	4	3	49	43	2718	2134	3	3	400	144
*	71	10	50	62	319	3233	24177	1	31	225	576
*	72	13	58	107	388	8625	36205	3	33	400	23409
	73	28	22	229	204	22437	33043	6	5	76176	441
	74	11	7	62	49	3445	3382	1	3	1225	4
	75	23	13	138	95	14657	8514	4	4	256	64
	76	21	31	85	187	5091	11341	1	28	27225	144
	77	23	13	138	95	14657	8514	4	4	144	36
	78	3	3	14	24	178	942	1	2	1	144
*	79	16	12	142	148	17036	26083	6	5	4900	9604
*	80	10	9	75	74	4830	6017	3	3	2304	2304
*	81	17	17	123	123	19091	17528	8	8	64	64
*	82	17	17	123	123	19091	17528	8	8	81	81
*	83	27	19	176	167	33562	32703	11	11	144	144
*	84	8	8	55	50	1858	1625	1	1	225	225
	85	21	33	85	190	5091	11523	1	28	81	4
*	86	35	11	160	60	13847	3746	1	3	49	36
*	87	20	25	168	215	17309	35124	5	5	2025	2025
*	88	11	9	62	53	3445	3659	1	3	4	4

Design and Source Metric Values for Project: CLUE continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	89	8	13	61	95	5128	14339	1	3	324	576
*	90	23	16	138	121	14657	13153	4	6	36	36
	91	21	35	85	200	5091	12472	1	29	784	144
*	92	10	10	76	69	5438	4501	1	1	25	25
*	93	10	8	86	78	5426	6630	3	3	3136	6400
*	94	20	12	135	124	17816	15456	5	8	16	64
*	95	13	11	133	164	22542	32991	6	5	36	64
*	96	18	16	118	118	10688	12031	3	3	256	256
*	97	9	12	58	99	4646	10187	2	2	36	36
*	98	11	8	198	194	57561	57964	12	12	16	36
*	99	23	16	165	159	30835	28620	9	5	1	1
*	100	16	18	105	130	8196	17642	2	5	8281	9604
	101	12	5	92	28	13674	1456	4	1	16	4
	102	14	56	63	299	4014	25891	1	16	256	40000
*	103	3	5	9	46	84	3031	1	3	4	4
*	104	4	11	19	60	295	6368	1	4	219024	2547216
*	105	3	4	14	31	233	1334	1	1	36	2025
*	106	3	4	14	31	233	1334	1	1	1	1

Table 19. Design and Source Metric Values for Project: DUNGEON

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	170	325	843	2398	124349	779522	4	87	3701776	40246336
	2	3	2	15	30	190	1361	1	2	20736	6561
*	3	10	5	38	42	1808	1581	1	10	144	144
*	4	28	51	200	387	20635	115597	1	9	4624	26244
*	5	6	6	47	38	2829	1743	1	1	2304	1296
*	6	28	25	166	191	23563	45296	9	10	25600	51984
*	7	10	6	49	45	2764	3528	1	4	289	25600
*	8	14	25	104	240	9057	56363	4	8	30976	116964
*	9	33	34	134	157	19752	35438	2	11	1296	3136
*	10	3	3	14	53	233	3014	1	2	12544	370881
*	11	14	10	79	70	9480	6769	5	5	20449	213444
*	12	9	7	58	55	4679	3679	1	3	1089	676
	13	9	9	48	68	1901	4606	1	2	14641	3136
	14	4	64	22	471	552	113629	1	2	676	121
*	15	66	31	520	318	161205	116475	1	19	256	3844
*	16	18	56	118	666	9430	415942	1	9	3136	11664
	17	53	70	543	285	192443	34597	5	17	43264	1
	18	41	70	177	285	16327	34597	1	17	100	1
	19	41	70	177	285	16327	34597	1	17	144	1
	20	41	70	177	285	16327	34597	1	17	225	1
	21	41	4	177	18	16327	258	1	1	256	1
*	22	4	8	19	55	204	6254	1	4	25921	30976
	23	3	26	14	143	178	8504	1	11	169	171396
*	24	15	15	149	191	21113	42441	5	8	7056	6084
*	25	14	13	87	128	8382	18708	4	5	1444	15876
*	26	19	14	165	161	22316	30636	8	6	12100	11664
*	27	67	30	1117	975	1565220	1153918	60	51	39204	97344
*	28	26	22	217	264	40943	76108	9	8	1296	2401
	29	8	6	52	60	3110	4287	3	4	33124	1024
*	30	3	8	9	36	47	844	1	1	1	1
*	31	30	22	243	187	39881	37228	7	7	28900	46656
*	32	12	12	110	147	12294	22549	5	6	9216	7056
*	33	22	15	168	163	26117	21940	1	9	116964	129600
*	34	18	13	169	170	24380	33712	7	5	6561	4096
	35	156	54	823	428	143916	169174	41	35	22838842	4264225
*	36	5	6	34	49	1473	3325	1	2	1	1
*	37	6	6	31	59	797	5681	3	3	1	4
*	38	6	6	31	59	797	5681	3	3	1	4
*	39	6	6	35	44	1333	2706	1	2	1	1
*	40	6	7	44	40	1551	3823	1	2	9	9
*	41	6	12	29	127	662	18320	3	6	1	4
*	42	3	4	16	16	239	310	1	1	1	1
*	43	6	13	29	131	662	20689	3	6	1	4
	44	20	21	188	193	27687	49012	1	1	49	25600

Design and Source Metric Values for Project: DUNGEON continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 45	3	3	29	18	677	323	1	1	36	100
* 46	3	3	29	18	677	323	1	1	25	144
47	20	21	188	193	27687	49012	1	1	1	23716
* 48	3	3	29	18	677	323	1	1	5184	14400
* 49	21	16	125	113	14119	14023	3	4	104976	147456
* 50	13	14	120	140	15189	21823	3	5	20449	90000
* 51	8	11	40	63	922	3844	1	2	48400	74529
* 52	7	11	183	98	14601	5797	1	1	5929	48400
* 53	13	9	137	137	22531	25888	7	7	327184	810000
* 54	5	9	24	68	363	7200	1	5	455625	921600
55	4	30	19	184	237	27065	1	9	1936	1849600
56	6	4	29	28	460	574	1	1	81	50625
* 57	3	6	9	37	84	1452	1	1	1	1
* 58	8	6	74	66	8839	9305	3	3	4624	12996
* 59	4	4	19	51	264	5284	1	2	234256	1132096
* 60	10	5	38	42	1808	1581	1	10	144	144
* 61	6	6	29	45	662	3479	3	3	1	1
* 62	6	6	29	45	662	3479	3	3	1	1
* 63	9	10	63	69	3786	5221	2	3	4	9
64	27	29	177	270	26821	83530	9	15	256	4
* 65	13	15	118	163	12176	29512	3	4	16	16
* 66	22	30	179	305	25419	89478	4	10	1225	3600
* 67	21	25	199	312	33259	93022	6	12	21609	33856
* 68	9	10	65	68	4398	5963	2	2	9	25
* 69	15	17	129	141	17233	26715	4	5	36	36
* 70	15	14	129	128	16347	22207	4	4	36	36
* 71	18	16	147	144	22587	29460	5	5	36	36
* 72	4	6	17	26	329	713	1	1	1	4
* 73	14	13	124	115	14868	17783	4	3	36	36
* 74	4	6	22	63	456	5266	1	2	784	4096

Table 20. Design and Source Metric Values for Project: FORM

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	49	61	339	304	35367	35111	3	2	99225	254016
*	2	13	10	90	53	6300	3888	2	2	196	900
*	3	17	7	126	70	17311	9868	5	3	144	225
*	4	4	4	19	19	294	591	1	1	16	144
*	5	6	5	33	26	933	866	1	1	324	625
*	6	4	5	19	22	294	684	1	1	1	81
	7	3	108	14	723	177	114885	1	26	1	4840000
*	8	13	6	55	33	1505	966	1	1	4	16
	9	129	281	797	2020	75808	464760	34	76	48441600	22325626
	10	13	172	93	1296	13553	264780	3	56	1521	1896129
	11	7	5	37	47	1979	4750	2	3	576	4
	12	4	6	19	33	286	1201	1	2	900	100
	13	22	6	194	26	56124	596	10	3	7056	2500
*	14	3	23	18	131	415	13353	1	10	16	16
*	15	5	5	26	28	615	710	1	1	625	576
*	16	3	3	18	16	415	242	1	1	900	50176
	17	4	3	19	38	438	3026	1	2	16900	3136
	18	18	135	105	1047	2476	169440	5	87	81225	11664

Table 21. Design and Source Metric Values for Project: GAMMON

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	45	311	317	2467	40448	488786	6	137	864900	1494440960
	2	9	205	46	1871	1249	826289	1	92	64	2265025
*	3	29	23	207	212	23832	29824	9	8	34969	65025
*	4	11	7	67	58	6377	4580	4	3	784	1225
*	5	6	9	24	70	1893	7960	3	6	4	16
	6	36	7	233	34	43987	588	9	1	17424	7744
*	7	3	2	15	23	190	865	1	1	16	9
	8	34	41	226	380	34253	118961	5	15	9801	2025
*	9	4	9	19	110	237	15100	1	9	28224	65025
	10	6	22	31	188	646	33993	1	7	81	36
*	11	5	12	25	162	381	30327	1	13	25	324
*	12	4	13	19	110	237	16500	1	6	16	9
	13	5	12	23	182	289	43614	1	17	1024	196
*	14	3	7	14	38	178	1432	1	1	100	6400
*	15	5	16	24	145	414	23625	1	4	1024	15876
*	16	5	14	24	149	363	25778	1	9	256	2704
*	17	3	12	14	65	178	3270	1	3	225	625
*	18	3	7	9	58	84	2850	1	1	1	16
*	19	3	7	9	58	84	2850	1	1	1	81
*	20	3	7	9	58	84	2850	1	1	1	100
	21	3	7	9	58	84	2850	1	1	1	121
	22	5	13	24	139	363	23489	1	8	400	116964
	23	4	8	19	50	237	2141	1	3	16	24336
*	24	5	13	24	139	363	23489	1	8	4	36
	25	23	193	241	2102	43608	514409	15	109	65025	37945600
*	26	7	106	36	1245	1054	532020	1	53	4356	8281
*	27	9	109	46	1283	1362	587655	1	54	1296	11664
*	28	7	106	36	1271	1054	601110	1	54	4900	27225
*	29	6	15	31	316	641	87566	1	19	4096	33124
*	30	6	15	31	402	641	138816	1	25	3969	43264
*	31	4	6	19	75	237	8445	1	6	3600	13689
*	32	4	3	19	53	237	3309	1	5	4225	4900
*	33	18	14	97	138	12332	19517	7	10	1521	9604
*	34	7	3	38	43	1924	2091	4	4	28224	28224
*	35	9	106	46	1273	1362	602056	1	54	676	18496
*	36	7	106	36	1273	1054	602056	1	54	8281	39204
*	37	30	37	150	224	13931	40025	9	12	400	3025
*	38	3	4	9	12	84	36	1	1	4	400
*	39	3	27	9	140	84	31477	1	6	1	9
	40	3	21	14	123	233	29321	1	3	1	324
*	41	3	27	9	140	84	31477	1	6	16	36
*	42	17	35	106	248	9500	66430	3	11	81	2304
*	43	6	10	24	58	482	3909	1	2	64	144
*	44	7	94	19	487	177	90047	2	6	81	1024

Design and Source Metric Values for Project: GAMMON continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 45	17	14	82	94	3252	6307	1	1	256	784
* 46	3	4	14	34	178	1886	1	2	4	4
47	3	6	14	32	178	640	1	1	25	5184
48	5	16	24	158	426	39548	1	16	36	1
* 49	8	16	34	207	704	48728	2	7	324	5929
* 50	5	5	24	39	426	2191	1	2	144	400
51	27	83	215	634	31587	138386	12	31	1764	202500
* 52	9	10	46	188	2536	60186	3	9	64	784
* 53	17	21	86	112	2692	8833	1	2	6084	78400
* 54	6	30	33	223	605	35215	1	3	36	576
55	9	31	71	341	5340	65096	5	21	16	15876
* 56	14	14	53	66	1538	6335	2	3	1225	1296
* 57	3	14	9	79	84	8362	1	3	1	1
* 58	3	10	14	64	178	6538	1	3	1	4
* 59	3	10	14	64	178	6538	1	3	1	4
* 60	5	23	24	257	363	77317	1	11	784	11664
61	35	74	217	624	30207	221568	11	31	14400	9090225
62	10	29	61	444	2040	94212	1	26	625	1327104
* 63	38	46	247	432	29621	186025	14	22	28224	1249924
64	19	78	135	688	12742	182546	7	39	3136	11628100
* 65	8	7	49	44	3250	2649	2	2	81	1296
* 66	13	22	74	150	4902	17888	1	1	81	5184
* 67	15	11	99	125	8562	19101	5	6	2025	18225
* 68	6	6	38	40	2472	2393	2	2	4	16
* 69	10	9	60	53	3724	2961	1	1	9	81
* 70	16	22	226	327	62734	130267	3	3	400	9801
* 71	5	23	25	158	571	26097	1	2	2304	50625
72	7	8	34	50	624	2221	1	2	25	2601
* 73	12	10	72	72	5680	6853	4	3	576	14161
* 74	9	6	58	82	5194	11140	3	6	4	4

Table 22. Design and Source Metric Values for Project: GOLF

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	55	332	376	3432	38966	1620545	1	191	160000	159719040
*	2	10	45	82	258	3420	25641	1	3	3969	6561
*	3	5	17	28	116	458	5324	1	1	1	9
*	4	3	26	13	125	130	4592	1	1	4	36
*	5	13	38	60	206	6020	28722	5	5	9	225
	6	23	30	127	159	22408	36398	6	5	225	100
	7	5	6	26	40	533	2420	1	2	9	1
*	8	5	7	20	43	358	2819	1	2	4	4
	9	6	8	29	40	607	1417	1	1	1600	196
*	10	14	39	56	181	4237	18396	2	3	81	81
	11	7	55	44	338	1156	79999	1	13	16	2500
*	12	3	31	9	136	84	488	1	1	4	4
	13	6	19	59	103	1926	4350	1	1	9216	3136
*	14	3	52	14	320	178	81316	1	28	4	324
*	15	30	64	114	362	15158	67197	4	19	256	21609
	16	7	120	39	644	2528	49113	2	17	64	88209
	17	8	17	91	117	7075	8775	2	2	900	81
*	18	22	25	197	201	23113	21767	5	3	2025	3136
	19	12	73	62	542	1684	105410	1	30	16	32400
	20	4	87	23	495	627	71936	1	19	4	439569
*	21	6	71	44	651	951	197300	1	21	324	7744
*	22	4	25	18	128	272	11581	1	11	1	25
*	23	4	84	23	505	660	115937	1	15	4	400
	24	3	37	12	175	162	7093	1	3	1	256
*	25	8	99	54	404	1412	13544	1	4	400	3025
	26	5	93	27	594	555	82579	1	26	81	152100
*	27	34	23	151	129	19661	18907	3	11	16	400

Table 23. Design and Source Metric Values for Project: MONOPOLY

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	94	565	518	4634	67723	1625705	1	174	82944	1075840000
*	2	4	4	19	12	438	252	1	1	256	1296
*	3	4	4	19	12	438	126	1	1	81	625
*	4	5	7	25	35	520	2723	1	2	10000	188356
*	5	4	4	19	12	438	252	1	1	81	81
*	6	121	120	419	871	85434	1031709	1	32	14400	62500
*	7	31	32	204	286	46700	137252	18	19	1600	1936
*	8	129	117	456	858	94639	973964	1	31	18225	140625
*	9	13	11	72	65	9408	11015	4	4	1102500	15681600
*	10	70	70	246	282	73800	94483	1	23	256	196
*	11	124	127	426	510	219816	306149	1	42	3600	28900
*	12	29	41	160	226	8989	33887	1	1	2916	14641
	13	50	5	484	41	148104	2682	1	3	16	1
*	14	16	34	86	232	8732	56129	1	7	36	36
*	15	16	27	86	178	8030	35308	1	6	36	36
	16	18	142	96	1017	6463	380700	1	14	36	24336
*	17	4	60	15	331	190	72656	1	6	1	1
*	18	4	41	19	237	284	93107	1	14	1	4
*	19	5	4	20	12	461	252	1	1	10000	65536
*	20	4	4	19	12	438	126	1	1	10000	65536
*	21	36	65	190	452	41785	330444	11	29	692224	659344
*	22	13	10	105	74	11458	5775	1	1	9144576	92352104
*	23	28	25	132	166	19839	58521	7	6	176400	1669264
	24	13	11	65	58	7887	11280	4	4	36	4
	25	51	170	292	1378	33418	649944	14	78	82944	497468416
	26	13	16	65	156	5428	28412	4	12	441	4
	27	51	97	364	779	73920	392310	17	23	32400	18879026
	28	9	35	49	272	1535	80589	1	12	171396	729
*	29	25	121	167	1050	32094	298860	10	47	781456	1155625
	30	71	164	499	1389	91736	1417721	42	56	614656	116791248
*	31	14	58	58	553	1899	334524	1	19	28900	925444
*	32	38	49	158	348	8969	109091	13	17	70756	112896
*	33	11	53	54	418	2600	257868	3	17	9216	132496
*	34	15	37	113	370	8502	54497	4	23	14400	636804
*	35	14	14	97	109	11233	13788	5	4	1225	1600
*	36	15	28	112	232	14760	72419	3	6	64	576
*	37	5	40	20	387	249	87331	1	28	9	225
*	38	11	44	101	453	12446	72046	8	34	2916	116964
	39	6	16	33	95	1360	11076	1	2	25	5184
*	40	10	7	66	55	3601	3797	6	5	36	144
*	41	11	9	84	79	12619	10919	5	5	36	144
*	42	9	9	63	77	6330	9068	4	5	64	225
	43	8	173	58	1652	2757	467585	4	96	900	2650384
*	44	12	8	87	91	13340	11727	7	8	16	81

Design and Source Metric Values for Project: MONOPOLY continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	45	4	44	23	529	350	262927	1	27	9	3969
*	46	12	8	88	83	14205	10362	7	6	9	9
*	47	4	27	19	237	288	77691	1	12	64	1764
*	48	6	12	28	242	766	53732	1	15	25600	417316
*	49	105	127	739	1263	270680	1512316	59	37	48400	200704
*	50	41	55	291	448	50607	197355	20	11	11664	44100
*	51	21	175	91	1224	3933	778124	4	49	9216	722500
	52	22	19	100	106	7751	12837	1	3	9	14641
	53	51	359	227	2617	19537	1057156	13	127	104329	27720226
*	54	31	57	126	390	6581	164298	10	23	46656	102400
	55	16	39	86	225	3721	47318	5	7	541696	74529
*	56	14	81	66	602	4237	257128	5	32	1296	32400
*	57	21	50	87	346	3330	111870	6	15	94864	263169
*	58	11	15	53	77	3147	2407	3	1	1089	8100
	59	12	13	50	92	1938	10834	1	6	9	17424
*	60	14	20	75	94	7199	298	6	1	1	36
*	61	4	31	10	115	93	2837	1	1	1	1
	62	13	11	72	64	9408	10120	4	4	4900	9090225
*	63	15	13	81	74	10334	9161	4	4	9000000	652853632
*	64	7	6	39	32	3047	1344	1	1	324	1764
*	65	33	29	212	256	53384	114981	19	19	2025	18496
*	66	18	15	120	137	18771	36613	10	10	225	900
*	67	8	6	53	43	2431	2152	1	1	4900	490000
*	68	17	12	85	86	6538	14506	4	4	17424	9025
*	69	4	4	19	12	438	126	1	1	625	4096
*	70	4	4	19	12	438	252	1	1	625	4096

Table 24. Design and Source Metric Values for Project: MONOPOLY2

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	106	771	615	5941	50395	1421461	2	281	2500	-494973952
*	2	16	29	121	266	19465	78783	6	13	49	2601
*	3	18	39	106	283	8658	87037	3	14	900	23716
*	4	18	37	141	275	16447	69137	6	7	400	2401
*	5	15	36	113	269	12684	66640	6	7	900	3136
	6	8	14	34	86	1196	7554	3	2	16	9801
*	7	17	31	147	297	15936	100311	10	11	625	3600
*	8	4	54	19	518	236	93094	1	77	784	23716
	9	3	3	17	25	263	736	1	2	400	104329
	10	3	96	13	970	130	130376	1	89	1	5688225
	11	15	19	74	243	3574	100684	4	12	25	25600
*	12	5	15	28	94	508	9125	2	4	1	64
	13	4	81	18	454	224	52479	1	17	1	92416
*	14	3	34	14	237	177	44921	1	10	49	1764
*	15	20	25	92	220	4647	56340	1	11	900	35721
	16	4	5	18	28	224	458	1	1	144	33124
*	17	22	5	161	28	17601	458	10	1	11664	27225
	18	7	14	36	102	1267	10387	3	4	1	225
*	19	4	15	28	102	708	9485	1	3	225	22500
*	20	8	5	51	47	2240	3743	4	3	4	324
*	21	5	4	28	22	559	447	1	1	441	484
*	22	10	54	46	465	1943	50183	3	43	25	1024
*	23	7	23	36	165	900	12411	1	7	1296	93636
*	24	7	6	43	59	3252	5681	2	4	256	1024
*	25	5	22	32	138	755	25755	1	7	144	256
*	26	5	3	30	24	1394	2970	3	2	1296	4225
	27	8	14	49	112	3740	11055	3	2	1	441
*	28	13	15	97	144	7658	19552	1	2	1	4
*	29	11	11	53	49	1280	1800	1	1	81	3025
*	30	4	22	19	137	474	17830	1	7	1	9
*	31	3	19	14	126	177	26054	1	4	1	4
*	32	20	49	115	258	7045	57051	1	8	3136	19600
*	33	3	29	11	209	185	66353	1	12	576	361
	34	7	14	42	239	2315	19092	4	39	1	3969
*	35	5	15	28	122	1115	24063	2	6	64	4096
*	36	16	4	84	21	5296	320	3	1	576	20736
	37	4	12	25	73	659	3989	1	1	1	196
*	38	7	18	62	131	4768	22156	2	3	81	441
	39	4	3	20	12	369	159	1	1	900	2190400
	40	4	32	18	339	269	132576	1	46	900	700569
	41	5	9	24	61	633	5145	1	3	100	11025
*	42	11	21	54	125	1805	13917	1	2	900	3600
	43	3	89	9	432	83	38920	1	5	16	2025
*	44	3	30	14	230	177	40262	1	31	16	16

