

AD A104249

LEVEL II



RADC-TR-81-145
Final Technical Report
June 1981

ANALYSIS OF IV&V DATA

Logicon, Inc.

Jane W. Radatz

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
SEP 16 1981
S D

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

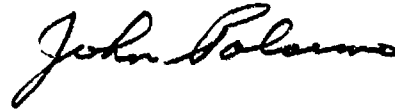
DTIC FILE COPY

81 9 16 015

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

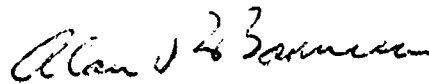
RADC-TR-81-145 has been reviewed and is approved for publication.

APPROVED:



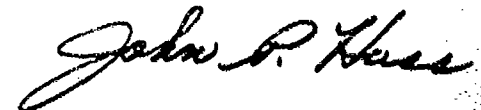
JOHN PALAIMO
Project Engineer

APPROVED:



ALAN R. BARNUM, Assistant Chief
Information Sciences Division

FOR THE COMMANDER:



JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER RADC-TR-81-145 ✓	2. GOVT ACCESSION NO. AD-A104 249	3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) ANALYSIS OF IV&V DATA ,		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. 1 Apr 80 - 31 Mar 81	
7. AUTHOR(s) Jane W. Radatz		6. PERFORMING ORG. REPORT NUMBER R:SED-81319 ✓	
		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0115/	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Logicon, Inc. 255 W. 5th Street San Pedro CA 90731		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 25280106	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		12. REPORT DATE June 1981	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		13. NUMBER OF PAGES 140	
		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
		16. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same			
18. SUPPLEMENTARY NOTES RADC Project Engineer: John Palaimo (RADC/ISIE)			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Independent Verification and Validation Development Productivity Software Reliability Software Error Analysis Software Maintainability Software Error Categories Development Cost Software Data Collection			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report presents the results of a one-year study of Independent Verifi- cation and Validation (IV&V) results. Five large IV&V projects are examined to determine the effects of IV&V on software reliability, main- tainability, and development cost/productivity. Current literature rele- vant to these topics is surveyed. IV&V-detected anomalies are categorized as to location, error category, effect, severity, acceptance, and resolu- tion. Differences in project results are used to formulate recommendations for improving IV&V effectiveness.			

DD FORM 1473 EDITION OF 1 NOV 68 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

4-3061

LB

TABLE OF CONTENTS

1.	Introduction	7
2.	IV&V Project Characteristics	9
2.1	Development Project Characteristics	9
2.2	IV&V Project Characteristics	11
3.	General Results	17
3.1	Number of Anomalies Found	17
3.2	Distribution of Anomalies Among Development Materials	20
3.3	Anomaly Categories	23
3.4	Anomaly Effects	25
3.5	Anomaly Severity	27
3.6	Phase of Anomaly Detection	27
3.7	Anomaly Report Acceptance	27
3.8	Anomaly Resolution	32
3.9	Data Relationships	32
4.	Results Concerning Software Reliability	37
4.1	Relevant Findings in the Literature	38
4.2	Project Results	40
5.	Results Concerning Software Maintainability	53
5.1	Software Attributes That Contribute to Maintainability	54
5.2	IV&V's Potential for Improving Maintainability	55
5.3	Project Results	58
6.	Results Concerning Development Cost/Productivity	69
6.1	Factors Affecting Development Cost/Productivity	70
6.2	Factors Affected by IV&V	70
6.3	Project Results	82
7.	Conclusions	99
7.1	Study Conclusions	99
7.2	Recommendations for IV&V Planning and Management	100
7.3	Recommendations for Future Study	101
7.4	Recommendations for Data Collection	103
	Appendix A - Project Selection	107
	Appendix B - Data Collection	113
	Appendix C - Software Features That Contribute to Maintainability	133
	References	137

LIST OF ILLUSTRATIONS

1. Anomalies Per Thousand Machine Instructions	19
2. Code Anomalies Per Thousand Machine Instructions	21
3. Development Materials in Which Anomalies Were Found	22
4. Predicted Effects of the Anomalies Reported	26
5. Severity Ratings of the Anomalies Reported	28
6. Severity Ratings of Code Anomalies	29
7. Anomalies Found in Each Development Phase	30
8. Anomaly Report Acceptance	31
9. Anomaly Resolution	33
10. Acceptance of Anomaly Reports Concerned With Reliability	41
11. Resolution of Anomalies Concerned With Reliability	42
12. Corrected Reliability Anomalies Per Thousand Machine Instructions	44
13. Development Materials in Which Corrected Reliability Anomalies Were Found	45
14. Number of Anomalies Affecting Each Aspect of Reliability	48
15. Severity Ratings of Corrected Reliability Anomalies	49
16. Development Phase in Which Corrected Reliability Anomalies Were Detected	50
17. Acceptance of Anomaly Reports Concerned With Maintainability	60
18. Acceptance of Anomaly Reports Concerned Solely With Maintain- ability	61
19. Resolution of Anomalies Concerned With Maintainability	62
20. Resolution of Anomalies Concerned Solely With Maintainability	63
21. Development Materials in Which Corrected Maintainability Anomalies Were Found	65
22. IV&V Cost	84