Design and Source Metric Values for Project: MONOPOLY2 continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	8	15	51	93	2999	12089	2	3	144	1764
	46	3	33	14	197	177	33623	1	7	1	784
*	47	3	3	22	18	354	226	1	1	144	1936
*	48	4	3	32	27	832	594	1	1	4096	48400
*	49	3	11	11	56	185	2408	1	2	9	25
	50	21	34	104	245	14276	87063	4	8	256	28900
	51	3	37	15	240	190	58268	1	15	36	4225
	52	4	58	19	374	286	138261	1	13	1	15876
*	53	12	12	67	73	4355	7120	3	2	2304	20736
*	54	4	8	29	52	870	1735	1	1	4	36
*	55	3	5	14	36	310	1739	1	1	9	36
*	56	13	29	100	225	10978	54151	4	8	225	324
*	57	4	35	25	360	571	34482	3	7	16	256
	58	3	38	14	304	177	28621	1	7	4	33124
*	59	3	16	15	112	187	14486	1	4	4	144
*	60	4	17	21	175	431	36311	1	6	5929	30976
*	61	4	6	19	38	350	3987	1	2	900	14641
*	62	4	10	19	54	236	2780	1	1	576	729
	63	9	10	48	62	2887	5306	3	3	196	90000
*	64	32	37	190	318	33426	136404	12	21	1600	28224
	65	4	125	19	837	236	223873	1	27	1	171396
*	66	11	10	56	73	4621	11181	2	3	36	196
	67	3	36	9	154	83	22375	1	8	1	144
	68	8	12	42	69	1921	9315	1	3	100	36

Table 25. Design and Source Metric Values for Project: MORLOC

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	58	136	320	917	24575	199931	2	17	12100	1658944
	2	7	210	24	1557	443	357496	4	15	9	2190400
	3	12	404	37	1965	983	470494	4	33	1	3504384
*	4	12	9	110	68	6583	2915	1	1	6561	277729
	5	33	134	209	1091	27965	344437	2	38	54756	1683143936
	6	5	45	29	407	680	53510	1	13	1024	213444
*	7	5	17	38	249	1478	66684	2	8	400	23716
*	8	3	52	9	249	84	54518	1	3	1	100
*	9	3	9	18	67	258	4960	1	3	1225	2401
*	10	13	28	70	417	2895	89323	7	10	2304	112896
	11	6	39	35	641	832	148511	1	12	121	2669956
	12	6	39	35	649	920	141685	1	13	1024	2965284
	13	6	39	35	641	832	152777	1	12	729	2689600
	14	6	39	35	647	832	149965	1	13	400	2396304
*	15	12	36	61	294	3022	49472	5	11	1764	40000
*	16	6	13	39	221	1855	31857	3	6	729	21609
*	17	4	6	30	87	703	9037	1	5	144	324
	18	16	225	96	1385	5915	457695	8	23	25	142884
	19	18	62	85	638	3473	86522	11	22	25	1512900
	20	5	59	28	629	576	98139	1	26	81	256036
*	21	9	15	67	171	3123	13671	4	6	576	41616
*	22	7	27	32	238	934	25745	3	10	36	3025
	23	7	18	50	154	3701	26831	4	8	4	784
*	24	12	44	96	255	4444	22476	1	7	2304	40000
	25	3	49	27	414	824	51398	1	19	4	18225
*	26	3	4	19	41	409	1840	1	2	1	1
*	27	3	20	13	209	130	29853	1	10	9	100
	28	4	56	23	505	547	78830	1	23	9	92416
*	29	5	14	41	155	1207	19755	1	4	36	324

Table 26. Design and Source Metric Values for Project: MTCLIMB

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	79	183	548	1147	89996	205349	5	15	1401856	9144576
	2	3	53	11	410	185	156081	1	10	4	900
	3	40	15	180	334	11137	284836	1	16	144	16
*	4	6	8	29	70	527	7769	1	3	4	225
*	5	9	30	45	173	1204	27477	1	4	16	1296
*	6	23	32	96	205	10291	22240	2	16	36	900
*	7	3	21	14	160	178	22982	1	7	1	64
	8	39	15	251	69	41106	3164	8	1	3136	324
	9	29	80	237	392	55510	89945	5	23	441	140625
	10	28	26	168	140	31139	41364	10	7	36	4900
	11	4	19	19	162	345	25113	1	10	16	2304
	12	4	49	19	214	284	41189	1	8	4	9801
	13	4	44	19	280	288	53407	1	7	16	8100
	14	9	57	44	581	892	226810	1	16	16	5929
	15	81	37	442	162	117912	13664	12	11	20449	100
*	16	5	82	24	739	414	403515	1	13	900	2025
*	17	13	85	34	621	1048	270782	1	23	2304	57600
*	18	3	52	9	206	84	3207	1	1	1	16
*	19	3	59	9	295	84	136706	1	1	1	16
*	20	3	102	9	518	84	449118	1	1	1	16
*	21	3	107	9	604	84	240157	1	3	1	16
	22	3	31	9	139	84	19824	1	3	1	324
*	23	5	29	24	205	414	30802	1	9	2025	4356
*	24	5	11	24	94	414	11052	1	3	784	2025
*	25	4	11	19	147	345	22933	1	9	64	4900
*	26	4	17	19	273	237	47841	1	19	324	5625
*	27	4	5	19	37	351	2200	1	2	441	2916
*	28	4	10	19	85	345	13534	1	2	576	1600
*	29	5	14	24	149	414	34122	1	2	576	2304
*	30	6	108	29	1094	583	767270	1	37	81	4356
*	31	3	102	9	747	84	862785	1	7	1	36

Table 27. Design and Source Metric Values for Project: OTHELLO

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	76	342	425	2310	22338	499783	19	70	104329	79744904
	2	14	5	51	45	630	3612	4	3	576	1
*	3	5	5	33	19	663	469	1	1	36	1600
	4	5	63	24	503	366	251753	1	17	1	41616
	5	12	30	87	209	6972	27376	5	11	144	190969
	6	6	26	29	201	415	30526	1	6	1	14400
	7	3	104	15	754	190	498301	1	26	1	1225
	8	10	3	46	38	1192	2628	4	3	16	4
	9	4	255	19	1808	288	370137	1	82	1	61465600
	10	4	17	18	106	224	13411	1	4	1	3025
	11	8	29	38	260	568	85270	2	17	16	65025
	12	4	21	19	165	192	26617	1	9	1	5184
	13	4	18	19	144	192	26092	1	6	1	2916
	14	12	18	70	146	2663	25555	1	6	441	16
	15	4	17	19	131	286	21497	1	5	1	1024
	16	3	17	9	100	84	11265	1	3	1	2304
*	17	3	10	9	43	84	3686	1	2	1	1
	18	6	84	45	583	2609	122834	2	22	625	4096576
*	19	11	4	53	35	533	1402	3	1	81	324
	20	5	9	23	56	292	2747	1	1	1	576
	21	5	235	23	2378	292	609581	1	138	1	4064256

Table 28. Design and Source Metric Values for Project: OTHELLO2

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	23	127	162	1066	12808	141762	1	64	61009	107910544
*	2	3	28	9	114	84	2891	1	1	4	4
	3	3	7	14	37	178	2245	1	2	3025	484
	4	34	136	255	1161	51481	430673	13	60	6400	8767521
*	5	8	8	65	89	5833	11222	4	4	256	1296
*	6	27	26	183	164	29212	25930	3	3	3600	2025
*	7	6	3	40	34	3188	2258	4	3	16	16
*	8	3	6	11	27	185	525	1	1	1	4
*	9	4	22	19	165	288	35084	1	6	900	81225
*	10	7	7	61	59	5951	7127	2	2	144	324
*	11	13	15	101	119	16780	22718	5	7	2025	14400
*	12	14	15	109	118	18167	21789	4	6	17424	16900
*	13	20	17	126	121	22329	21374	7	5	10816	14400
	14	19	12	87	82	15245	14213	1	9	400	100
	15	3	6	14	22	178	592	1	1	36	10000
	16	16	41	87	754	5497	157756	3	72	19600	6350400
*	17	4	20	17	106	254	14798	1	1	1	4
	18	3	12	14	74	178	6682	1	1	1	16900
*	19	8	26	52	170	2203	34495	1	4	400	2304
*	20	7	26	46	157	1383	21971	1	2	400	3025
	21	25	22	177	185	28900	44777	7	7	1225	400
*	22	3	49	14	222	178	24298	1	11	7744	171396
*	23	6	22	34	165	741	31085	1	9	144	576
	24	5	78	22	508	391	93780	1	50	4	43264
	25	3	167	9	1290	84	1339405	1	5	1	144
*	26	7	42	33	314	659	97302	1	8	64	900

Table 29. Design and Source Metric Values for Project: PENTE

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	1	45	59	257	345	25990	57204	2	3	22500	1327104
	2	43	17	428	238	76629	61846	14	6	746496	82944
*	3	4	8	20	47	355	2742	1	1	196	169
*	4	23	19	168	165	23975	19975	3	6	57600	57600
*	5	11	7	98	89	10420	8281	6	6	900	900
*	6	11	7	121	112	13550	11107	8	8	900	900
*	7	11	7	121	112	13550	11107	8	8	784	784
*	8	11	7	98	89	11180	8281	6	6	324	324
*	9	11	7	98	89	10420	8281	6	6	576	576
*	10	11	7	98	89	10420	8281	6	6	784	784
*	11	11	7	121	112	12812	11107	8	8	576	576
*	12	11	7	121	112	13550	11107	8	8	324	324
*	13	23	16	168	167	23975	22924	3	8	61009	61009
*	14	10	7	77	69	6169	4690	4	4	900	900
*	15	10	6	93	84	9925	7804	5	5	900	900
*	16	10	7	93	85	8644	6820	5	5	784	784
*	17	10	7	77	69	6169	4690	4	4	324	324
*	18	10	7	77	69	6169	4690	4	4	576	576
*	19	10	7	77	69	6169	4690	4	4	784	784
*	20	10	6	93	84	8644	6740	5	5	576	576
*	21	10	6	93	84	9925	7804	5	5	324	324
*	22	16	10	130	117	17255	12656	6	5	20736	20736
*	23	25	15	195	181	34412	26488	10	9	16384	18225
*	24	25	15	195	307	34412	58609	10	17	11664	140625
*	25	3	31	14	233	178	67075	1	6	676	65025
*	26	37	3	251	81	44938	4626	3	1	65025	262144
	27	4	6	86	77	5177	7167	1	3	147456	576
*	28	9	6	85	97	9177	12251	3	4	576	576
*	29	9	6	105	97	15024	10570	4	4	576	324
*	30	9	6	105	77	13072	7777	4	3	324	784
*	31	9	6	85	77	9909	7167	3	3	784	900
	32	9	6	85	77	9177	7777	3	3	900	324
*	33	9	6	85	97	9909	11405	3	4	324	900
*	34	9	6	105	97	14042	12251	4	4	900	784
*	35	9	5	105	88	15024	8369	4	4	784	1764
*	36	9	5	95	108	8958	13335	4	5	1764	1296
*	37	9	5	115	108	14046	11623	5	5	1296	900
*	38	9	5	115	88	12253	9025	5	4	900	900
*	39	9	5	95	88	9656	8369	4	4	900	1296
*	40	9	5	95	88	8958	9025	4	4	1296	1600
*	41	9	5	95	108	9656	12474	4	5	1600	1764
*	42	9	5	115	108	13145	13335	5	5	1764	1600
*	43	9	4	115	31	14046	1334	5	1	1600	26244
	44	24	4	189	29	32959	2116	16	2	11664	1521

Design and Source Metric Values for Project: PENTE continued ...

Metric Values

	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	11	8	81	91	7041	9115	3	3	2601	4096
*	46	13	11	156	210	21446	48486	9	13	4	400
*	47	8	6	46	40	3536	3112	2	2	4	9
*	48	16	10	130	117	17255	12656	6	5	20449	20449
*	49	25	15	195	181	34412	26488	10	9	18225	19600
*	50	25	15	195	181	34412	26488	10	9	14161	16384
*	51	13	8	160	151	22588	19098	5	5	42849	48400
*	52	23	16	168	166	23975	22787	3	8	63504	63504
*	53	10	6	77	68	6169	4622	4	4	900	900
*	54	10	6	93	84	9925	7804	5	5	900	900
*	55	10	6	93	84	8644	6740	5	5	784	784
*	56	10	7	77	69	6169	4690	4	4	324	324
*	57	10	7	77	69	6169	4690	4	4	576	576
*	58	10	6	77	68	6169	4622	4	4	784	784
*	59	10	6	93	84	8644	6740	5	5	576	576
*	60	10	6	93	84	9925	7804	5	5	324	324
*	61	24	4	189	31	32959	1334	16	1	12100	32400
*	62	24	10	221	203	40064	33834	18	12	12100	23716
*	63	24	10	221	203	40064	33834	18	12	11664	18496
*	64	12	18	81	132	5642	21282	7	8	2304	32400
	65	17	23	86	132	9164	26634	6	6	64	1
*	66	12	18	90	174	8622	66221	11	12	4	9
	67	17	23	86	132	9164	26634	6	6	64	1
	68	17	23	86	132	9164	26634	6	6	64	1
*	69	36	12	160	69	14721	6301	2	1	25	2401
	70	3	39	9	229	84	61202	1	8	1	576
*	71	26	27	113	229	9930	84239	4	2	36	144
*	72	27	27	155	245	42482	90125	2	2	4	144
*	73	27	27	171	229	46868	84239	2	2	4	144
	74	27	12	159	69	43579	6301	2	1	4	2401
*	75	3	10	9	55	84	6622	1	4	1	1
	76	5	28	26	180	535	17831	1	7	4	576
*	77	9	62	49	423	4086	165518	4	9	9	576
	78	12	15	72	126	4335	36855	4	8	400	16
*	79	6	11	34	76	1049	8711	2	5	900	1600
	80	11	5	66	39	2304	2154	1	3	8100	576
	81	5	34	20	403	249	66505	1	12	196	8294400
*	82	5	21	20	258	249	83895	1	9	64	4225
*	83	5	21	20	286	249	99642	1	10	196	4900
*	84	5	21	20	286	249	90544	1	10	400	5184
*	85	5	21	20	258	249	83895	1	9	324	5929
*	86	5	21	20	258	249	83895	1	9	64	5184
*	87	5	21	20	258	249	83895	1	9	324	3136
*	88	5	21	20	286	249	90544	1	10	400	5184

Design and Source Metric Values for Project: PENTE continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	89	14	21	207	286	40082	99642	8	10	8100	4900
*	90	12	67	72	502	3232	265571	1	21	2500	139876
	91	4	19	19	309	237	108595	1	15	1	11664
	92	4	4	19	35	295	1941	1	2	36	4356
*	93	4	4	24	35	602	1941	1	2	100	5184
*	94	4	4	24	35	602	1941	1	2	144	4900
*	95	4	4	24	35	602	1941	1	2	100	3600
*	96	8	9	47	54	2725	3466	1	1	25	64
	97	9	41	41	328	1383	127351	1	11	256	330625
	98	3	6	18	32	332	1494	1	1	2401	441
*	99	14	11	86	77	10319	7780	5	4	44100	73984
*	100	15	12	101	92	14461	11306	6	5	54756	51984
*	101	15	12	101	92	12840	9911	6	5	44100	83521
*	102	14	11	86	77	10972	8329	5	4	54756	57600
*	103	14	11	86	77	10319	7780	5	4	54756	57600
*	104	14	11	86	77	10972	8329	5	4	33856	72900
*	105	15	12	101	92	13647	10605	6	5	44100	83521
*	106	15	12	101	92	14461	11306	6	5	57600	54756
	107	3	12	9	69	84	6301	1	1	1	2401

Table 30. Design and Source Metric Values for Project: POKER

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	280	249	1748	1300	143947	119036	45	9	47334400	1679616
	2	8	31	46	194	2198	31191	1	8	4	1764
*	3	12	11	101	105	16219	12921	4	3	36	49
	4	58	130	504	1417	201942	628944	24	58	4900	14212900
	5	4	4	31	24	955	733	1	1	9	1
	6	9	6	103	101	9585	12839	3	2	4900	1600
*	7	28	37	253	405	56754	127674	12	16	3969	17424
	8	20	15	134	124	24030	21345	9	7	36	16
*	9	25	20	188	191	47341	46504	12	11	36	36
*	10	7	4	50	44	2928	2319	4	3	9	36
*	11	11	50	69	372	5895	200275	2	20	144	1024
*	12	25	62	190	513	36253	165560	3	23	23716	608400
*	13	12	15	64	123	3277	17297	2	8	324	729
*	14	10	8	60	51	3465	3110	2	3	784	2304
*	15	11	19	59	136	3074	15108	2	7	576	1024
*	16	6	7	41	45	2155	2064	3	2	324	400
*	17	12	14	64	95	3277	9430	2	5	576	729
*	18	12	14	69	121	4151	16153	2	8	900	2401
*	19	9	8	53	54	2615	3800	3	3	1089	1296
	20	10	8	60	51	3465	3110	2	3	9	2304
	21	12	14	69	121	4151	16153	2	8	16	2401
*	22	13	24	70	155	4008	25803	2	8	900	1225
*	23	13	19	70	157	3778	33956	2	9	784	1024
*	24	37	26	187	328	16451	45966	1	10	16	36
*	25	39	25	214	425	21624	88958	3	11	16	36
*	26	16	33	71	180	6825	26827	1	13	9	16
	27	28	11	133	62	20446	6505	1	5	9	1
*	28	9	25	62	198	5505	22976	4	11	576	3969
	29	14	15	58	74	4484	9221	1	4	25	9
*	30	12	11	56	59	4723	5627	1	5	1	100
*	31	6	8	34	47	1167	2632	1	2	1	1
	32	3	2	20	15	502	269	1	1	72900	36
	33	15	22	98	145	11428	45438	3	9	100	16
*	34	3	2	22	17	603	350	1	1	16	16
*	35	11	35	78	226	8657	20855	1	11	36	2916
*	36	22	18	144	142	16657	30329	5	6	4356	2500
	37	8	8	43	40	1434	2002	1	1	144	64
	38	3	2	19	14	394	251	1	1	16	4
*	39	21	21	93	112	11059	15681	1	4	9801	8100
*	40	12	13	55	79	3071	3609	1	1	225	1600
*	41	32	50	201	306	61895	54788	10	10	400	10816
	42	36	33	160	180	22800	26827	1	13	625	16
	43	3	2	20	15	502	269	1	1	1764	4
	44	32	111	202	888	21389	198831	11	53	1024	876096

Design and Source Metric Values for Project: POKER continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	31	25	150	159	22511	23482	1	11	576	1225
	46	12	13	57	69	3062	3864	1	1	3025	900
*	47	49	85	234	468	49471	166113	6	14	576	9216
	48	4	53	19	365	351	58821	1	23	441	72900
	49	19	83	119	615	13283	147183	10	37	576	419904

Table 31. Design and Source Metric Values for Project: SCHEDULE

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	111	478	587	2651	55380	339501	6	83	127449	243984400
	2	20	17	116	133	8422	23730	7	7	44100	11025
*	3	10	7	59	54	6412	6740	4	4	144	256
*	4	17	14	105	103	10191	12309	3	5	625	1024
*	5	21	28	136	192	18780	40238	7	5	1600	4900
	6	20	15	117	126	11040	20382	5	7	2500	576
*	7	19	32	117	222	12394	44487	5	6	2304	12100
*	8	22	15	144	161	16938	31348	7	8	900	900
*	9	7	17	43	116	2812	13056	2	2	16	400
*	10	14	38	68	348	4676	173282	8	10	36	225
*	11	31	57	177	378	24257	79590	7	10	1764	33124
	12	19	25	106	236	13607	49080	5	7	144	64
	13	8	5	54	47	3835	3447	2	4	1296	576
*	14	4	8	25	48	750	2540	1	2	100	484
*	15	3	3	23	20	539	259	1	1	4	9
*	16	3	2	9	4	83	6	1	1	1	1
*	17	8	8	77	78	8205	9074	2	2	225	3025
*	18	3	2	17	8	304	56	1	1	36	169
*	19	14	21	69	96	5010	9470	2	6	2304	2401
*	20	12	16	68	90	4053	9260	3	3	36	256
	21	7	62	47	357	3870	83827	3	18	64	50625
	22	6	62	33	387	1518	49741	2	16	16	153664
*	23	25	51	122	324	15514	78443	5	17	8100	25600
*	24	7	13	47	68	3870	8447	3	3	784	3136
*	25	6	29	33	201	1297	25117	2	7	2401	57600
	26	19	25	108	240	13904	46340	5	7	784	324
*	27	16	17	70	71	5200	5673	2	5	144	441
	28	5	4	37	33	1060	1297	1	2	4	1
*	29	4	8	10	32	92	114	1	1	4	100
*	30	3	2	9	4	83	6	1	1	1	1
	31	3	3	9	6	83	12	1	1	16	4
*	32	5	21	26	131	936	33863	2	5	144	441
	33	6	5	27	22	351	293	1	1	1	4225
	34	5	4	23	18	291	225	1	1	1	400
	35	8	48	58	297	4688	51071	2	18	4	5929
	36	3	6	15	28	148	338	1	2	1	144
	37	11	165	69	1112	7662	211250	3	36	1	439569
*	38	5	5	23	22	291	343	1	1	1	1
*	39	3	4	18	24	243	76	1	1	1	4
	40	74	80	832	578	445001	238802	1	1	16	1
	41	5	19	19	181	287	21524	1	5	16	1
*	42	19	17	76	65	9207	5131	5	5	9	36
	43	7	4	33	21	963	630	1	2	4	1
	44	30	59	189	408	31089	72767	14	18	36	69696

Design and Source Metric Values for Project: SCHEDULE continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 45	3	2	9	4	83	6	1	1	1	1
* 46	6	14	30	86	1471	9803	2	2	9	64
47	13	77	79	518	8764	157780	4	18	1	23104
* 48	3	2	18	4	243	6	1	1	1	9
49	5	45	26	276	757	30123	1	11	784	226576
* 50	5	14	16	34	216	1249	2	1	1	100
* 51	5	6	27	30	1130	879	3	1	1	1

Table 32. Design and Source Metric Values for Project: SCORE4

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	10	52	76	248	3679	9904	1	3	9	173056
	2	3	133	9	1086	84	233428	1	53	1	50176
*	3	3	76	9	455	84	143530	1	25	1	1
	4	3	51	9	217	84	65562	1	9	1	441
*	5	16	56	56	279	2996	111096	2	8	9	196
*	6	3	38	13	162	130	19112	1	11	1	16
*	7	4	25	20	142	301	23412	2	7	16	25
	8	6	25	29	142	712	25353	1	7	64	25
*	9	17	9	71	64	5717	4307	3	4	1	64
	10	11	30	69	165	5331	24146	3	7	324	16
*	11	7	23	30	166	615	43946	2	13	225	900
*	12	4	12	10	74	93	5584	1	2	81	2025
	13	3	366	9	2302	84	562241	1	104	1	177156096
	14	3	63	9	521	84	117910	1	25	1	14161
*	15	3	37	9	170	84	8650	1	1	1	1
	16	4	22	10	186	93	22485	1	11	16	3136
	17	3	422	9	2927	84	716039	1	175	1	50979600
	18	3	61	9	1728	84	702745	1	103	1	174724
*	19	7	9	30	59	615	3487	2	2	100	1600
	20	4	366	10	2302	93	562241	1	104	81	177156096
*	21	3	32	9	238	84	86990	1	1	1	1
	22	3	63	9	521	84	117910	1	25	1	14161
	23	3	26	9	224	84	53626	1	8	1	256
*	24	3	29	9	185	84	31913	1	8	9	256
	25	3	142	9	861	84	250402	1	51	1	50176
	26	3	64	9	373	84	43707	1	18	4	11025
	27	3	133	9	1086	84	233428	1	53	9	50176

Table 33. Design and Source Metric Values for Project: STARTREK

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	79	139	420	1006	33788	127995	8	38	44100	14992384
*	2	9	91	21	360	195	8909	2	1	36	256
	3	3	18	9	100	84	2362	1	6	1	1296
	4	5	7	22	42	331	2753	1	3	4	1
	5	4	91	13	645	219	145123	2	26	1	2304
*	6	11	49	17	295	158	169033	1	23	81	144
*	7	3	34	9	124	84	6289	1	1	1	4
*	8	3	48	9	190	84	14455	1	1	1	4
*	9	3	46	9	180	84	9129	1	1	1	4
*	10	4	19	10	74	93	1554	1	1	1	4
*	11	3	32	9	118	84	2478	1	2	1	64
*	12	3	9	9	38	84	798	1	1	1	4
*	13	3	14	9	52	84	1092	1	1	1	4
*	14	3	9	9	38	84	798	1	1	1	4
	15	3	24	9	50	84	175	1	2	1	484
*	16	11	122	63	1169	2023	778011	1	37	144	196
*	17	31	39	155	206	8831	24280	3	5	225	1225
	18	49	46	221	309	13297	53851	5	16	529984	56644
	19	4	18	25	155	586	18794	1	1	1	144
	20	4	20	23	173	348	14067	1	8	2401	324
	21	5	5	29	32	638	2125	1	1	100	36
*	22	11	56	71	321	4261	19007	1	11	1764	112896
*	23	12	28	58	173	3428	17554	2	6	400	3136
*	24	4	10	20	70	310	5460	1	1	16	324
*	25	10	19	73	150	2840	6160	1	1	1	36
*	26	3	14	13	78	176	7125	1	2	81	8100
	27	9	23	48	109	1725	17474	1	3	1024	25
*	28	3	24	11	110	185	21509	1	3	1	25
	29	3	14	11	92	185	17190	1	2	4	5184
	30	9	4	48	18	2912	378	3	1	2304	1
*	31	3	4	12	14	202	147	1	1	1	1
*	32	3	4	12	14	202	147	1	1	1	1
	33	4	20	13	197	117	18271	1	9	1	67081
*	34	4	20	21	199	263	17439	1	9	1521	32400
*	35	6	16	45	266	2676	79101	3	5	784	2304
	36	3	17	17	184	264	35561	1	10	3600	225
	37	13	9	74	129	4530	22450	4	2	5184	64
*	38	22	14	114	176	10903	44585	2	4	36	36
	39	18	39	101	403	3527	210958	4	13	5184	1936
	40	4	26	34	172	1274	38124	1	6	9	6084
*	41	7	6	37	28	2313	902	1	1	16	16
	42	21	24	127	111	16328	17795	7	3	2025	25
	43	3	30	13	221	176	22569	1	9	64	27225
	44	3	12	14	71	355	9622	1	2	1	5184

Design and Source Metric Values for Project: STARTREK continued ...

Metric Values

	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	45	25	6	174	36	30663	2491	7	1	1600	1
*	46	9	82	53	546	3929	159142	3	21	1296	3600
*	47	3	4	12	24	202	645	1	1	1	36
*	48	4	20	24	128	747	17505	1	2	64	3969
*	49	3	28	19	166	681	18681	1	5	36	144
	50	5	10	28	53	738	3516	1	1	4	576
	51	5	181	23	1310	825	351822	1	27	1	40144896
*	52	25	42	177	333	30621	90693	8	12	44100	189225
*	53	9	14	57	110	3305	8099	1	4	6400	84100
*	54	9	10	71	76	4075	2464	3	2	7056	32400
*	55	4	8	21	76	316	4944	1	8	900	28224
	56	3	10	17	238	264	77236	1	16	2304	828100
	57	18	12	162	68	16438	3756	11	2	73984	7056
*	58	9	12	56	68	3774	3274	1	2	1764	3136
*	59	9	16	56	96	3774	7118	1	2	1764	6400
*	60	11	16	70	96	5792	7320	1	2	1296	9216
*	61	11	12	70	68	5792	3756	1	2	1600	1024
*	62	9	12	56	68	3774	3274	1	2	1600	9216
*	63	8	16	51	96	3409	5874	1	2	900	1024
*	64	11	16	70	96	6281	7118	1	2	144	7056
*	65	11	18	70	104	6281	11317	1	2	484	4900
	66	11	6	68	32	3825	1344	3	1	1225	1
	67	3	20	9	123	84	12456	1	4	1	2304
	68	13	23	76	109	4186	21313	4	3	1089	36
*	69	3	8	14	34	355	126	1	1	1	16
*	70	3	15	12	181	202	43032	1	11	1	4
	71	19	6	168	45	32952	2261	11	1	4	1600
*	72	3	24	11	110	185	21509	1	3	4	49
*	73	4	9	16	54	216	3392	1	8	16	16
*	74	10	12	67	95	6770	13272	2	3	576	22500
*	75	4	7	16	32	270	3501	1	1	1	1
	76	6	8	40	78	958	7618	1	5	400	36
*	77	26	18	155	156	15371	14322	1	8	1	4
*	78	3	4	13	22	176	761	1	1	17424	10000
*	79	3	4	15	28	233	1140	1	1	1936	28224
*	80	4	5	19	30	288	967	1	1	50176	254016
	81	3	5	13	23	176	796	1	1	784	256
*	82	3	4	15	28	233	1140	1	1	1296	25600
*	83	3	4	15	26	233	813	1	1	32400	65536
*	84	16	117	94	579	8748	88289	2	15	3025	202500
*	85	8	8	37	105	951	11055	1	7	144	576
*	86	4	6	19	38	288	1808	1	1	9	625
*	87	5	3	29	23	1987	1051	3	2	4	4
*	88	4	4	19	37	237	1883	1	2	1	1

Design and Source Metric Values for Project: STARTREK continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 89	3	18	9	78	84	11423	1	3	16	25
* 90	3	21	9	80	84	1316	1	1	1	1
* 91	3	12	22	76	543	5919	1	3	2500	25600

Table 34. Design and Source Metric Values for Project: STRATEGO

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	54	126	259	902	11985	124958	7	35	112896	26615282
	2	3	111	9	911	84	237247	1	46	16	12996
*	3	4	14	19	236	241	63402	1	19	1	16
	4	8	6	39	27	835	927	1	1	1	256
	5	8	6	39	27	835	927	1	1	1	225
*	6	3	29	14	189	178	39451	1	8	36	441
*	7	7	6	38	36	1223	976	1	1	256	625
	8	3	27	9	104	84	2309	1	1	1	144
*	9	3	25	9	130	84	11980	1	1	1	16
	10	5	41	24	301	426	43791	1	16	36	28224
	11	3	245	14	1884	178	661657	1	58	1	230400
*	12	4	8	19	43	295	1879	1	1	1	4
*	13	5	44	24	265	344	44966	1	3	1	4
*	14	3	26	9	134	84	12349	1	1	1	16
	15	4	111	19	922	286	158160	1	37	36	611524
	16	10	308	41	2912	878	2383247	1	183	4	24285184
	17	16	82	58	425	4089	72956	4	24	16	127449
	18	11	9	46	74	1065	7107	1	2	4	784
	19	8	52	39	532	835	235200	1	31	1	2916
*	20	7	24	35	204	638	39687	1	8	256	1600
	21	3	111	14	922	178	158160	1	37	1	611524
*	22	16	152	66	976	1656	231435	1	35	44100	275625
	23	5	111	16	976	203	274686	1	35	4	525625
*	24	6	12	17	81	216	5905	1	4	400	441
	25	8	47	39	392	835	121859	1	23	1	400
	26	8	47	39	386	835	120839	1	22	1	900
	27	3	222	14	1600	178	513299	1	54	1	213444
	28	36	215	168	1793	2136	792268	12	85	49280400	207025
*	29	5	12	22	68	365	5679	1	7	144	144
	30	8	153	39	1159	835	280043	1	70	1	1615441
*	31	6	7	35	33	838	1289	1	1	81	144
*	32	6	11	29	64	460	3450	1	2	1	9
	33	3	16	9	88	84	3619	1	13	1	900
	34	4	3	10	56	93	4053	1	7	1	900
	35	3	230	9	1819	84	321050	1	101	1	46580624
*	36	3	26	9	128	47	9020	1	1	16	16
*	37	3	20	9	98	47	6375	1	1	1	16

Table 35. Design and Source Metric Values for Project: SUBQUEST

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	8	227	42	1557	1649	449468	1	17	4225	9734400
*	2	34	26	184	147	12759	19954	2	5	16	784
*	3	12	25	46	224	969	38562	1	7	36	81
	4	10	87	64	1082	4646	216774	4	42	1	9144576
	5	12	11	61	48	4195	2622	2	1	1	4900
	6	10	36	66	225	4793	43029	2	10	4	1
*	7	25	20	184	131	38968	12846	10	2	36	81
*	8	12	9	81	65	6856	4888	2	3	4	36
*	9	13	21	91	192	10617	42060	4	12	4	16
*	10	3	9	25	69	652	5086	1	4	4	16
*	11	7	17	32	132	639	20626	1	4	1	16
	12	20	6	127	45	10685	2698	8	1	36	4
*	13	9	15	41	76	2682	11908	2	2	64	400
	14	3	24	9	189	84	33284	1	8	1	3136
*	15	12	9	54	65	2312	3541	3	2	256	225
	16	3	24	9	350	84	105887	1	1	1	1296
	17	3	51	9	717	84	208811	1	49	1	129600
	18	13	21	73	260	3376	69415	4	4	36	9801
	19	3	24	14	350	233	105887	1	1	4	1296
	20	3	15	14	131	233	13910	1	5	1	1296
*	21	3	14	11	115	185	10941	1	3	1	36
*	22	3	12	9	128	84	4778	1	1	1	1
*	23	19	33	141	374	23040	114353	6	14	144	2304
	24	3	24	9	350	84	105887	1	1	1	1296
*	25	24	36	184	469	32804	163817	11	16	36	784
	26	3	21	9	260	84	69415	1	4	1	9801
*	27	3	19	9	224	84	51604	1	2	1	100
*	28	5	7	22	46	365	2692	1	3	144	400
*	29	14	73	55	397	1479	80616	1	5	16	729
*	30	4	4	13	8	219	19	1	1	1	4
*	31	7	50	35	268	941	65465	1	1	9	400
	32	13	29	56	132	4053	11528	4	4	25	3600
	33	12	148	59	657	5284	32025	2	20	16	82944
*	34	3	14	11	115	185	10941	1	3	1	36
*	35	3	13	9	52	84	4554	1	3	1	4
*	36	3	14	11	115	185	10941	1	3	9	36
	37	21	115	90	462	7224	15087	4	6	36	4356
*	38	3	14	11	115	185	10941	1	3	1	36
	39	18	99	86	789	5712	259465	1	14	9	4900

Table 36. Design and Source Metric Values for Project: TALLY

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	43	652	261	4833	22148	975960	3	155	75625	1569585920
	2	3	6	9	40	84	2923	1	2	1	225
	3	3	25	9	190	47	60301	1	7	1	144
	4	35	10	173	20	8813	69	3	1	70756	64
	5	3	107	9	641	84	126883	1	34	1	495616
	6	4	71	17	469	212	109790	1	6	49	9025
*	7	3	55	16	382	199	88581	1	4	144	2704
*	8	17	54	68	358	3086	61338	1	11	576	33124
*	9	17	54	68	356	3086	60964	1	11	2304	46656
*	10	3	14	9	128	84	19533	1	3	9	16
*	11	8	68	41	355	1225	18761	1	9	256	20736
*	12	7	21	38	97	965	5121	1	3	4	400
*	13	5	6	26	38	721	1547	1	1	1	9
	14	17	126	77	759	4957	49733	1	17	576	379456
	15	11	47	63	305	4425	37093	1	8	64	14161
*	16	3	70	16	367	199	21417	1	9	225	22500
	17	30	106	160	794	28789	191079	10	31	441	278784
*	18	12	18	72	154	4872	26270	1	4	196	7744
*	19	4	6	19	52	295	2514	1	1	1	9
	20	5	14	19	120	409	8721	3	3	16	18496
*	21	20	15	98	267	6078	80531	11	5	16	441
	22	7	234	41	1793	1710	420206	3	60	4	15968016
*	23	6	5	51	55	2810	3965	3	2	8100	15876
*	24	24	52	110	144	7756	26486	6	5	900	900
*	25	10	12	50	70	2124	7493	1	1	25	25
*	26	24	64	110	143	7756	26302	6	5	1296	1296
*	27	9	12	46	70	2038	7493	1	1	16	25
	28	31	63	202	414	47082	75341	15	19	1	24336
	29	9	20	57	200	3384	25922	2	10	225	705600
	30	5	3	31	45	905	3792	3	2	4	14400
	31	10	53	50	293	1752	69106	1	14	1	4624
	32	10	52	50	289	1752	65108	1	14	4	6400
*	33	26	29	69	249	2949	62499	2	14	121	12100
*	34	8	8	43	43	2838	2210	2	2	1	64
	35	5	6	29	119	690	21502	3	3	1	256
	36	5	18	29	273	690	90805	3	8	1	5184
	37	6	20	42	316	1294	112425	5	10	1	7056
	38	6	8	42	163	1294	52262	5	5	1	1296
	39	3	3	25	49	1121	3580	4	4	49	19600
	40	3	3	15	22	571	660	1	1	25	342225
	41	5	6	33	38	1253	1799	1	1	4	729
	42	10	53	58	456	2930	69065	1	10	25	61009
*	43	12	14	73	92	7325	16873	3	4	64	225
	44	12	9	47	43	3303	5738	2	2	16	4

Design and Source Metric Values for Project: TALLY continued ...

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 45	15	13	97	106	12232	14961	5	3	2304	4900
* 46	17	29	116	460	24817	496672	10	10	36	36
* 47	14	16	99	178	12630	39603	7	10	81	144
* 48	18	9	82	259	4445	153691	15	11	144	144
* 49	7	6	43	122	1898	31524	6	5	16	16
* 50	5	4	31	97	930	19717	5	4	16	16
* 51	15	28	87	224	19090	45862	8	11	4	144
52	7	139	28	1035	533	285983	1	31	1936	298116
53	4	24	10	161	93	25007	1	5	1	1225
* 54	15	42	89	309	8296	98125	1	7	25	441
* 55	11	30	73	207	5335	44693	1	9	25	400
* 56	4	4	10	14	93	233	1	1	4	36
* 57	19	19	63	65	5446	15218	2	2	49	49
* 58	21	19	77	82	2760	2723	1	1	16	36
59	16	59	124	466	26688	84539	6	13	49	43264
60	50	52	313	461	70532	237818	20	18	70756	27225
61	4	34	19	392	341	147403	1	23	25	78400
* 62	16	24	87	164	7905	42718	3	8	441	1225
* 63	21	26	130	254	15237	73274	13	10	576	1225
* 64	17	16	104	107	17904	20250	7	6	1	1
* 65	9	10	54	60	2601	2315	1	1	900	3136
66	42	67	280	496	78224	99908	10	21	1225	254016
* 67	3	36	14	327	533	78445	1	16	100	1764
68	4	86	10	968	93	519673	1	34	144	92416

Table 37. Design and Source Metric Values for Project: TWIXT

Metric Values										
PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
* 1	73	343	497	2052	65896	534731	3	66	853776	29452330
* 2	4	28	19	211	341	37464	1	11	400	4096
* 3	4	27	19	216	341	39001	1	12	36	1600
* 4	17	34	127	281	18587	59331	8	13	7056	44100
5	55	31	473	292	169428	86844	22	16	7744	256
* 6	5	2	31	24	1293	801	3	2	16	16
7	8	4	50	44	5351	6181	4	3	144	36
8	21	18	126	194	13697	40020	6	8	144	36
* 9	11	14	82	131	5624	15617	1	1	36	36
* 10	62	43	483	320	171777	69583	20	16	58564	38025
* 11	21	15	180	217	42408	51943	7	16	144	144
* 12	63	33	646	815	545814	730171	31	64	100	100
* 13	21	15	180	217	42408	51943	7	16	144	144
* 14	25	17	276	493	95088	216172	12	35	784	9801
15	9	2	69	60	9213	7168	7	4	36	16
16	9	2	69	60	7795	7168	7	4	36	16
17	8	4	50	46	5351	6462	4	3	729	1
* 18	16	56	141	652	21019	167470	5	29	28224	81225
* 19	18	27	119	244	17717	68877	4	8	625	900
* 20	25	17	276	493	95088	216172	12	35	256	9801
21	9	2	69	60	9213	7168	7	4	36	16
22	9	2	69	60	7795	7168	7	4	36	16
23	8	4	50	46	5351	6462	4	3	144	36
* 24	5	3	31	27	1293	885	3	2	25	36
* 25	18	17	137	162	23373	28256	11	10	225	324
* 26	21	38	127	305	12841	59394	3	13	784	5184
* 27	25	17	276	493	95088	216172	12	35	400	9801
28	9	2	69	60	9213	7168	7	4	36	16
29	9	2	69	60	7795	7168	7	4	36	16
* 30	13	22	82	153	7311	21238	5	4	64	441
* 31	63	44	714	747	694801	652205	27	21	900	900
* 32	9	2	69	46	7795	4473	7	2	36	36
33	16	19	92	138	4593	11387	1	1	36	16
* 34	15	37	83	208	3055	31899	1	4	8281	123904
* 35	4	20	22	145	380	31462	1	7	9	9
* 36	3	12	14	70	178	3809	1	4	1	1
37	4	17	22	168	380	37158	1	9	36	9
* 38	11	12	67	69	2771	2819	1	1	1	1
* 39	15	40	74	260	6107	64459	2	11	576	8281
* 40	6	6	34	36	1020	1126	1	1	1	1
* 41	20	19	129	183	20716	41358	8	12	64	1764
* 42	10	11	65	97	6634	12214	4	7	81	900
43	35	94	159	551	16333	114731	3	33	16900	8785296
44	5	6	24	75	426	9175	1	6	81	12544

Design and Source Metric Values for Project: TWIXT continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	3	8	14	44	233	1927	1	1	1	36
*	46	3	8	14	44	233	1927	1	1	1	81
	47	3	33	9	257	84	83124	1	17	1	17424
*	48	5	6	30	31	963	732	1	1	144	784
*	49	10	8	77	86	8714	15666	5	4	2304	14400
*	50	8	5	34	29	2422	2508	1	4	4	4
*	51	6	4	41	36	2894	2537	3	2	324	196
*	52	4	4	19	26	345	1030	1	1	400	324
*	53	10	8	77	83	9025	14818	5	4	2025	13689
*	54	8	6	34	30	2422	2595	1	4	4	4
	55	4	5	19	42	284	2452	1	1	64	7225
	56	14	65	181	731	30825	248678	3	24	4096	1073296
*	57	12	11	83	260	7487	99435	4	11	36	324
*	58	17	32	159	401	32071	128563	2	19	1	36
*	59	6	5	35	30	890	771	1	1	225	576
*	60	7	11	119	183	20367	43184	3	9	4	16
*	61	17	77	32	1769	482	3556891	1	10	4	16

Table 38. Design and Source Metric Values for Project: VEGAS

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	10	113	49	996	2048	207765	1	42	18496	30041362
*	2	15	52	42	296	2029	96076	2	13	16	81
*	3	3	88	9	324	84	43590	1	1	1	1
*	4	3	32	9	130	84	5830	1	1	1	1
*	5	28	36	108	162	1819	9583	1	1	1	4
*	6	3	50	9	234	84	36312	1	1	1	25
	7	17	56	83	394	6058	128846	2	17	25	14161
	8	48	159	323	1058	51319	354193	7	20	9216	1218816
*	9	10	27	57	170	2288	18698	1	2	1296	18496
	10	8	73	51	495	2985	211371	1	5	4	1249924
*	11	6	25	34	130	1560	17844	1	1	4	324
*	12	7	9	34	56	1114	4053	1	3	576	2025
	13	8	73	51	495	2985	211371	1	5	1	1249924
*	14	6	13	41	116	2195	21138	1	3	4	144
	15	8	73	51	495	2985	211371	1	5	1	1249924
*	16	8	19	45	115	1800	6234	1	1	324	3136
*	17	7	39	64	360	2419	85041	1	15	2500	78400
	18	10	68	58	604	3610	165915	3	27	2304	1102500
	19	6	25	35	186	1374	25628	1	4	36	6400
	20	8	73	51	495	2985	211371	1	5	1	1249924
	21	8	73	51	495	2985	211371	1	5	4	1249924
	22	6	13	41	116	2195	21138	1	3	4	1225
	23	8	73	51	495	2985	211371	1	5	1	1249924
	24	5	156	24	1089	426	302932	1	32	1	10890000
	25	8	73	51	495	2985	211371	1	5	1	1249924
	26	4	79	22	350	442	62340	1	5	900	9
*	27	4	11	22	65	628	4507	1	4	36	25
*	28	5	45	32	345	934	99354	1	14	256	5625
*	29	5	20	32	155	1561	25905	1	13	4	36
	30	17	107	90	795	10539	234743	2	39	3969	3610000
*	31	6	25	35	186	1374	25628	1	4	144	8100
	32	8	73	51	495	2985	211371	1	5	1	1249924
	33	8	73	51	495	2985	211371	1	5	16	1249924
*	34	6	13	41	116	2195	21138	1	3	100	1600
	35	8	73	51	495	2985	211371	1	5	1	1249924
	36	5	156	24	1089	426	302932	1	32	64	10890000
	37	8	73	51	495	2985	211371	1	5	1	1249924
*	38	6	29	35	186	1384	40777	2	6	2500	4096
*	39	6	85	35	527	1545	165806	1	22	36	3249
*	40	43	88	185	733	15189	92648	5	18	14400	132496
*	41	3	44	9	148	84	492	1	1	1	16
*	42	3	36	9	155	84	12253	1	2	1	49
*	43	4	15	19	72	341	2732	1	1	16	49
	44	17	154	97	902	10030	130957	5	35	900	1806336

Design and Source Metric Values for Project: VEGAS continued ...