LIST OF TABLES

1. Selected Projects	10
2. Software Tools Used on the IV&V Projects	13
3. Data Collected From the IV&V Projects	18
4. Number of Anomalies Reported in Each Category	24
5. Anomaly Severity Relationships	34
6. Anomaly Resolution Relationships	35
7. Number of Corrected Reliability Anomalies in Each Category	46
8. Number of Corrected Maintainability Anomalies in Each Category	66
9. Factors Affecting Development Cost/Productivity	71
10. Cost Benefits of Early Detection--Scenario 1	88
11. Cost Benefits of Early Detection--Scenario 2	90
12. IV&V Effects on the Cost of Defect Removal	93
13. Cost Effects of Invalid Anomaly Reports	96
14. Summary of Anomaly Report Processing Effects	98

SUMMARY

The Analysis of IV&V Data Study was a one-year project undertaken to determine the effects of Independent Verification and Validation (IV&V) on software reliability, maintainability, development cost, and development productivity. Five large IV&V projects were selected for study. Information was collected from each of the five projects concerning the development effort, the IV&V effort, and each anomaly reported by IV&V. Current literature was surveyed to obtain information relevant to software reliability, maintainability, development cost, and development productivity. The IV&V results were analyzed in the light of findings from the literature, and conclusions were drawn and recommendations formulated for improving IV&V effectiveness.

The results of the study may be summarized as follows:

- General Results: 1575 anomalies were reported by the 5 projects. Of these, 1023 concerned software reliability, 854 concerned software maintainability, and 167 concerned efficiency, usability, and other effects. Multiple effects cause the sum to exceed 1575.
- Effects on Reliability: The primary concern of IV&V is software reliability.
 - Each IV&V project reported an average of 150 anomalies (2.2 per thousand machine language instructions) that would have affected program reliability and that were considered important enough to be acted on.
 - Of the three programs that have recorded operational performance, none has required modification to correct reliability problems after undergoing IV&V.
- Effects on Maintainability: IV&V effects on maintainability varied dramatically from one project to another, depending upon project objectives:
 - Where the IV&V charter included maintainability as a concern, as many as 197 anomalies concerned solely with maintainability were reported and corrected.
 - Where maintainability concerns were deemphasized, as few as three such anomalies were reported.
- Effects on Development Cost/Productivity:
 - IV&V cost averaged 25% of development cost and 20% of total acquisition cost on the projects surveyed.
 - Cost savings resulting from the detection of reliability anomalies alone ranged from 5% to 25% of development cost, in some cases exceeding the cost of the IV&V effort.

- The detection of maintainability and other anomalies had additional cost benefits on the software life cycle.
- Out of 125 cost/productivity factors identified from the literature, IV&V has no effect at all on 90, indicating the limited overhead that IV&V places on the development process.
- Of the remaining 35 factors, IV&V has a positive effect on 27 and a negative effect on 9 (on one, both a positive and negative effect could be seen).
- IV&V enhances programmer productivity by decreasing the time spent in false starts and defect removal.

The major conclusions of the study were as follows:

- IV&V results depend on project charter and directives--IV&V finds the types of problems it is directed to find.
- IV&V has a significant effect on software reliability.
- IV&V is being underutilized as a tool for improving software maintainability.
- IV&V can pay for itself through the detection of reliability anomalies alone.
- The cost benefits of IV&V are enhanced by early detection of anomalies.

Major recommendations derived from these conclusions are as follows:

- To increase IV&V's effect on reliability: encourage independence of outlook and techniques, allow for early detection of problems, and ensure that anomaly corrections are reverified.
- To increase IV&V's effect on maintainability: include a maintainability evaluation in the IV&V project's charter and allow time in the development process for verification of final program documentation.
- To increase IV&V cost benefits: begin IV&V early in the development process, require delivery of preliminary development materials, especially those for requirements and design, and ensure prompt action on IV&V findings.