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	45	5	191	25	1596	513	384778	2	73	4	4941729
	46	6	193	31	1515	903	584638	3	86	9	369664
*	47	3	44	9	154	84	552	1	1	1	100
*	48	31	99	181	876	28420	421809	5	16	4900	140625
*	49	13	28	118	283	17415	102312	4	12	49	3136
*	50	14	14	45	114	1449	13685	1	8	144	1764
*	51	21	38	93	339	8586	52683	3	15	256	5184
*	52	14	22	45	248	1459	49530	1	6	100	3136
	53	23	121	143	1064	21070	174203	4	49	144	3748096
*	54	3	10	16	62	242	2455	1	1	36	3136
	55	4	33	10	152	93	10839	1	1	1	121
*	56	3	69	14	264	178	1040	1	2	9	484
	57	9	136	51	839	3904	217456	3	23	144	82944
*	58	3	46	9	212	84	52180	1	1	1	25
*	59	24	143	133	1104	14569	958734	2	3	3136	10816
*	60	8	5	43	35	1374	1400	1	1	784	784
*	61	5	26	29	263	696	41203	1	1	1	16
*	62	3	8	9	324	84	261954	1	1	1	1
	63	4	35	18	351	269	171666	1	14	16	13689
	64	13	100	78	915	6201	187999	2	42	9	6205081
	65	8	195	34	1953	627	636913	1	87	4	35760400
*	66	3	44	9	256	84	29286	1	1	1	1
*	67	17	17	71	138	4772	20188	1	6	36	225
*	68	16	25	81	144	6831	16480	3	2	1	36
*	69	28	27	108	124	1819	5561	1	1	1	1
	70	6	82	33	1008	688	292537	1	53	36	70756
*	71	20	40	96	373	6509	152886	1	9	16	1296
*	72	3	6	9	317	84	252966	1	1	1	1
	73	25	195	150	1314	19463	1379574	5	19	1764	3952144
	74	6	85	35	325	1374	21493	1	14	1	156816
	75	3	56	9	249	47	11966	1	3	1	6084
	76	3	69	17	549	529	44194	1	5	1	16384
	77	16	94	101	546	17577	75130	3	11	36	219024
*	78	16	19	69	112	6128	12621	4	7	1	49
	79	3	72	9	389	84	68135	1	6	1	2304
*	80	3	84	9	360	84	57309	1	1	1	36

Table 39. Design and Source Metric Values for Project: XWORD

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	92	313	514	1673	65053	336809	6	37	360000	109998144
*	2	33	49	202	295	41612	78709	5	8	144	6561
*	3	3	18	9	75	83	6072	1	2	1	49
*	4	71	89	519	718	137910	263200	22	26	23716	56644
*	5	13	12	79	71	8217	6429	2	2	729	729
*	6	66	48	714	716	508764	505435	27	32	12100	17424
*	7	5	17	24	152	426	35270	1	7	16	16
*	8	15	19	84	129	12848	36802	3	3	81	81
*	9	7	8	32	73	731	10966	1	5	400	400
*	10	10	8	65	59	4442	3413	1	3	81	81
*	11	69	67	431	476	138799	192634	17	14	20736	23716
*	12	6	12	34	79	1701	11792	1	3	144	144
	13	19	19	118	123	13623	15491	2	4	1	256
*	14	18	14	115	112	10400	10357	3	3	9801	13689
*	15	9	21	47	172	2154	16499	3	2	81	1521
*	16	8	7	47	44	2421	2139	1	2	4	4
*	17	10	8	64	59	4307	3976	1	3	25	25
*	18	14	8	98	73	21907	11404	7	5	196	196
*	19	13	13	76	79	6799	7129	3	2	2500	3600
*	20	7	21	34	143	893	30906	1	3	4	4
*	21	7	8	32	73	731	10966	1	5	324	324
*	22	28	26	181	188	33046	32843	9	9	2025	6561
*	23	23	24	186	180	38013	33265	3	4	2304	4096
*	24	6	18	30	142	1124	33886	1	4	1	16
*	25	7	8	32	73	731	11404	1	5	400	400
*	26	13	18	65	102	3841	9013	6	7	64	4900
*	27	5	4	24	18	426	217	1	1	30976	640000
*	28	3	6	14	22	232	614	1	1	1	16
*	29	6	20	32	164	853	23771	1	7	4	36
*	30	3	17	9	67	83	5556	1	2	1	49
*	31	3	34	9	162	83	19909	1	1	1	16
*	32	3	29	9	136	83	14916	1	1	1	16
*	33	3	3	9	12	83	133	1	1	1	1
*	34	3	3	9	12	83	133	1	1	1	1
*	35	3	3	9	12	83	133	1	1	1	1
	36	4	12	19	100	294	8695	1	3	81	97344
*	37	3	3	9	12	83	133	1	1	1	1
*	38	3	14	15	91	202	7587	1	16	1	1
*	39	12	20	73	110	3062	7423	1	1	441	1764
	40	8	32	49	208	2763	45880	2	9	4	4225
*	41	11	43	51	260	1309	36680	1	8	6561	72900
*	42	10	20	69	197	5840	54375	3	9	4	49
	43	3	6	9	29	83	831	1	1	16	1
*	44	14	13	76	96	3489	4283	1	7	4	9

Design and Source Metric Values for Project: XWORD continued ...

Metric Values										
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
*	45	32	29	181	209	22933	55528	8	13	5184	9801
*	46	13	13	76	80	4186	6696	2	1	4900	9216
*	47	15	25	124	244	19500	78712	6	10	81	1764
*	48	18	13	119	106	12410	12336	7	7	400	441
	49	9	30	82	312	9602	117530	5	10	1	196
*	50	13	13	76	80	4186	6696	2	1	5184	9604
*	51	15	26	124	245	19500	79035	6	10	81	1764
*	52	18	13	119	106	12410	12336	7	7	400	441
*	53	9	30	82	312	9602	117530	5	10	144	196
*	54	16	11	107	99	10614	11575	7	7	1936	1521
*	55	17	10	90	120	6489	11414	1	12	36	400
*	56	14	17	98	112	8563	16212	2	2	900	3969
*	57	6	13	29	121	517	20643	1	8	4	9
*	58	8	15	35	98	1030	14137	1	4	49	4096
*	59	3	20	9	91	83	11596	1	4	1	16
*	60	16	5	97	21	6994	266	1	1	324	5184
*	61	14	28	62	159	2586	21267	1	6	64	2401
*	62	4	14	19	56	236	5094	1	2	1	36

Table 40. Design and Source Metric Values for Project: YAHTZEE

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	30	79	200	582	16651	92955	2	44	5929	894916
*	2	14	27	79	159	9118	22243	6	10	9	900
*	3	4	3	22	22	456	587	1	1	9	36
*	4	19	19	114	119	12082	23247	3	6	5184	7744
*	5	8	21	23	108	348	11456	1	5	576	28900
*	6	7	26	26	271	533	52559	1	18	225	225
*	7	6	6	32	30	1319	2080	1	1	196	324
*	8	5	4	26	21	609	648	1	1	400	625
*	9	16	2	153	170	10708	17783	14	13	36	36
*	10	4	3	25	20	781	571	1	1	4	4
*	11	10	8	57	52	3626	2922	1	2	64	1296
*	12	7	8	47	67	4217	7140	2	4	81	7056
*	13	9	25	20	134	254	16071	1	5	36	1225
*	14	14	26	85	179	7325	28640	2	2	9	36
	15	21	11	139	47	11380	3122	1	3	30976	1
	16	10	11	60	47	4388	3122	1	3	36	1
*	17	8	32	49	123	3868	4640	2	3	1	100
	18	23	100	54	835	1146	119528	1	48	100	66748900
	19	26	7	57	55	1209	3558	1	3	144	64
*	20	26	5	57	40	1209	3025	1	3	144	196
	21	22	45	53	457	982	82848	1	31	100	116964
	22	25	46	56	608	1038	184671	1	45	36	57600
*	23	22	44	125	499	14432	113251	4	35	12100	43264
*	24	8	37	23	417	348	89308	1	26	576	18225
	25	5	30	32	330	1536	46179	3	17	36	9604
	26	10	19	57	127	3626	34034	1	10	100	17424
*	27	11	21	37	108	961	11456	1	5	900	28900
	28	5	10	16	69	203	7517	1	7	1	144
*	29	8	8	27	52	408	2922	1	2	256	1296
	30	16	152	81	1283	9518	275881	7	79	784	21372130
	31	9	11	36	70	1713	6324	3	3	1600	484
	32	5	17	16	111	203	11802	1	6	1	1296
	33	14	9	85	82	7288	17369	5	8	2304	784
	34	5	8	16	44	203	6213	1	2	9	3136
*	35	7	50	47	395	4217	120023	2	27	100	2916
*	36	9	15	20	101	254	12308	1	3	64	3136
*	37	11	14	51	94	4605	11925	4	4	16	400
	38	6	8	14	67	236	7140	1	4	1	7056
	39	13	21	48	114	4804	13202	4	5	9	1024
*	40	5	10	26	98	758	13753	2	12	1	64
*	41	5	10	26	74	758	12686	2	6	1	4
*	42	11	17	17	98	158	17428	1	9	1	9

Table 41. Design and Source Metric Values for Project: YAHTZEE2

Metric Values											
	PROC NUM	LOC ADL	LOC PAS	N ADL	N PAS	E ADL	E PAS	CC ADL	CC PAS	INFO ADL	INFO PAS
	1	24	126	150	806	10124	102579	1	30	14400	20793600
	2	15	22	79	105	5218	10607	2	2	1	6400
*	3	11	11	93	93	12161	14863	2	3	625	2500
*	4	7	7	44	43	2575	2572	1	2	4	36
*	5	8	7	49	43	3059	2572	1	2	4	36
*	6	7	6	47	43	2890	2331	1	2	4	4
	7	3	115	16	1136	242	788816	1	8	16	3969
*	8	8	39	44	265	2083	62534	2	4	9	324
*	9	4	111	17	605	366	251361	1	20	1	9
*	10	22	21	146	151	31577	39864	8	6	256	256
*	11	9	26	60	184	4794	36547	2	2	4	25
*	12	10	24	58	208	5214	40525	3	3	4	64
	13	9	22	43	122	1972	9959	1	1	16	6084
	14	27	112	140	809	22134	196768	4	83	1	51984
*	15	17	39	96	264	10042	56095	1	3	81	1225
	16	4	39	22	220	391	41614	1	9	1	6561
	17	11	18	85	101	7628	10702	1	2	4	900
	18	8	61	43	333	1345	35195	1	28	1	33124
*	19	12	12	78	65	7566	4695	2	2	1	9
	20	3	70	12	443	162	75279	1	13	1	900
*	21	8	11	37	74	1493	5752	1	4	1	9
	22	35	176	114	887	3258	196573	4	70	576	1375929
	23	25	34	118	192	11345	34742	5	9	4	5184
*	24	8	13	43	97	1301	9628	1	2	36	900
	25	24	26	127	253	11531	51608	12	12	16	226576
	26	46	57	285	594	124615	185620	15	37	1	1296
	27	53	332	239	3131	28590	1693053	42	173	4	885419520
	28	30	101	146	1251	20621	514837	15	59	9	1327104
	29	7	26	37	168	1188	24763	1	27	1	17424
	30	3	113	17	478	305	134515	1	53	4	608400
*	31	15	36	86	232	9129	46893	3	9	9	25
	32	20	70	90	472	2980	52962	2	16	1	327184
	33	13	44	62	347	3744	155693	2	9	1	2500
*	34	46	5	285	34	124615	741	15	1	1	9
	35	15	38	73	259	6985	50234	3	6	1	324
	36	7	77	37	525	1188	94528	1	17	1	90000
*	37	10	9	53	59	3026	3832	3	3	1	16
*	38	10	29	60	233	3849	37581	2	17	4	25

Appendix D. Data Plots

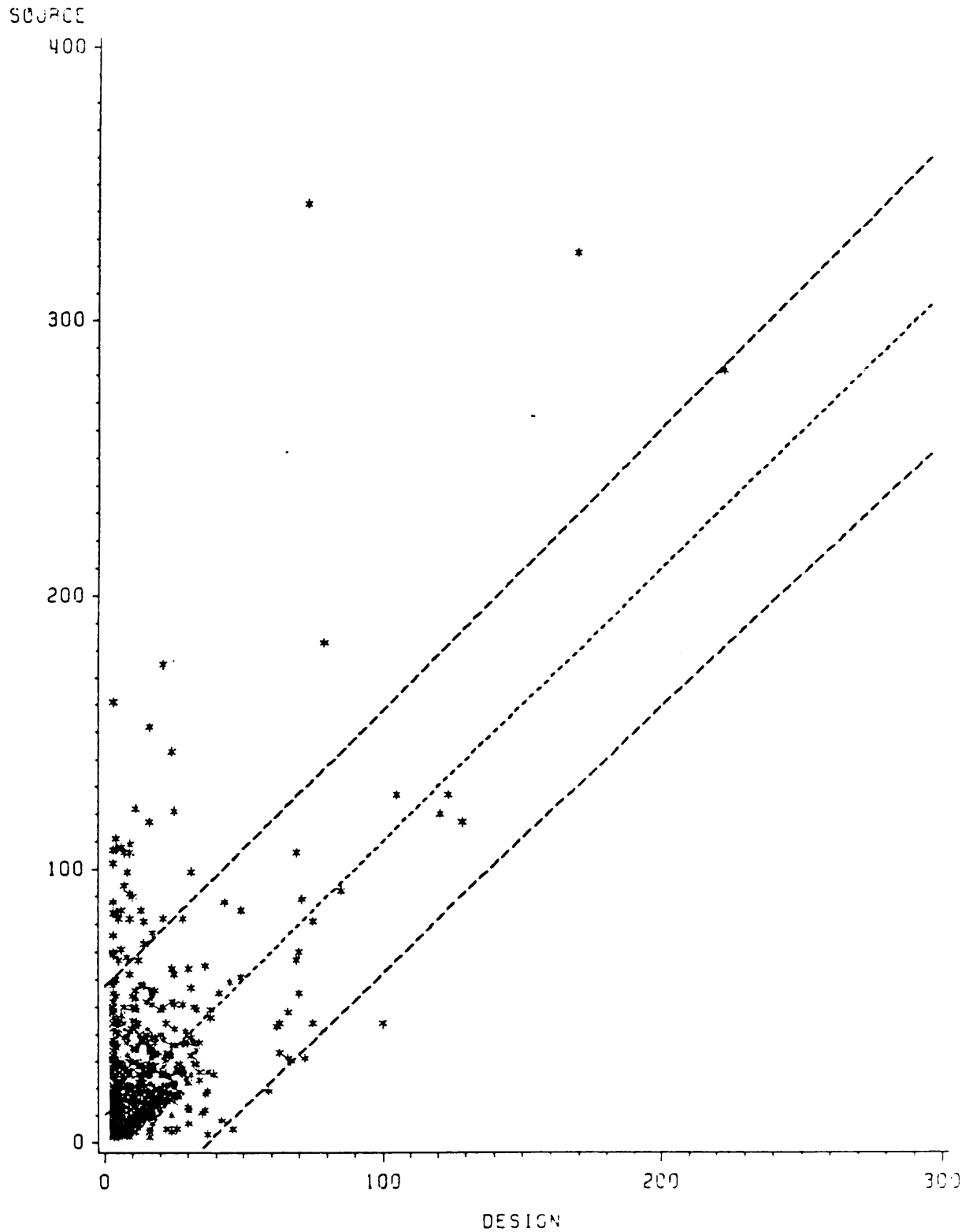

Figure 19. Regression and 95% Confidence lines for LOC

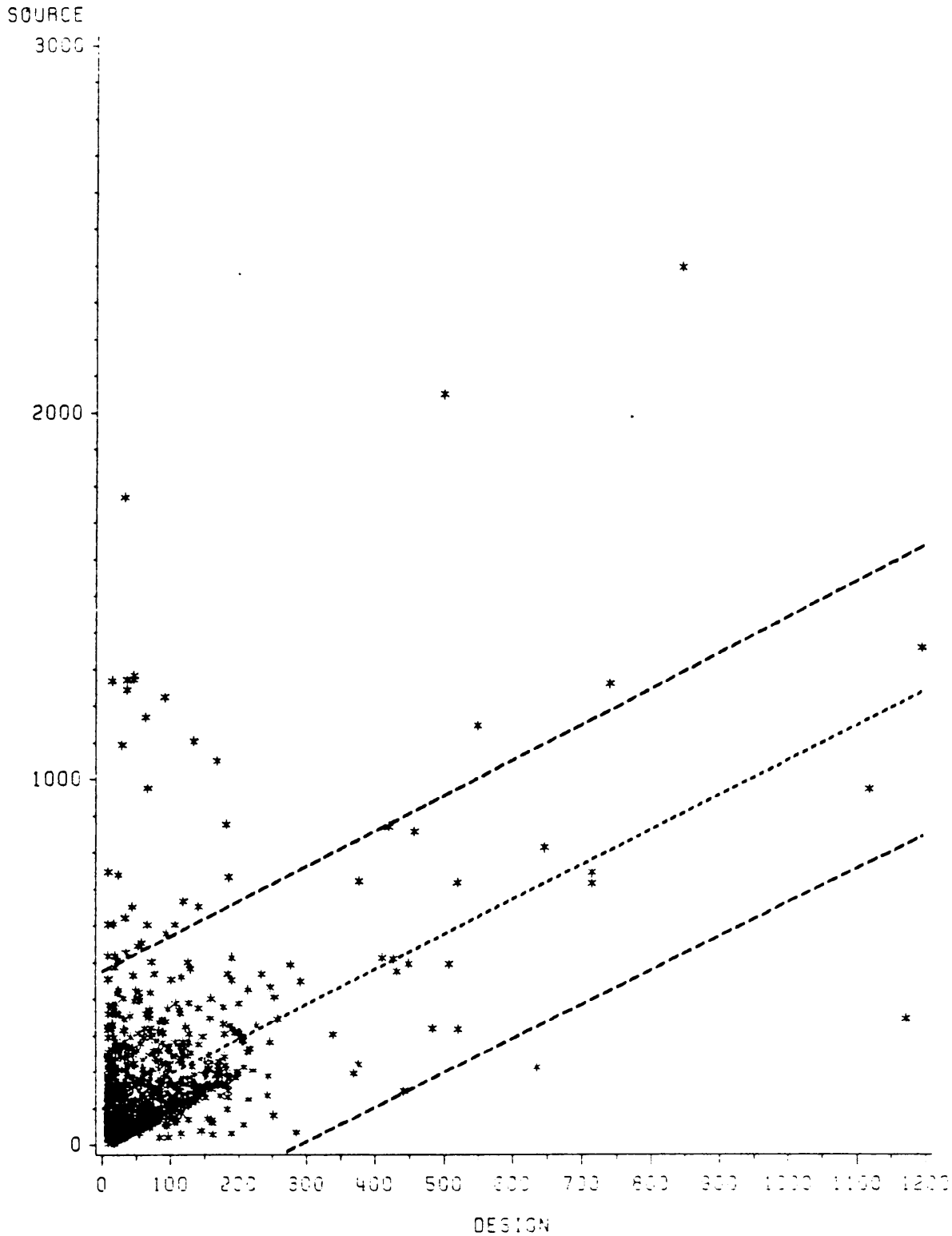

Figure 20. Regression and 95% Confidence lines for N

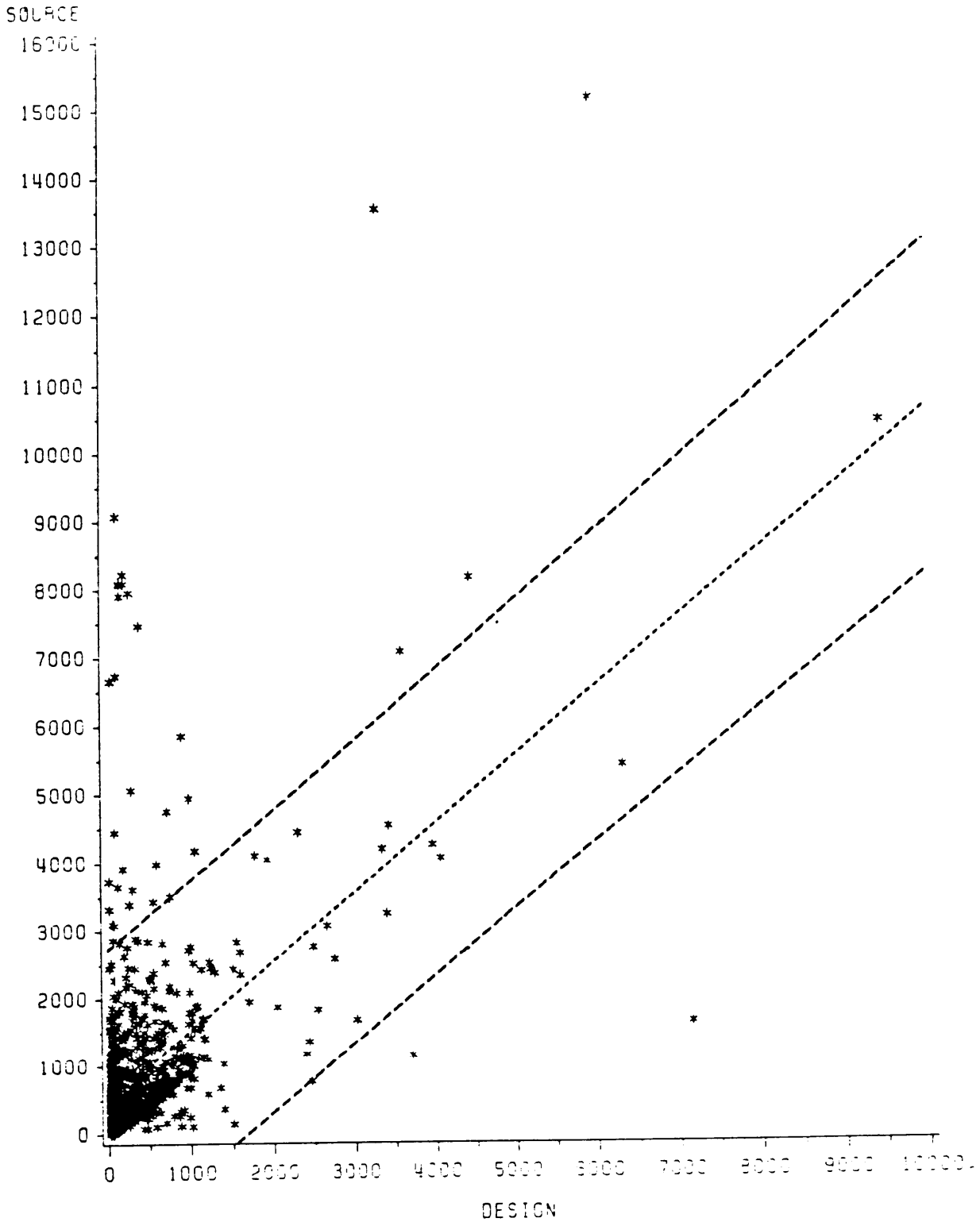

Figure 21. Regression and 95% Confidence lines for V

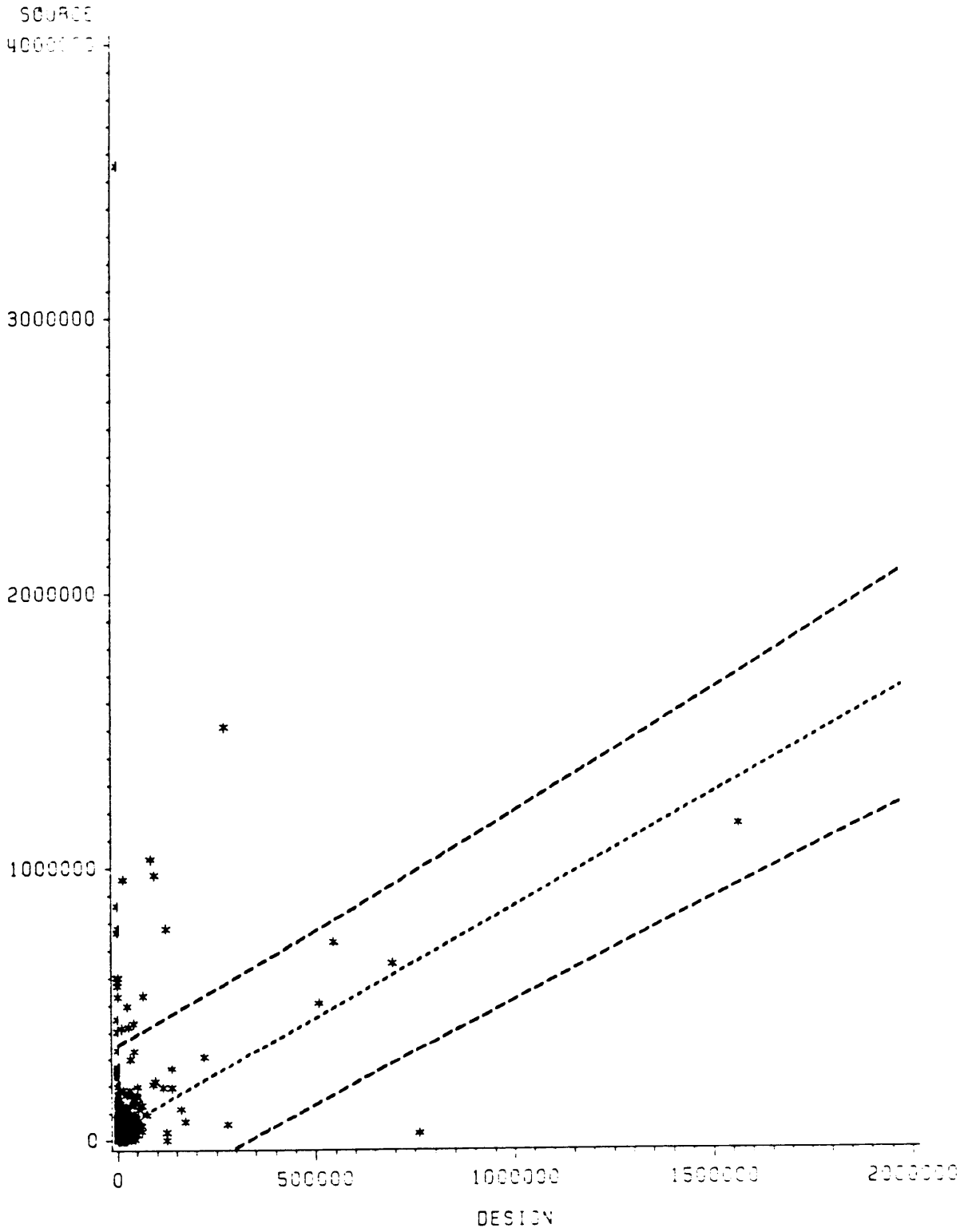

Figure 22. Regression and 95% Confidence lines for E

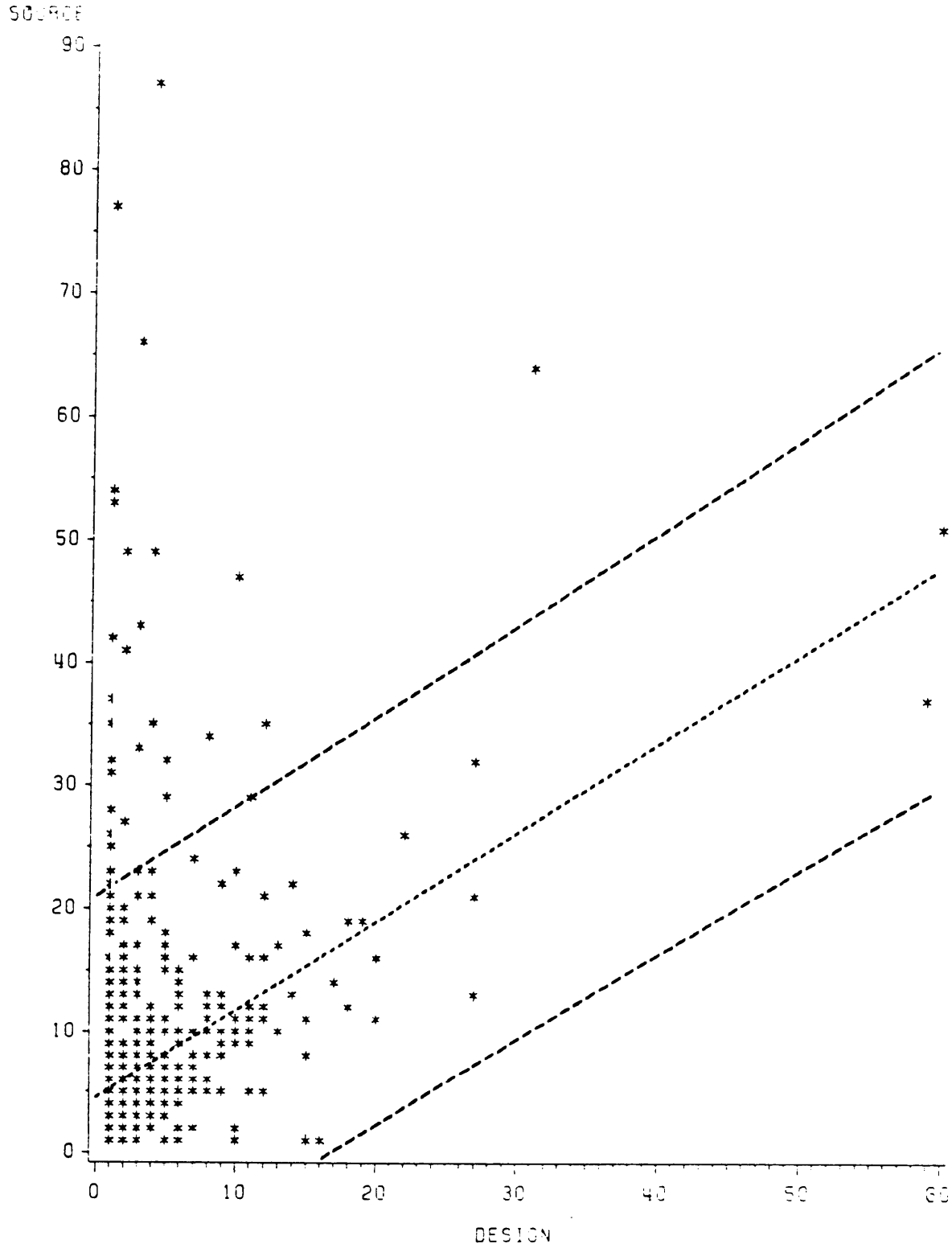

Figure 23. Regression and 95% Confidence lines for CC

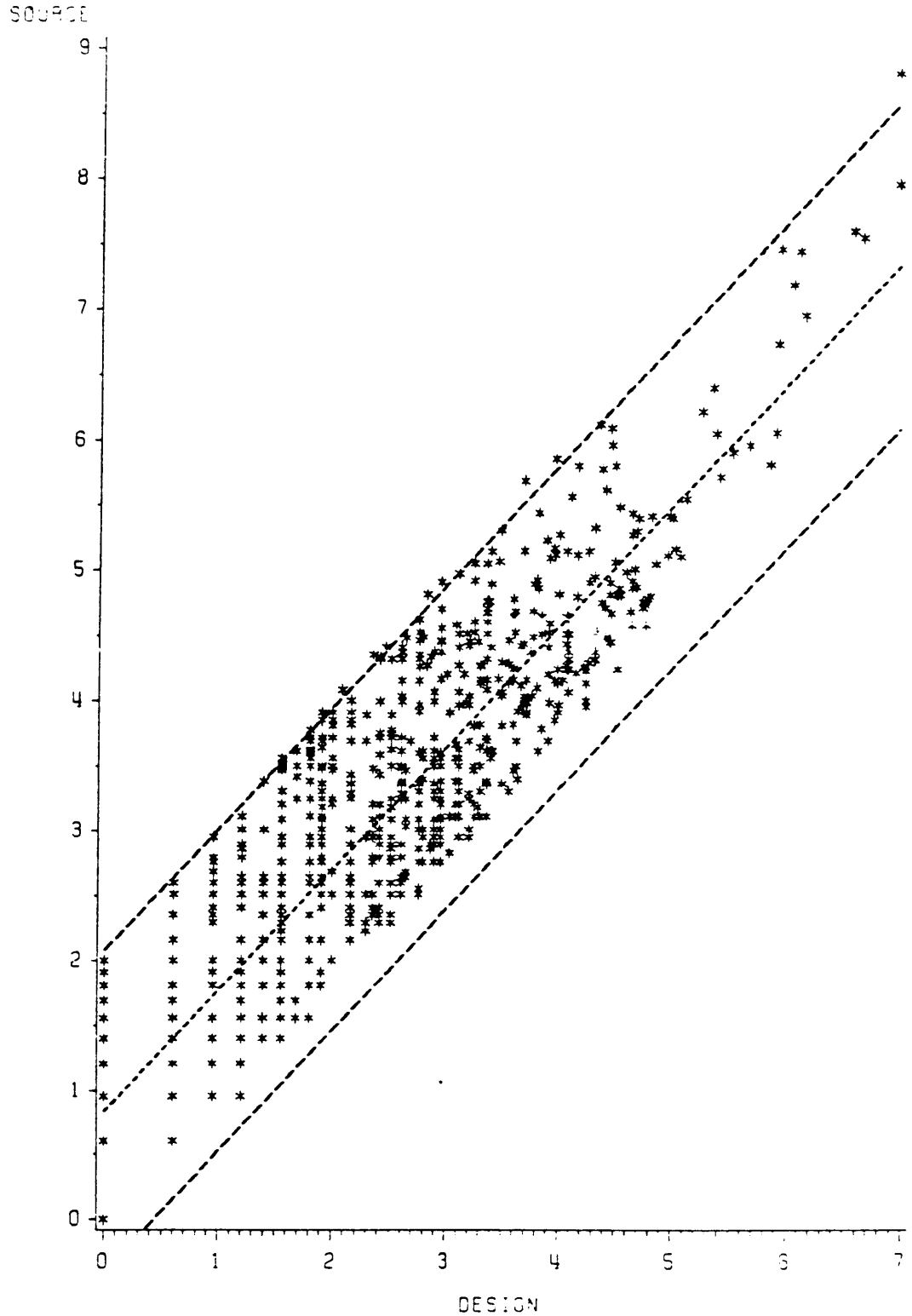

Figure 24. Regression and 95% Confidence lines for INFO

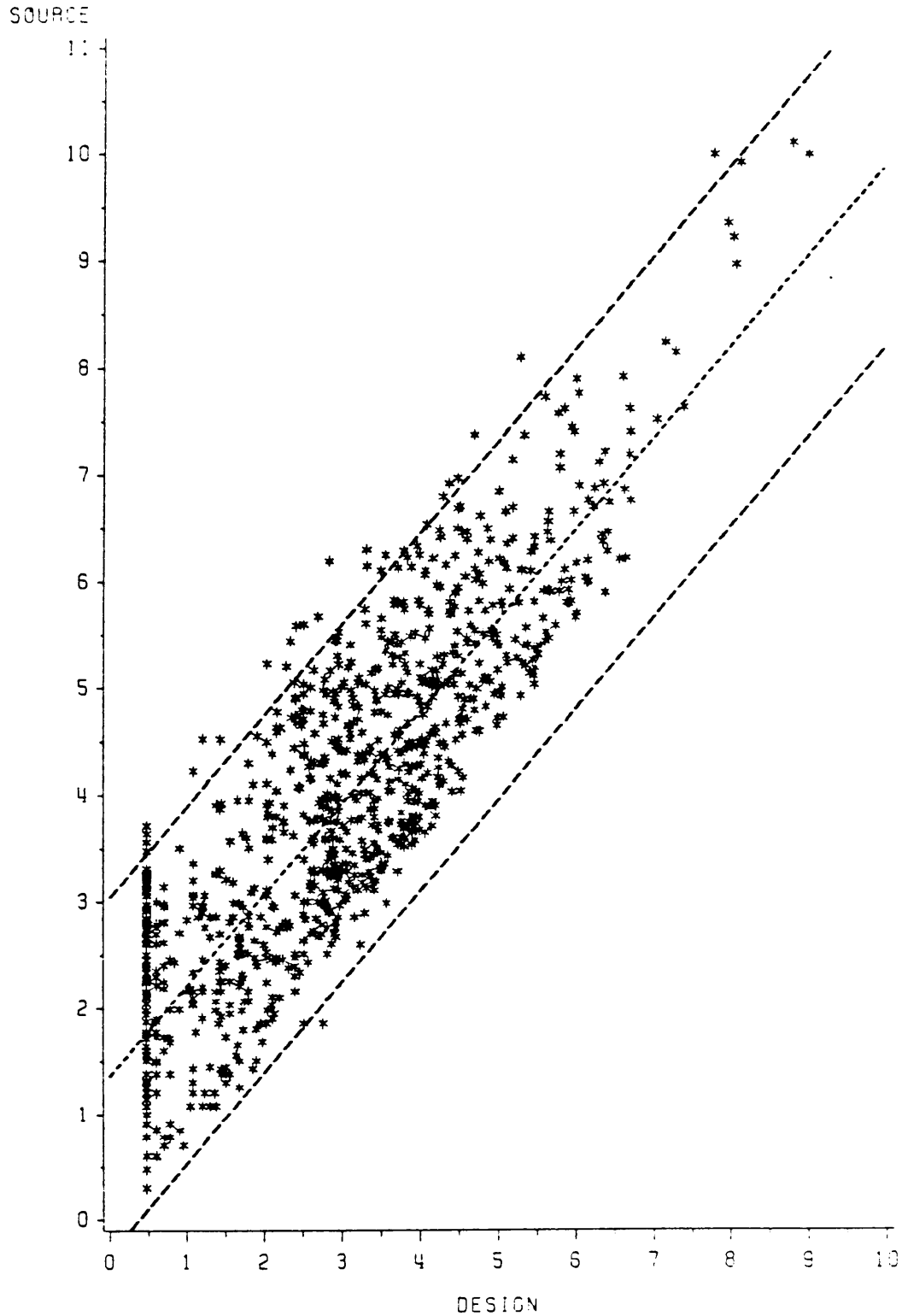

Figure 25. Regression and 95% Confidence lines for INFO-L

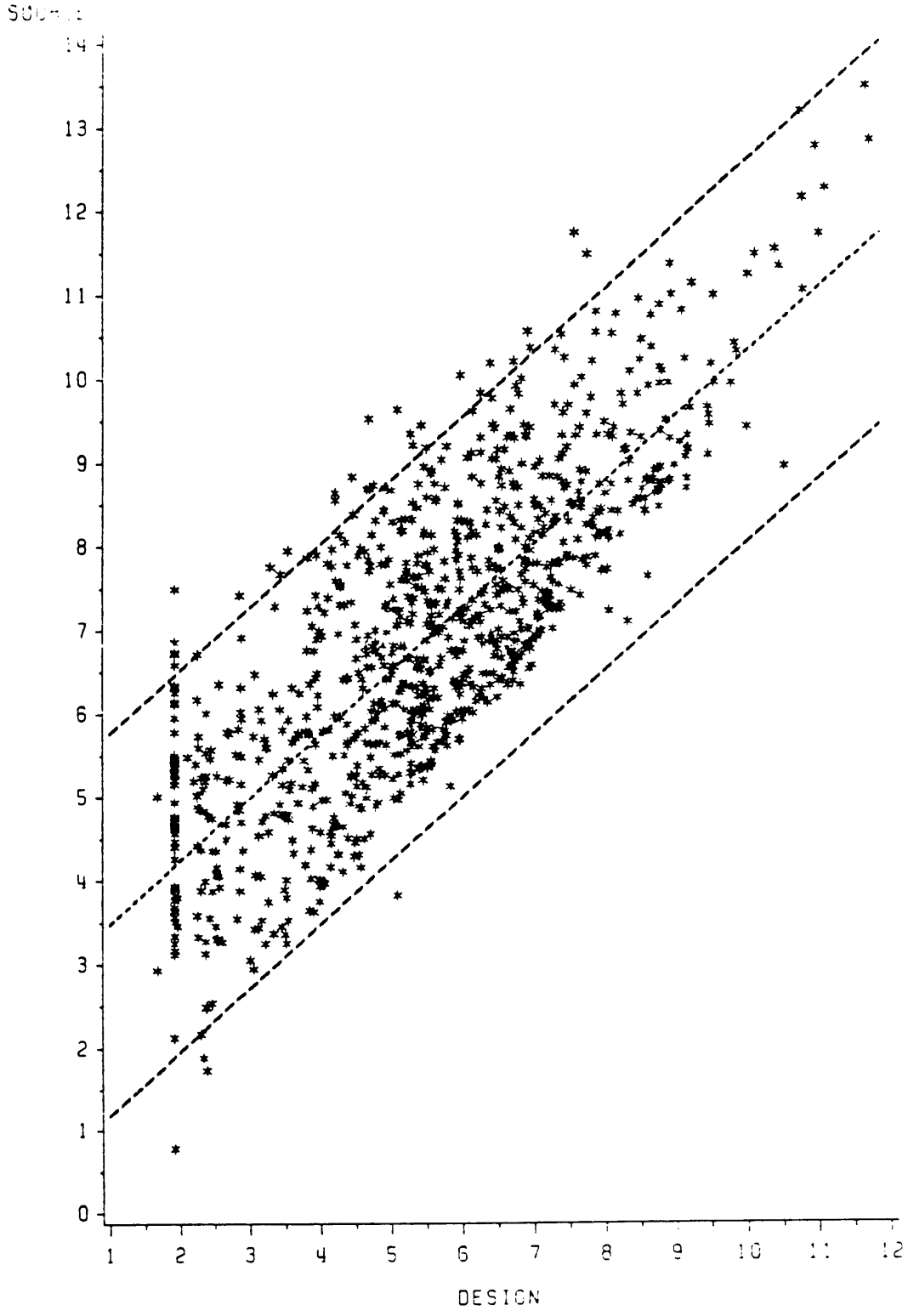