PREFACE

This document is the technical report for the Analysis of IV&V Data Study performed by Logicon, Inc., under Contract F30602-80-C-0115 with the Rome Air Development Center (RADC). The work was performed during the period 1 April 1980 to 31 March 1981. The author of this report is Jane W. Radatz. Technical direction was provided by Mr. John Palaimo, the RADC project engineer. Special thanks are owed to Mr. Donald Fletcher (WSMC/RSCS), Captain John Grelck (BMO/MNNAG), and Lt. Colonel Thomas Jarrell (ASD/YYM), who provided data essential to the study. Significant contributions to the data collection activity were also made by Elaine Renner, Norie Roeder, and Joan Small of Logicon. Susan Moy and Myra Chern provided expertise in the statistical analysis of the IV&V data. Marilyn Fujii, Edward Hinton, Jeffrey Laub, and Dennis Meronek reviewed this report and provided valuable comments and suggestions.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

DTIC
ELECTE
SEP 16 1981
D

1. INTRODUCTION

Independent Verification and Validation (IV&V) is the systematic evaluation of a computer program by an agency independent of the developer. Usually performed in parallel with the software development effort, IV&V has as its major objectives detecting development problems as early as possible and providing the program office with increased visibility into the development effort. The basic tenet of this approach is that the independence of the IV&V agency provides a fresh viewpoint, an objective attitude, and tools and techniques specifically designed for error detection.

Current trends in large-scale Department of Defense (DoD) software procurement are toward increasing use of IV&V as a managerial aid. Air Force Regulation 122-9 requires IV&V for all software that exercises direct control over nuclear weapons. The Navy's software life cycle management guide, NAVELEXINST 5200.23, "strongly recommends" full IV&V for all projects in certain categories and requires program managers to "keep in mind the advantages of having an IV&V contractor" when planning a software development project. Additional Air Force policy statements are expected in the near future making IV&V a part of the development of all embedded computer systems. This increasing use of IV&V has resulted from growing concern for the quality of computer software and from recognition of the serious impact of software errors on critical and costly systems.

On most projects, use of IV&V is at the discretion of the program manager. Because IV&V can be a significant cost factor in a software development project, the decisions of whether and how to use it are major ones. Until now, no quantitative data have been available to help a program manager make these decisions or to provide baselines against which IV&V results could be measured. The Analysis of IV&V Data Study was undertaken to provide this information.

The objectives of the study were as follows:

- To determine the effects of IV&V on software reliability, maintainability, development cost, and development productivity
- To formulate recommendations for improving the effectiveness of IV&V on future projects

The approach that was taken was to obtain the results of actual IV&V projects, to identify significant similarities and differences among them, and to correlate the findings with results found in the literature. Section 2 describes the projects that were used for the study. Sections 3 through 6 present results of the data analysis. Conclusions and recommendations are presented in Section 7. Details of the technical approach are given in Appendixes A and B. Appendix C identifies software features that contribute to maintainability.

2. IV&V PROJECT CHARACTERISTICS

The study examined five large IV&V projects. These projects were selected from 35 candidate projects based on criteria including the size and application of the system being evaluated and the availability of data for the IV&V study. Table 1 identifies the projects that were selected. The following paragraphs describe these projects in terms of the development projects undergoing IV&V and the characteristics of the IV&V projects.

2.1 Development Project Characteristics

The development efforts evaluated by the five IV&V projects involved a variety of scientific applications and employed different development techniques. Their basic characteristics are summarized below.

2.1.1 Project 1

Project 1 evaluated three interrelated programs:

- Two command and control programs
- A mathematical program invoked by the command and control programs

The command and control programs each consisted of 24,000 assembly language source lines and were real-time, interrupt-driven programs. One was being modified; the other was undergoing initial development. The mathematical program consisted of 39,000 FORTRAN source lines and 2,000 assembly language source lines. It was a time-critical batch program (that is, it had to finish executing within a given amount of time) and was undergoing initial development.

Development of this system took place over a 3-1/2-year period. Programming practices included top-down program design, use of a basic program support library, and use of a modified programmer team.

2.1.2 Project 2

Project 2 evaluated a major modification to the three programs involved in Project 1. Program sizes, real-time characteristics, and programming practices remained virtually the same.