Figure 26. Regression and 95% Confidence lines for INFO-E

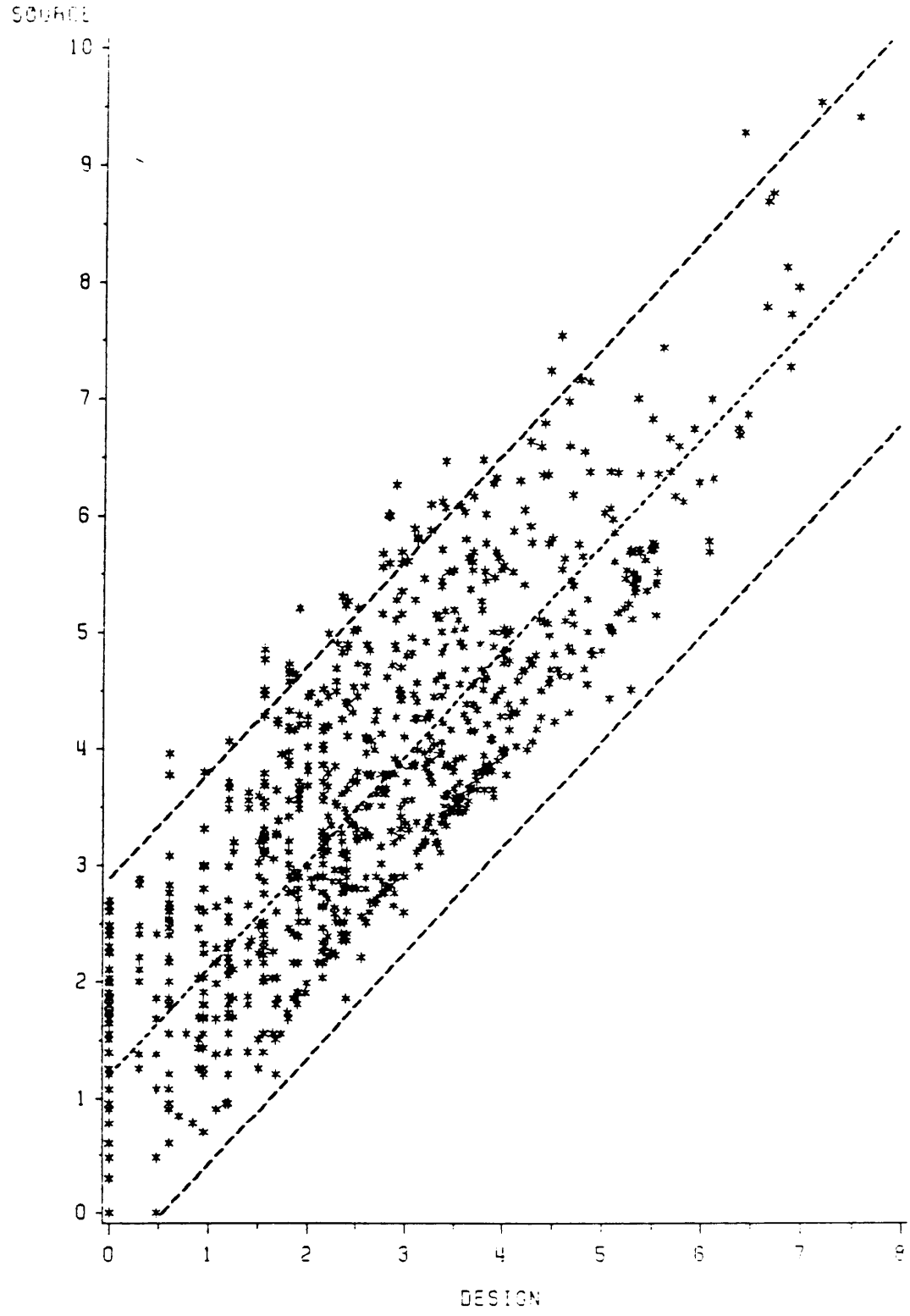

Figure 27. Regression and 95% Confidence lines for INFO-CC

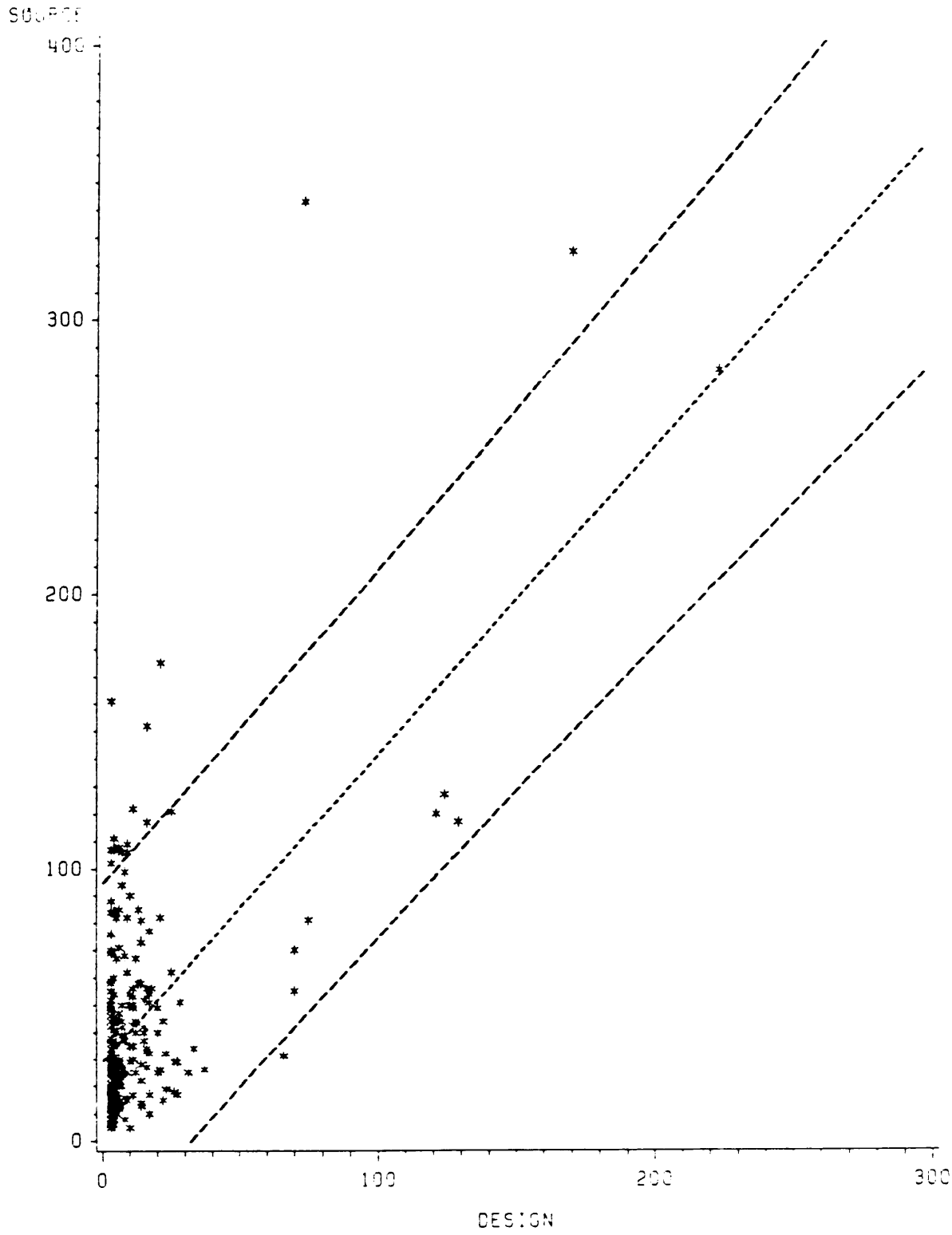

Figure 28. Low Refinement Regression and 95% Confidence lines for LOC

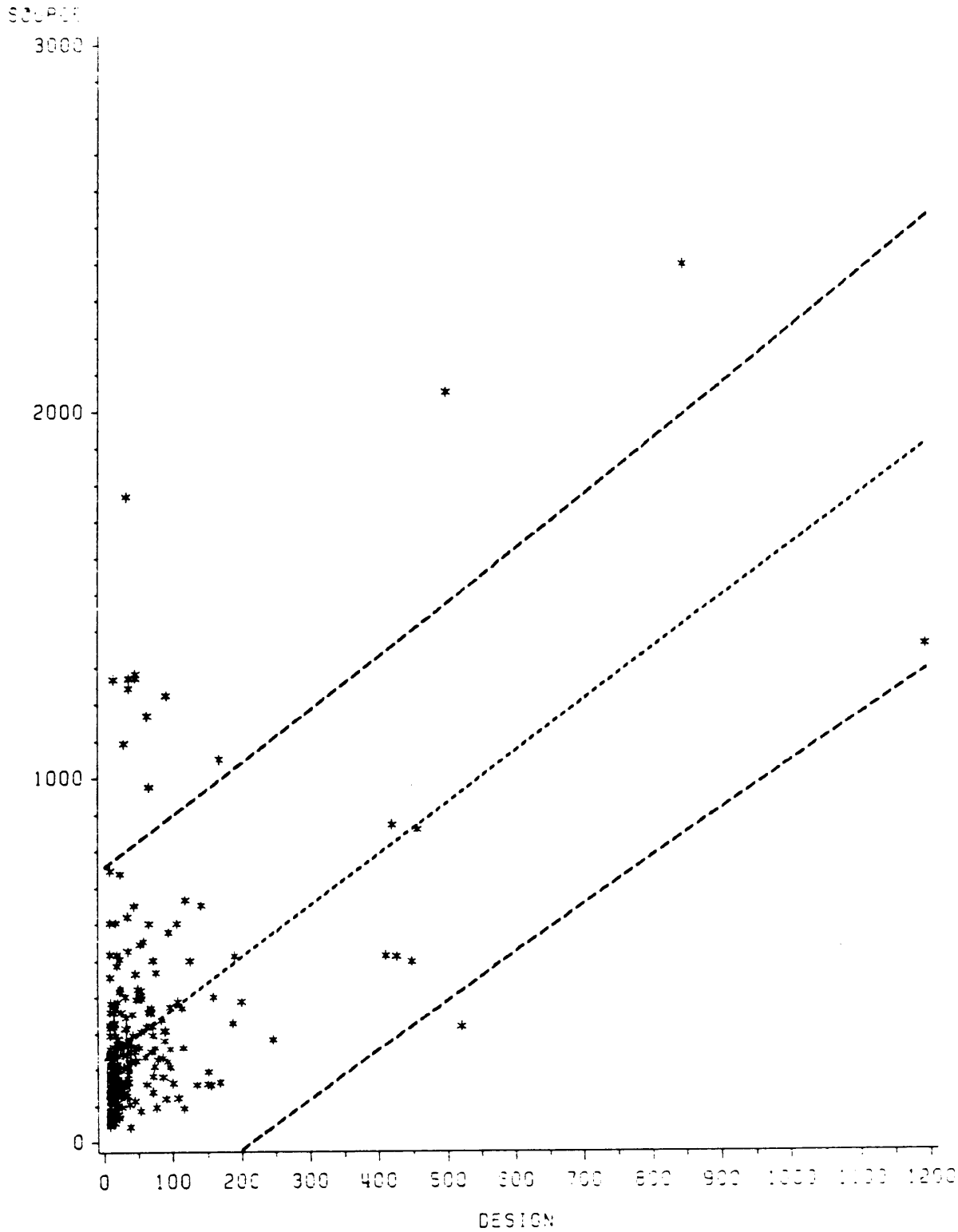

Figure 29. Low Refinement Regression and 95% Confidence lines for N

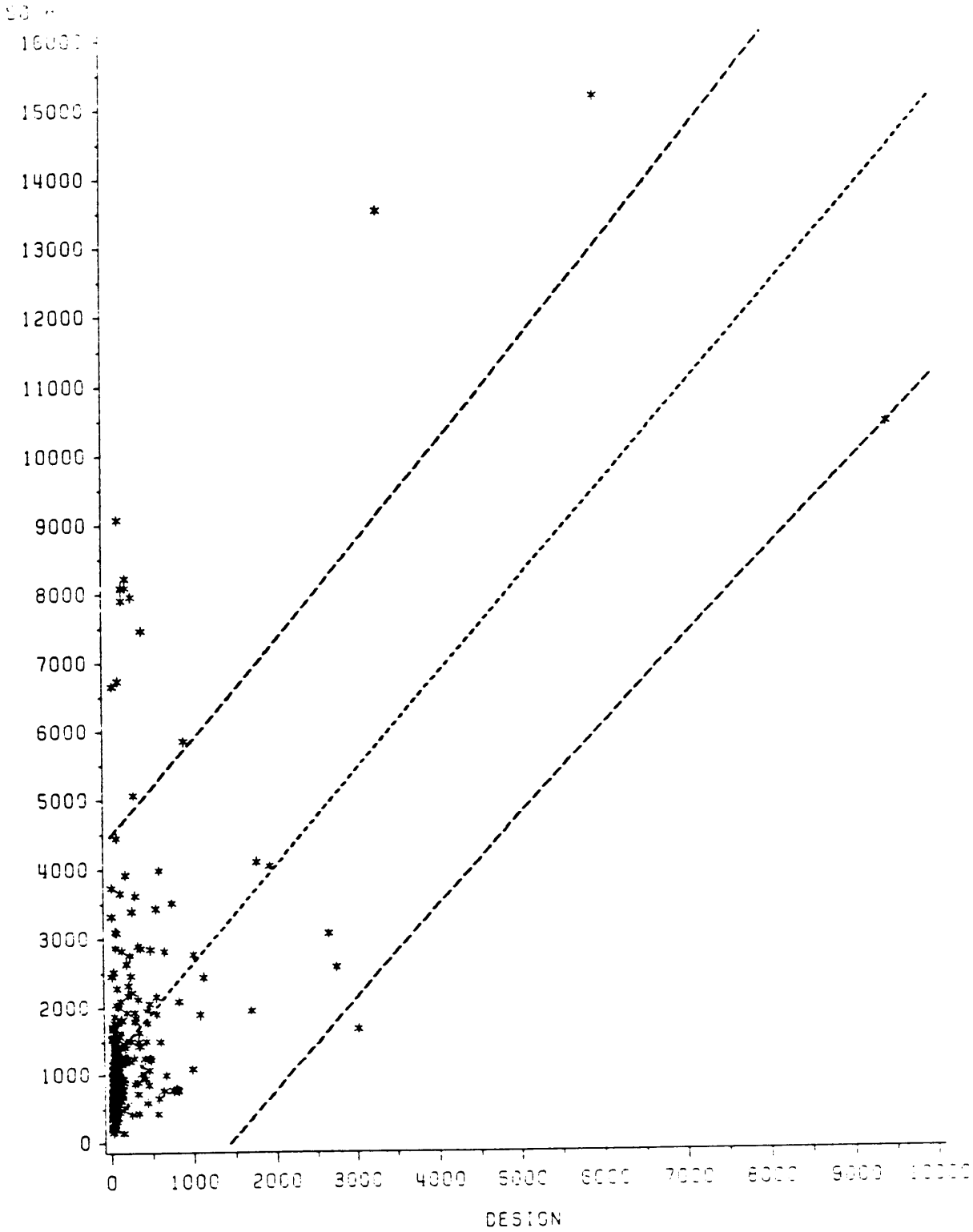

Figure 30. Low Refinement Regression and 95% Confidence lines for V

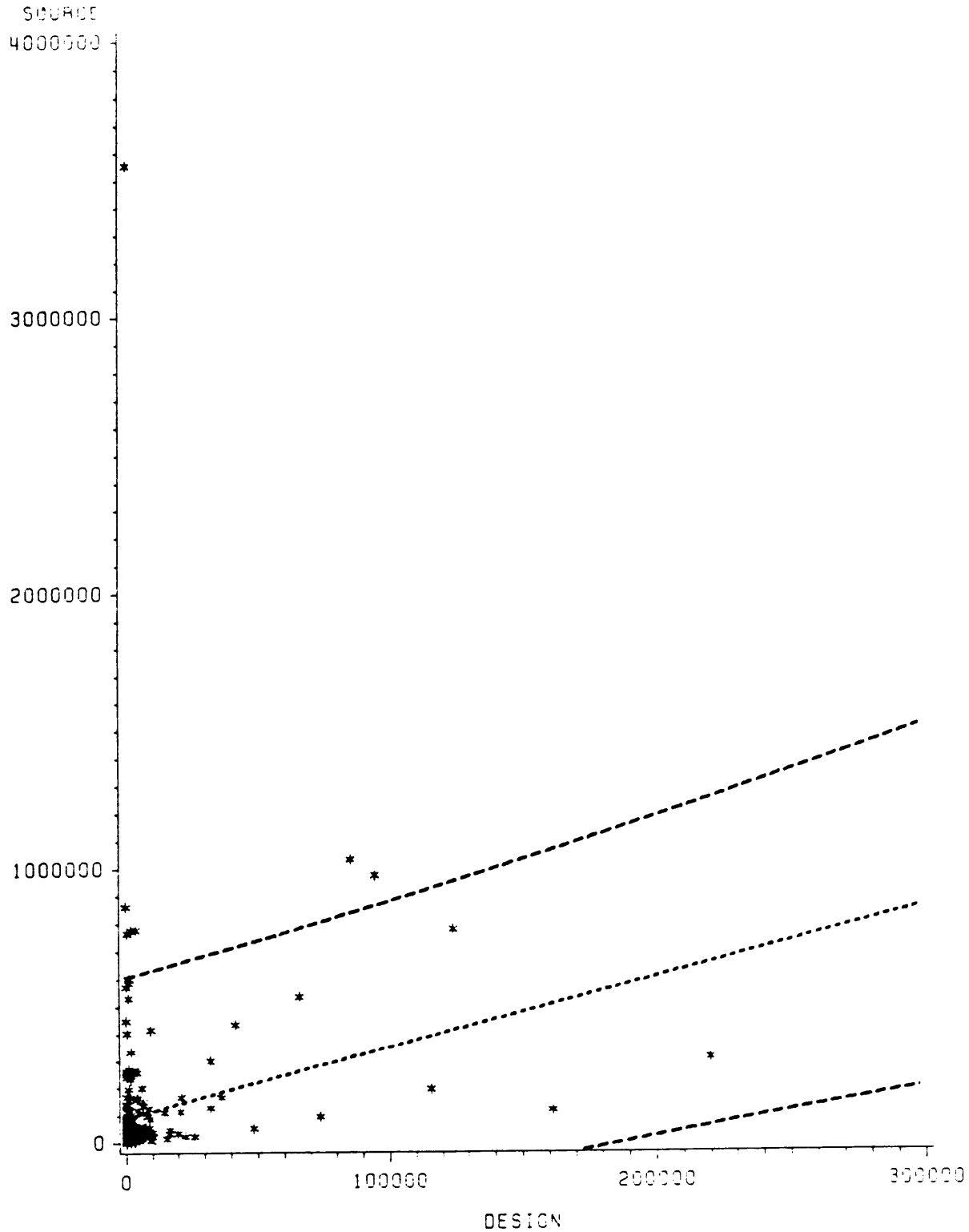

Figure 31. Low Refinement Regression and 95% Confidence lines for E

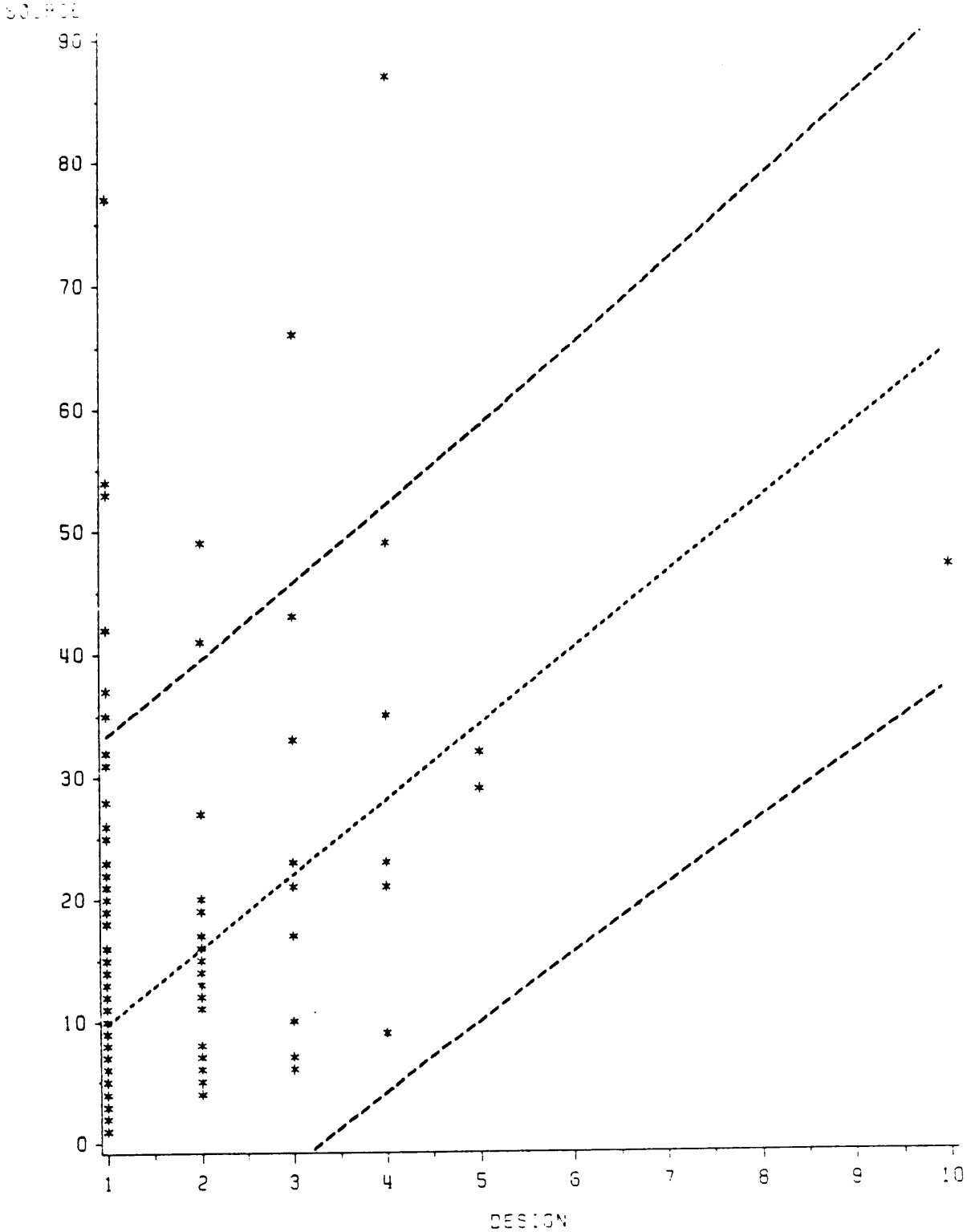

Figure 32. Low Refinement Regression and 95% Confidence lines for CC

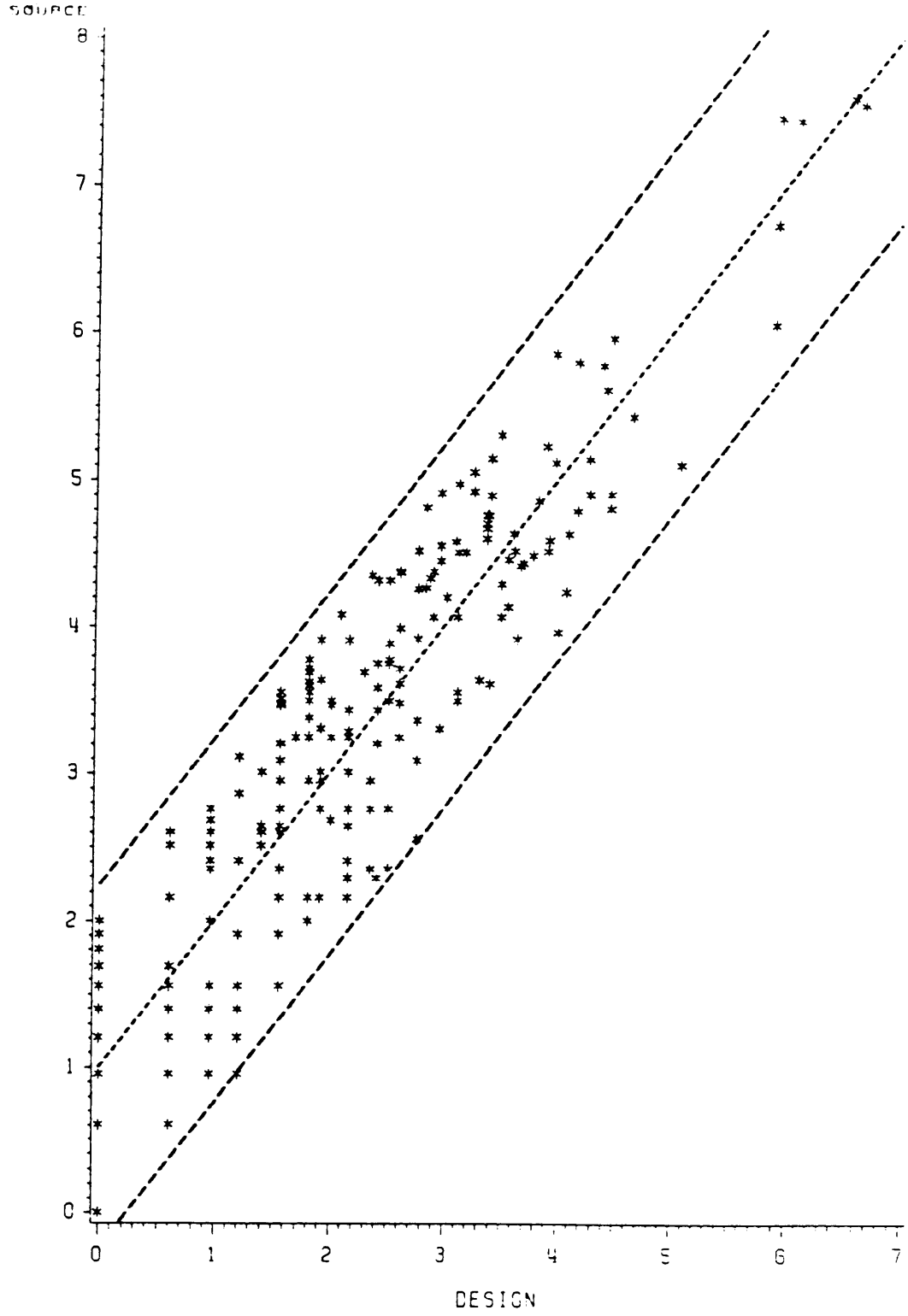

Figure 33. Low Refinement Regression and 95% Confidence lines for INFO

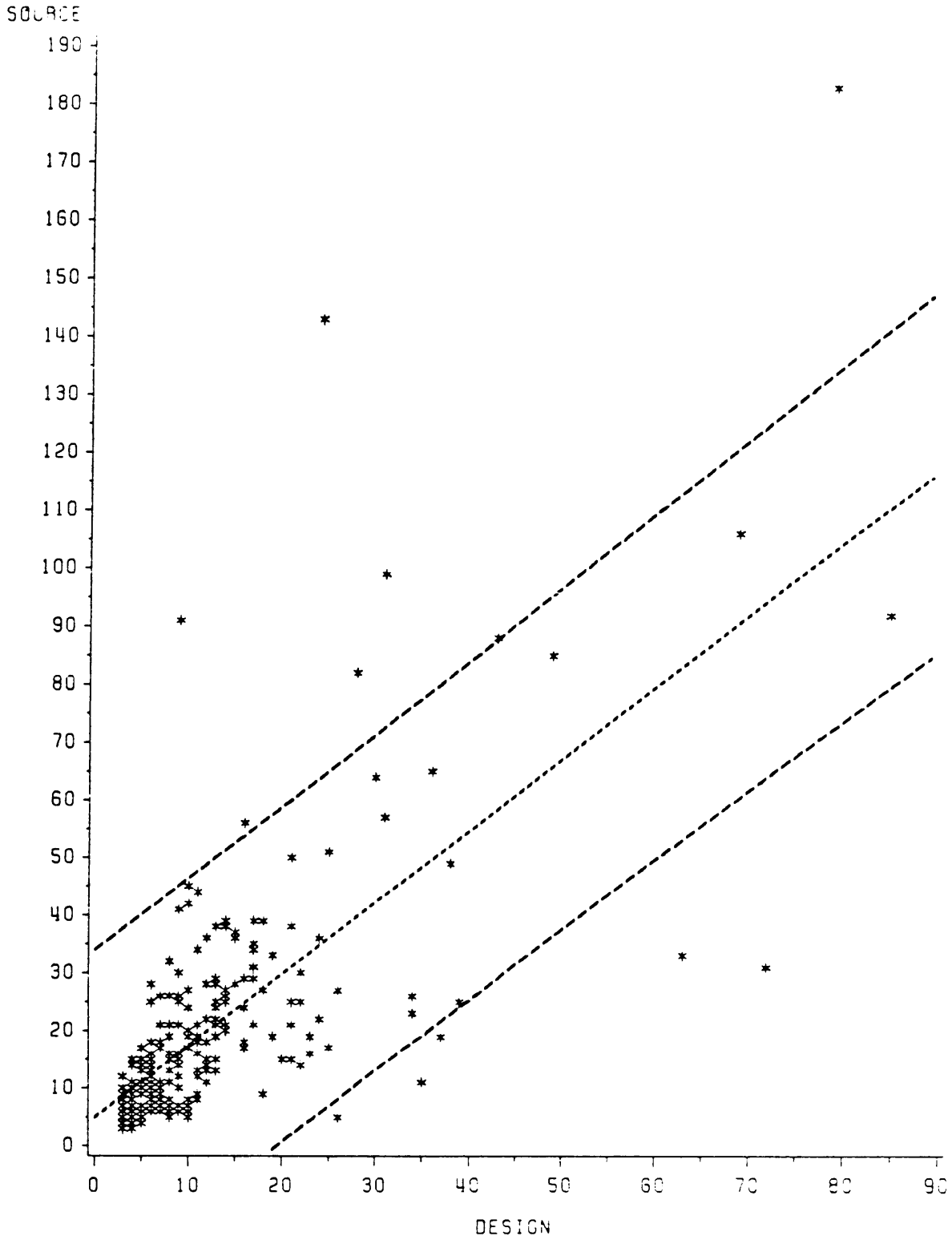

Figure 34. Mid Refinement Regression and 95% Confidence lines for LOC

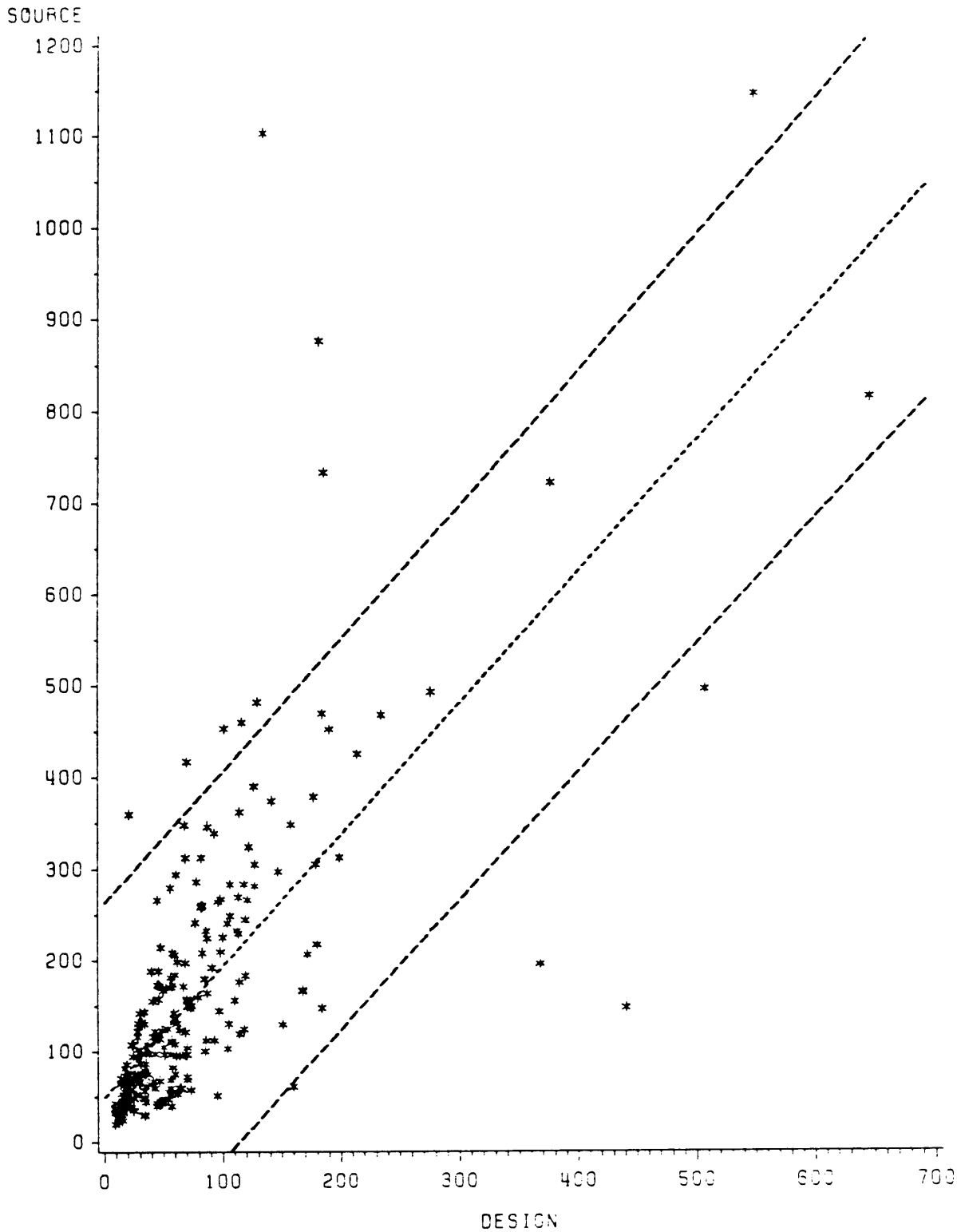

Figure 35. Mid Refinement Regression and 95% Confidence lines for N

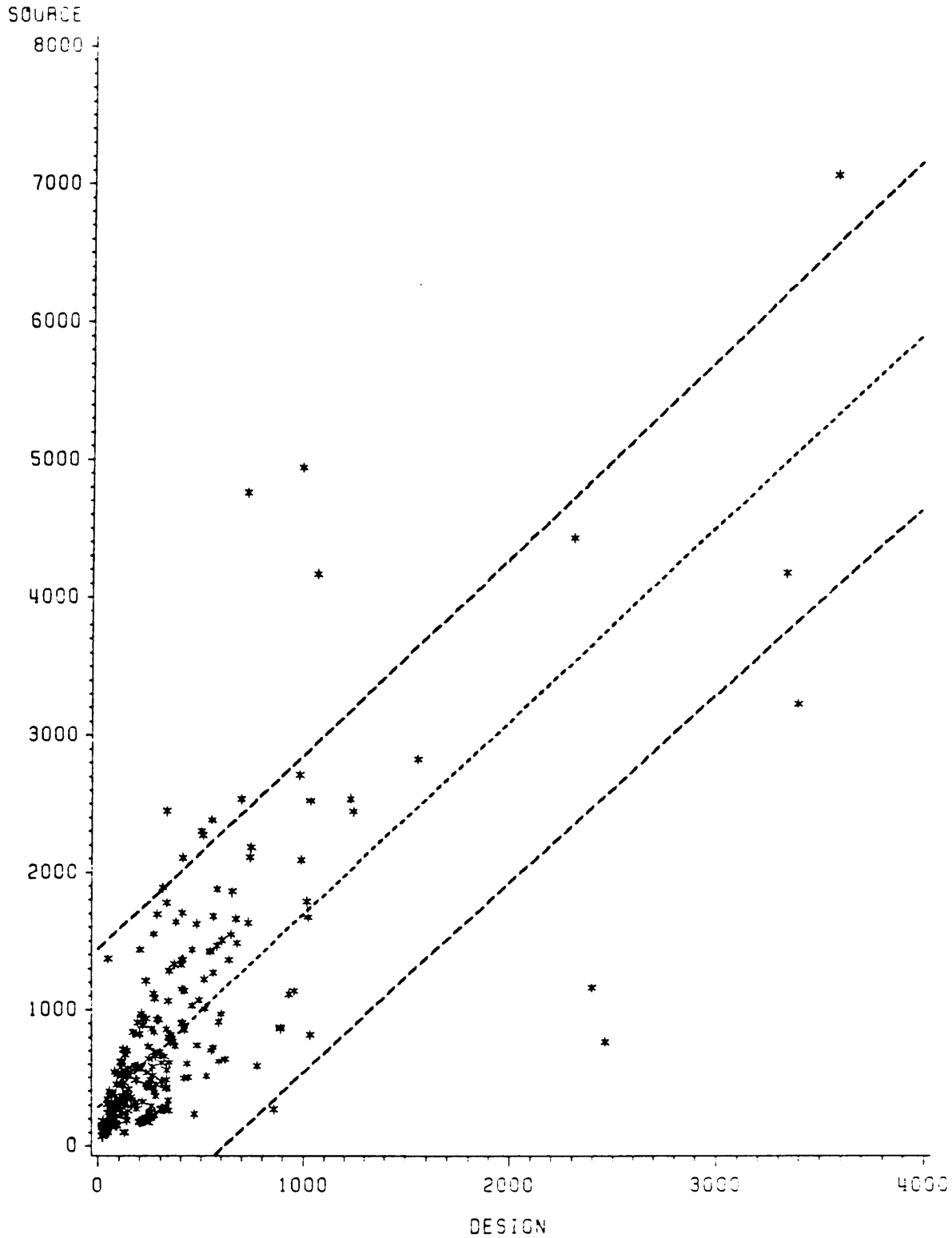

Figure 36. Mid Refinement Regression and 95% Confidence lines for V

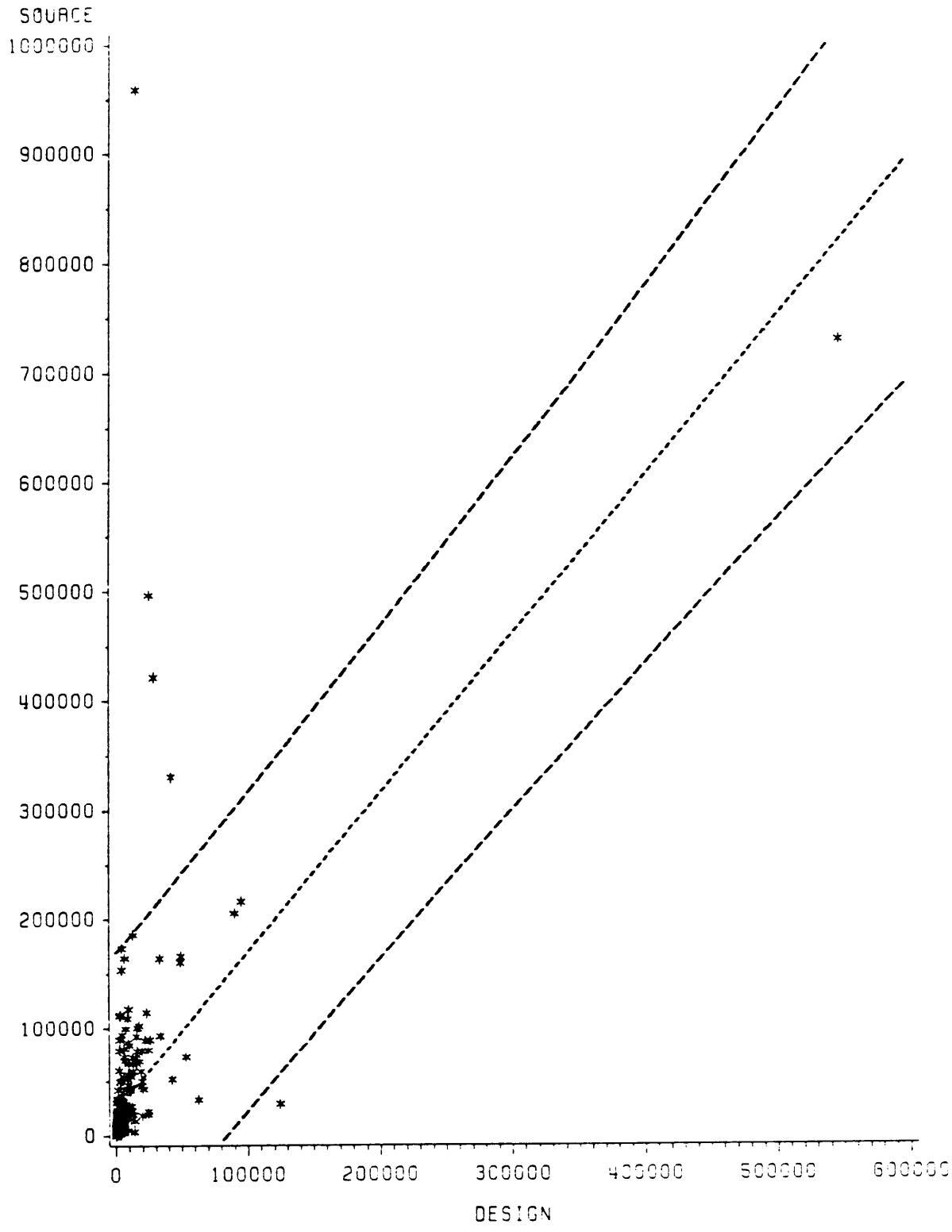

Figure 37. Mid Refinement Regression and 95% Confidence lines for E

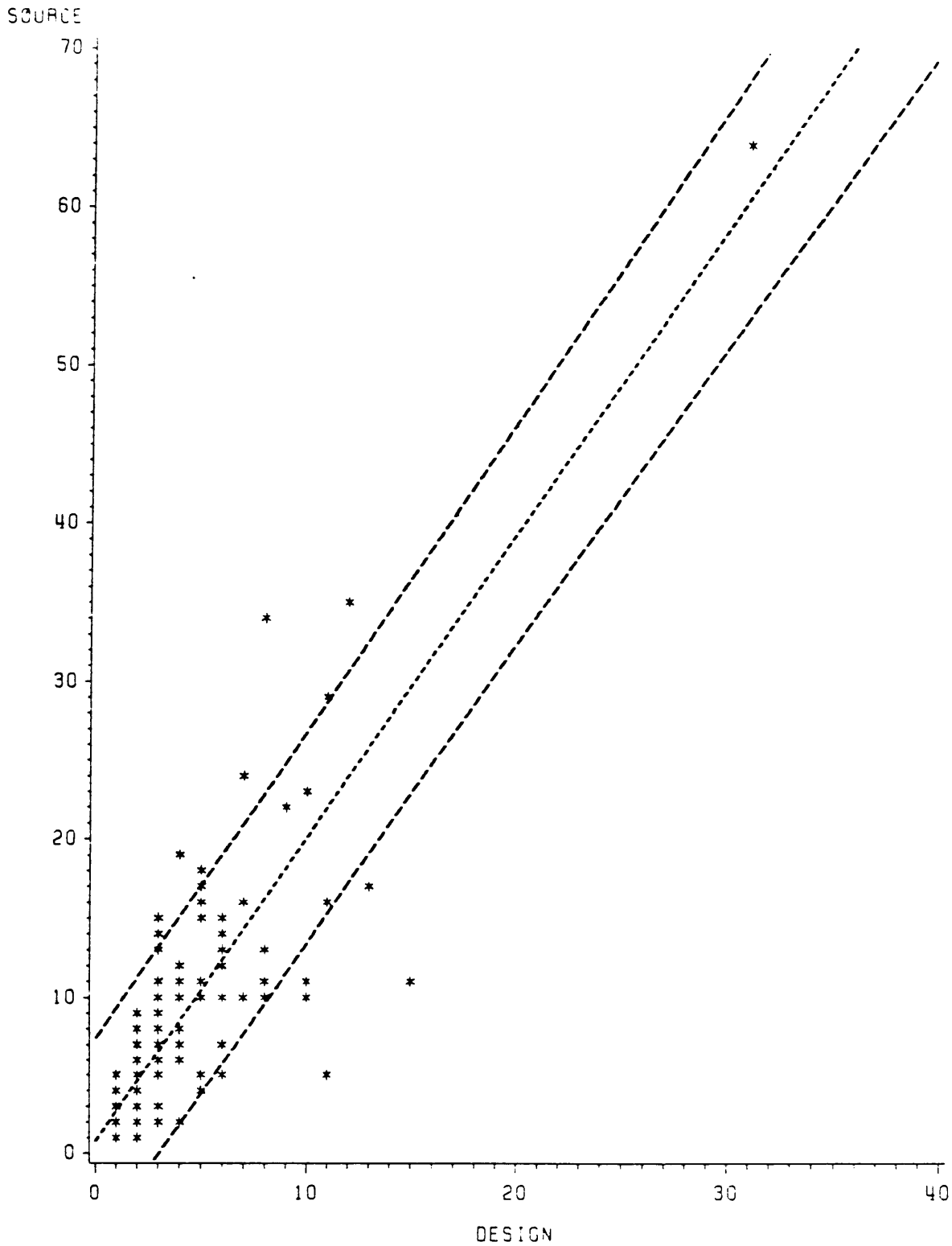

Figure 38. Mid Refinement Regression and 95% Confidence lines for CC

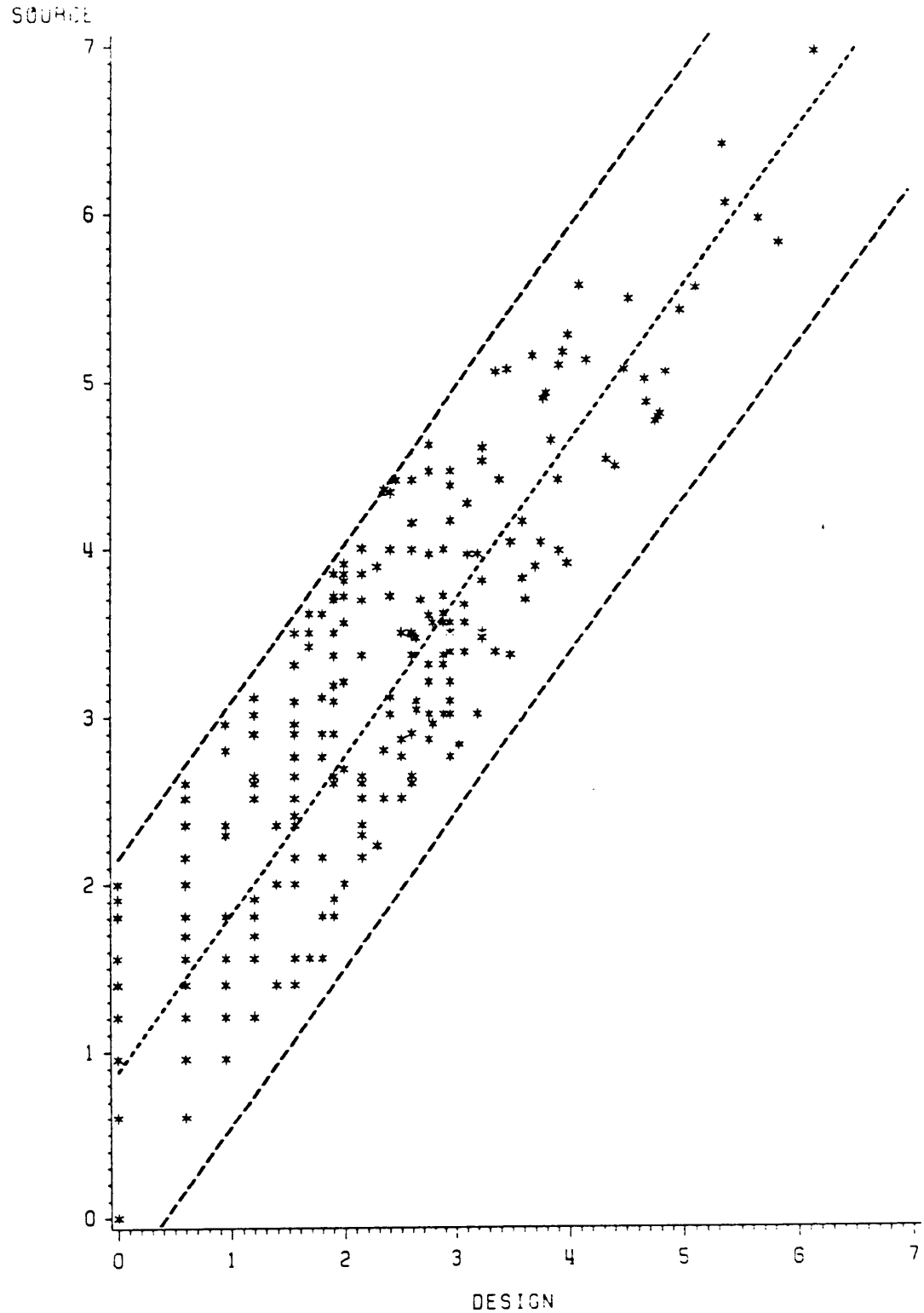

Figure 39. Mid Refinement Regression and 95% Confidence lines for INFO

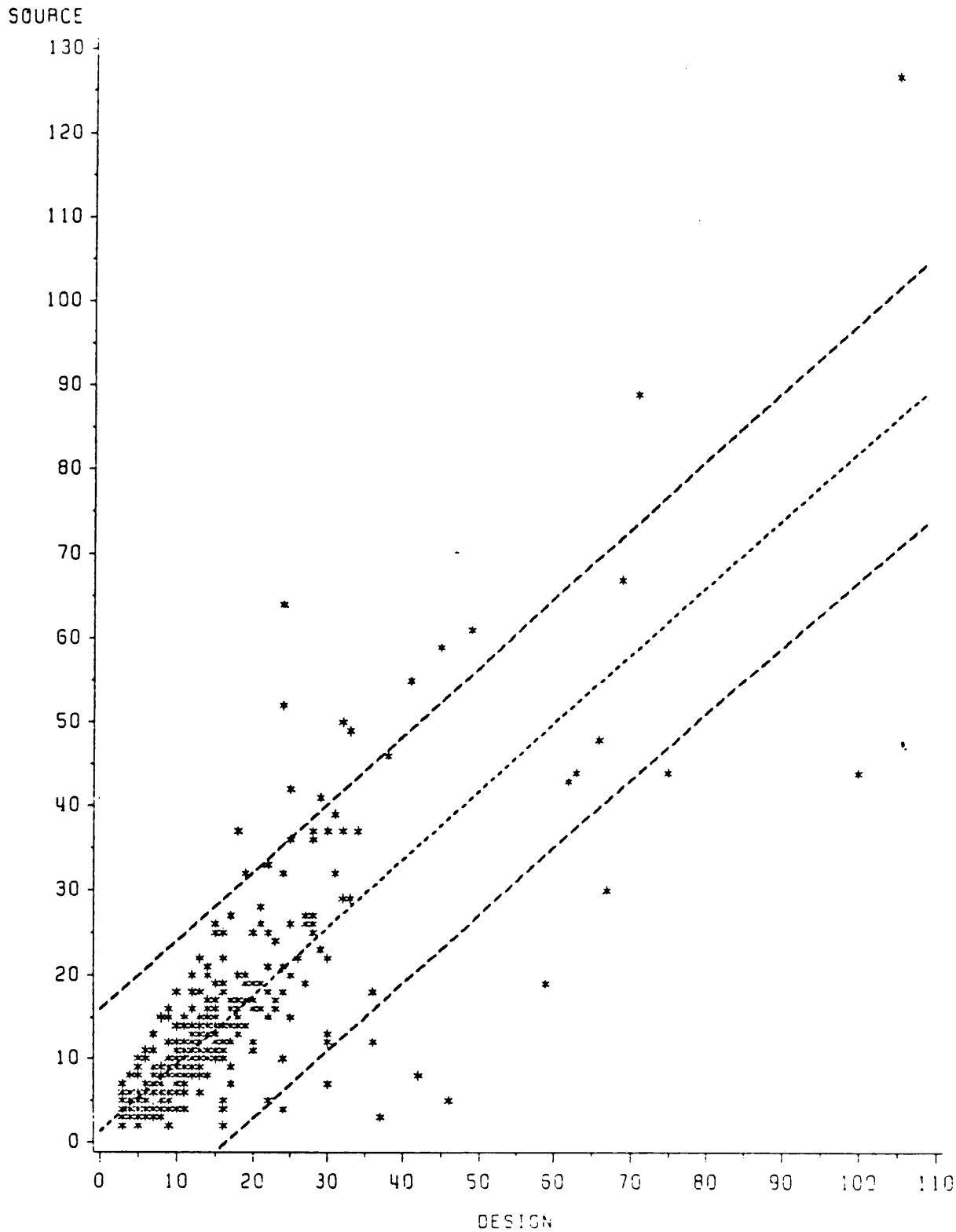

Figure 40. High Refinement Regression and 95% Confidence lines for LOC

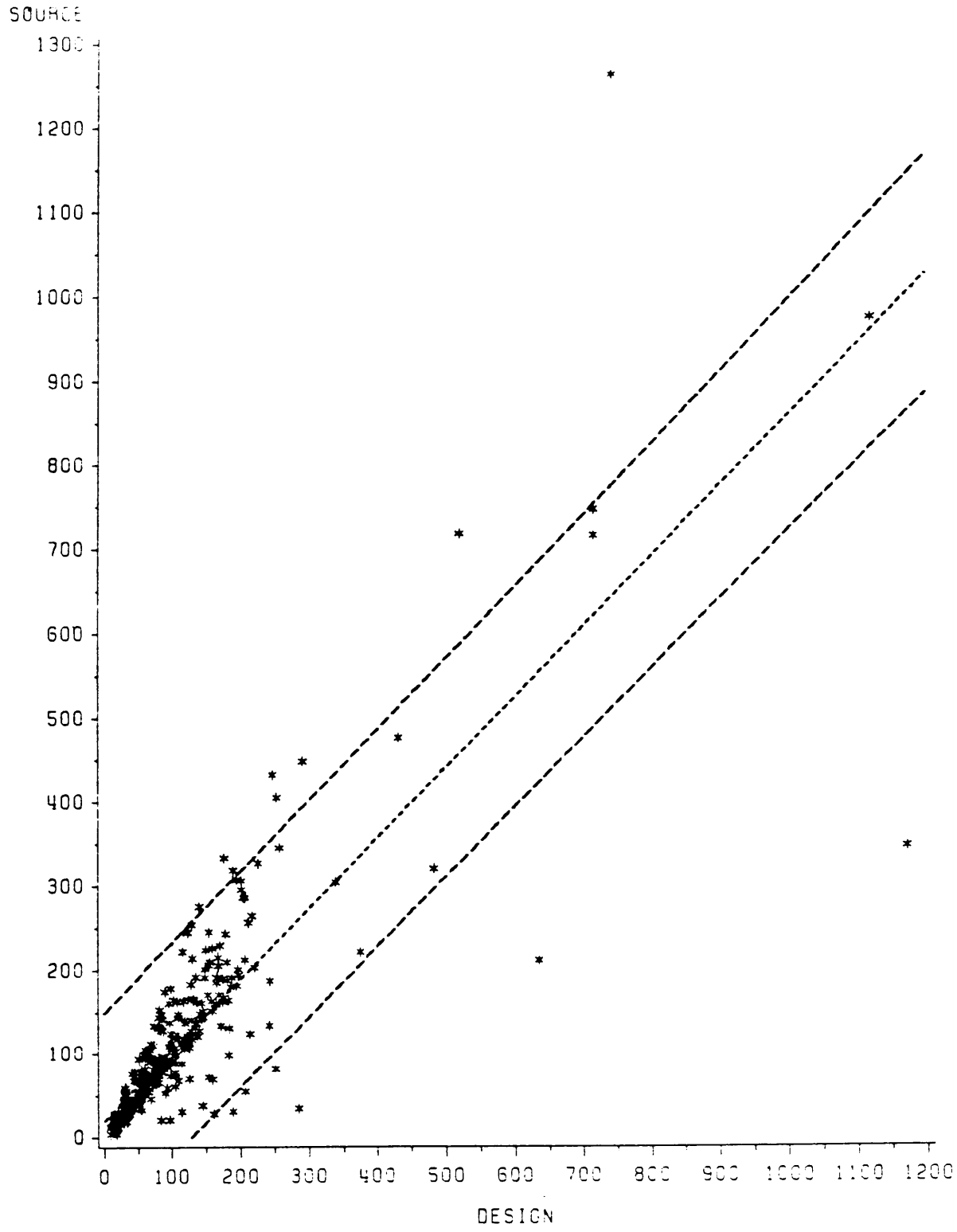

Figure 41. High Refinement Regression and 95% Confidence lines for N

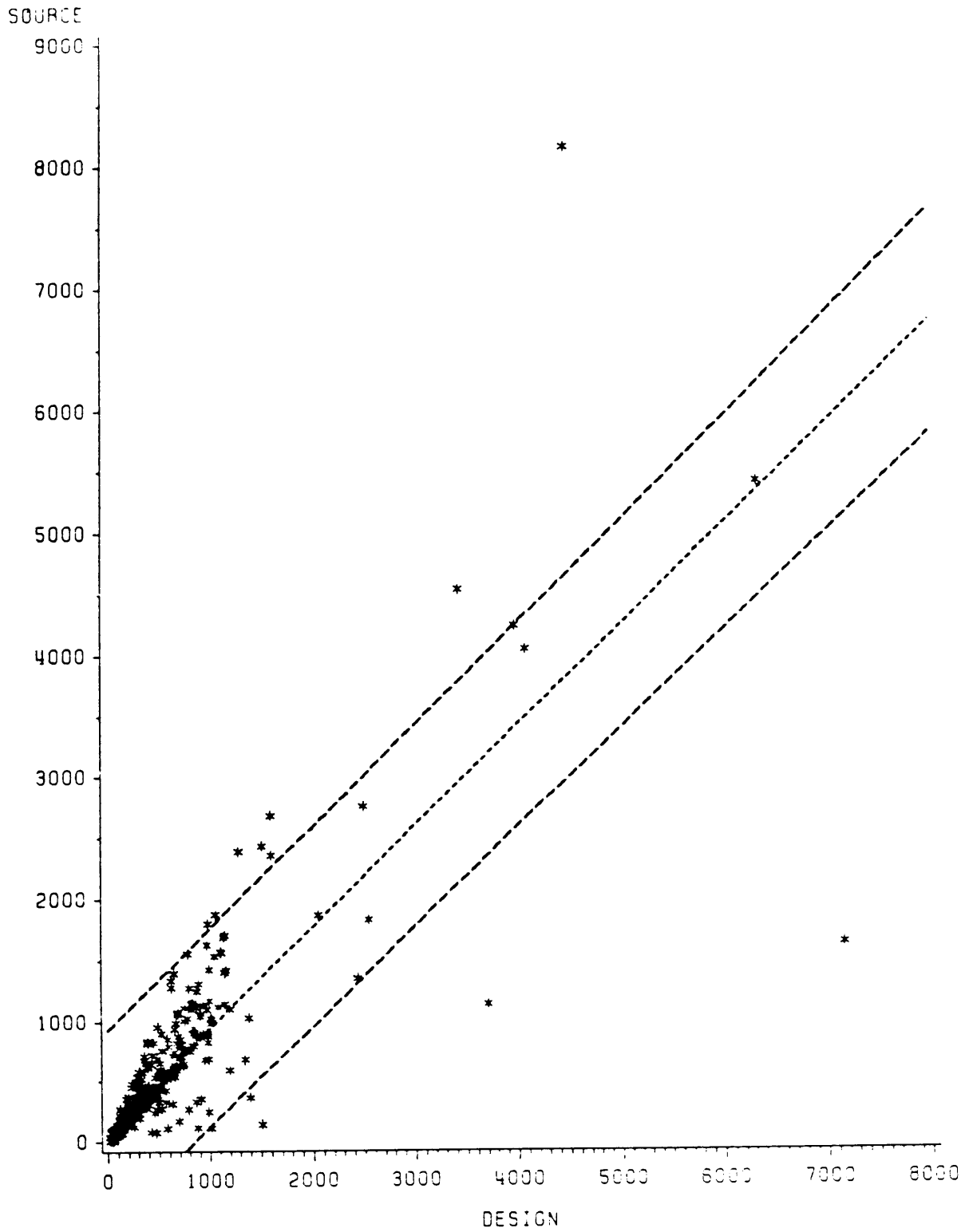

Figure 42. High Refinement Regression and 95% Confidence lines for V

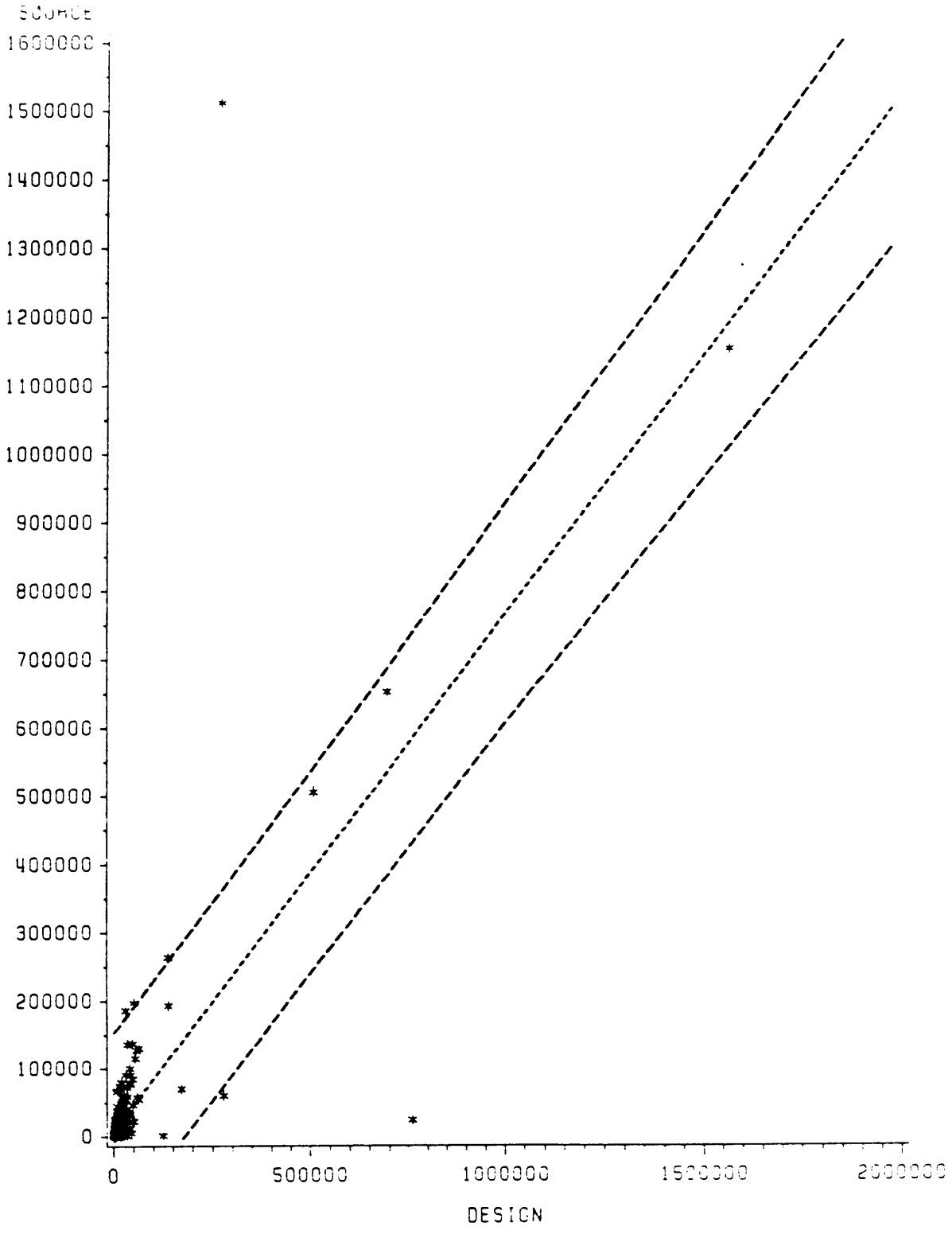

Figure 43. High Refinement Regression and 95% Confidence lines for E

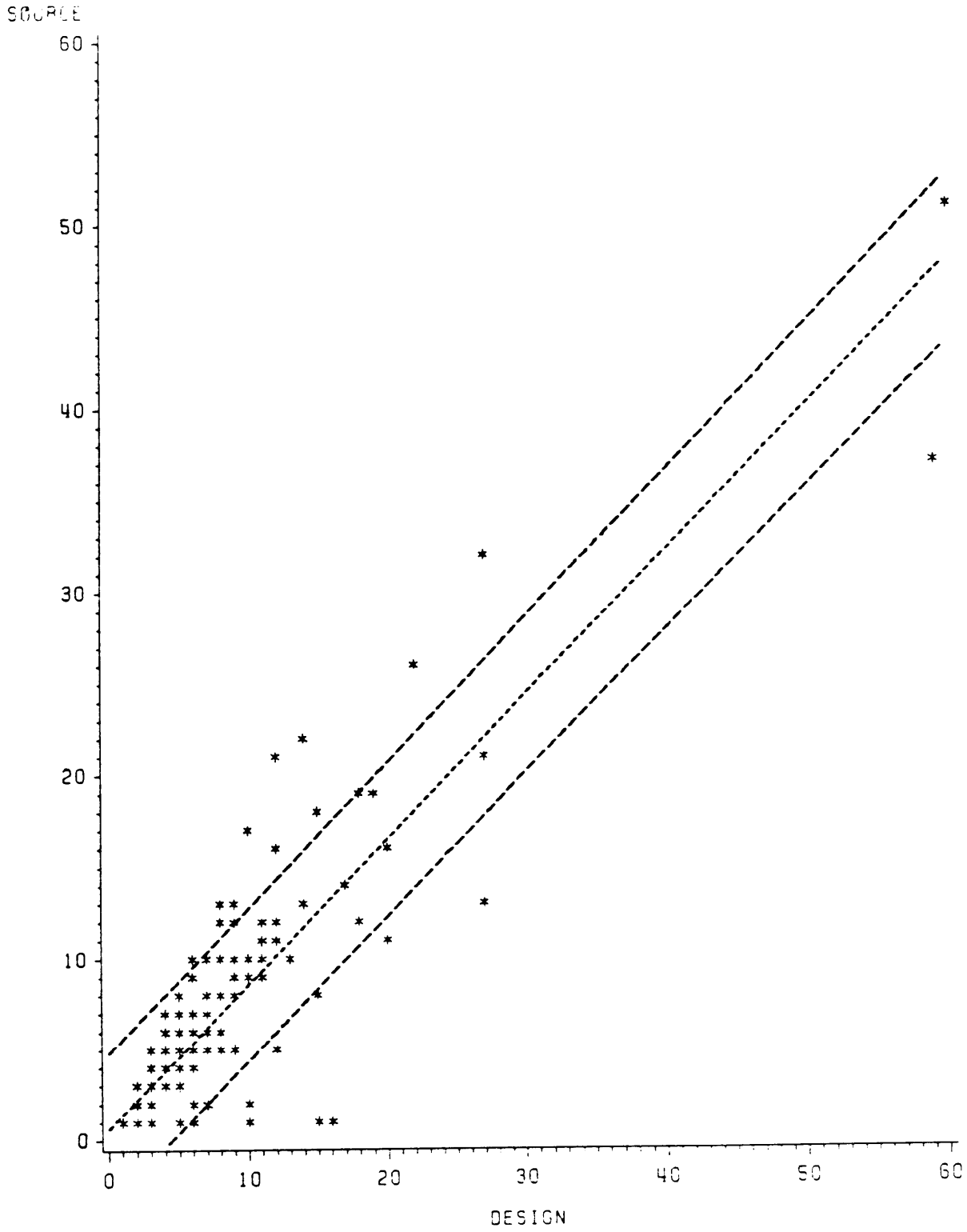

Figure 44. High Refinement Regression and 95% Confidence lines for CC

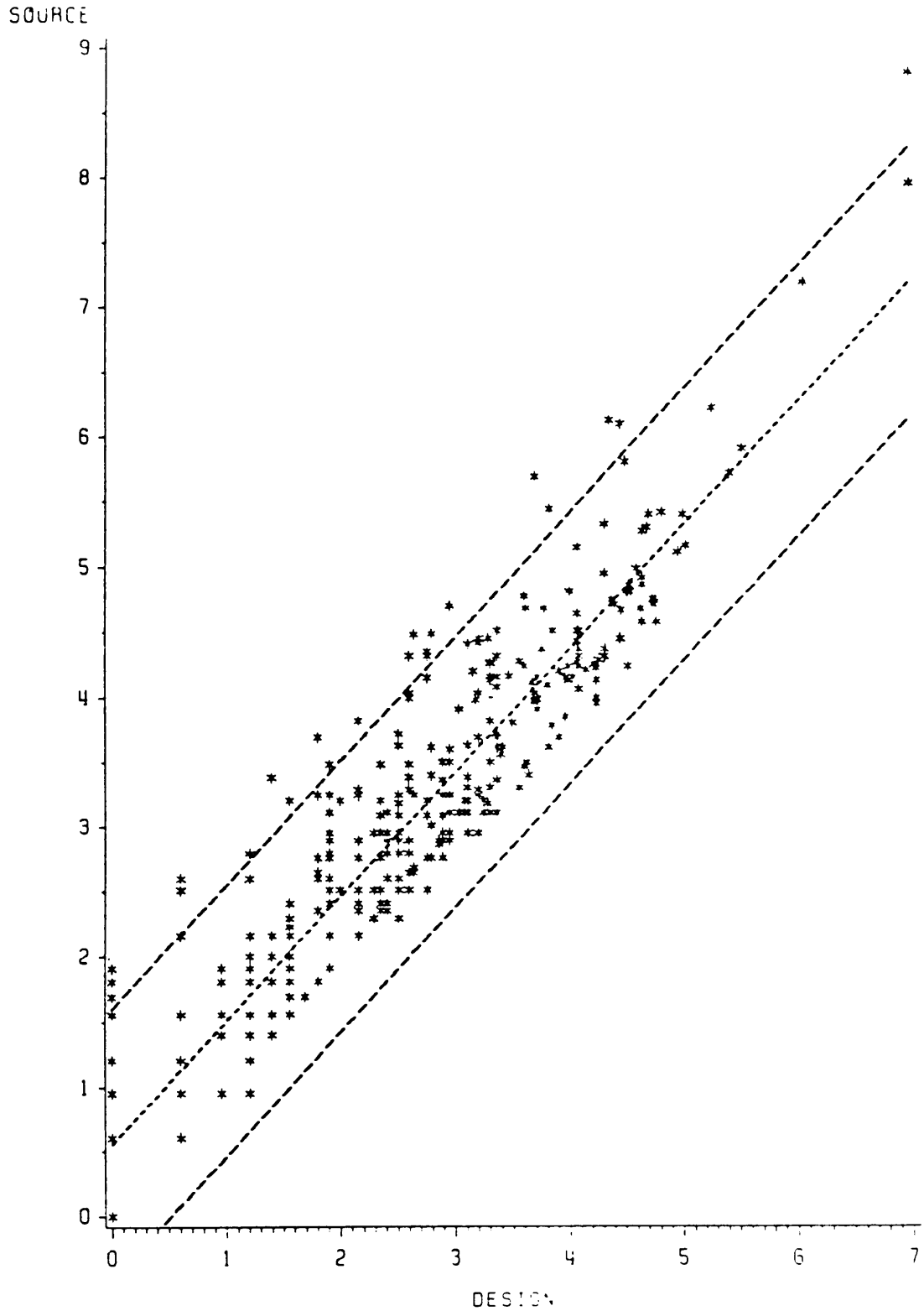

Figure 45. High Refinement Regression and 95% Confidence lines for INFO

The vita has been removed from the scanned document