2.1.3 Project 3

Project 3 evaluated critical portions of a missile tracking and analysis system. The overall system contained 176,000 source lines. Included in the study were IV&V efforts focusing on two key programs:

- A dedicated operating system
- A missile tracking program

Table 1. Selected Projects

<u>Project Number</u>	<u>Description</u>	<u>Source Lines</u>	<u>Language Type</u>	<u>RT/ NRT</u>	<u>Modern S/W Eng</u>	<u>New/ Modification</u>
1	IV&V of a command and control system	93K	HDL/Asm	Both	No	Both
2	IV&V of a command and control system modification	93K	HDL/Asm	Both	No	Major modification
3	IV&V of a missile tracking and analysis system	175K	HDL/Asm	Both	No	New
4	IV&V of a real-time display system	90K	HDL/Asm	Both	Yes	Both
5	IV&V of an avionics software system	158K	HDL/Asm	RT	No	New

The operating system was a real-time program consisting of 14,000 assembly language source lines. The missile tracking program was a real-time program consisting of 23,000 FORTRAN source lines and 16,000 assembly language source lines.

Development of the system took place over a 4-year period. Programming practices included the use of both full and modified programmer teams.

2.1.4 Project 4

Project 4 evaluated a real-time display system. Included in the study were IV&V efforts focusing on programs totaling 40,000 FORTRAN source lines and 1,000 assembly language source lines. The system included both real-time and nonreal-time components and was developed over a period of 2-1/2 years.

The Project 4 development was the only one of the five that was considered to have used modern programming practices. Features of the development effort included:

- Top-down design at both the system and program levels
- Use of a program design language
- Weekly walk-throughs from program inception
- Use of a modified programmer team
- Use of structured FORTRAN implemented with a preprocessor
- Development in "builds"
- Use of a fully automated program support library

2.1.5 Project 5

Project 5 evaluated portions of an avionics system totaling 158,000 lines of JOVIAL and assembly language code. Included in the study were IV&V efforts focusing on a real-time portion consisting of 52,000 assembly language source lines.

Development of the program took place over a 6-year period. Programming practices included top-down design at both the system and program levels and the use of a modified programmer team. A unique feature of the Project 5 development effort was a change in charter that occurred during the coding and check-out phase. Rather than completing the initial development as originally planned, the developer was redirected to explore alternative implementations as part of the development effort.

2.2 IV&V Project Characteristics

The standard approach to IV&V entails five distinct activities proceeding in parallel with the development effort:

- Requirements Verification: Evaluation of the program requirements, as documented in requirement specifications, to ensure that they are clear, complete, correct, and consistent with one another and with higher level specifications

- Design Verification: Evaluation of the preliminary and detailed design, as documented in the before-code design specification, to ensure that it is a complete and correct implementation of the verified requirements
- Code Verification: Inspection of the coded version of the program to ensure that it is a complete, correct, and (sometimes) optimal implementation of the verified design
- Program Validation/Testing: Formal testing of the program to ensure that it satisfies its specified requirements
- Documentation Verification: Inspection of the requirement and design specifications, and sometimes the user manuals and other documents, to ensure that they accurately describe the program as implemented

There was considerable deviation from this process among the projects surveyed. The IV&V efforts comprising Projects 1 and 2 focused primarily on code verification and testing. Requirement specifications were for the most part accepted as valid and used as the baseline against which the code was evaluated. Design materials were not published until well after code production was under way, so were not available for design verification. Analysis of the requirement specifications, design specification, and other documents for adequacy as program documentation was not within the scope of the projects.

On Project 3, the requirement specifications contained a significant amount of design material. As a result, requirements verification detected both requirement and design problems. The before-code design specification contained only high-level design information, preventing a detailed design verification activity using this document. Code verification and testing took place as usual, and considerable attention was devoted to documentation verification.

Project 4 addressed the development effort that used modern programming practices. On this project, IV&V participants took part in the weekly walk-throughs of the evolving requirements and design. Many problems were reported in these meetings rather than through the normal medium of anomaly reports. Code verification and testing were performed as usual. Documentation verification was deemphasized, and most documentation problems were reported by letter rather than by anomaly report.

Project 5 was the only one of the five to include a full design verification step. This project was also nonstandard, however, in that requirements verification was not performed and in that, with the redirection of the development effort to explore alternative implementations, the IV&V effort conducted extensive testing in support of this new effort.

Table 2 identifies the software tools used on the five IV&V projects. All five projects used both static analysis tools, which process or evaluate the program without executing it, and dynamic analysis tools, which aid in program testing by providing a test environment, controlling and monitoring program execution, or modeling program behavior.

Table 2. Software Tools Used on the IV&V Projects

Tool	Function
<u>Projects 1, 2</u>	
Tape Comparison Program	Identify differences between object tapes
Source Comparison Program	Identify differences between source files
Code Inspection Aid	Generate global cross-references and annotated source listings; identify certain errors
Software Environment Simulator	Simulation of flight computer, peripheral devices, and external environment
Data Base Analyzer	Generate set/use information from the global cross-reference
Memory Decode Program	Translate load module to source language
Extension Register Analyzer	Verify correct setting of extension registers
Real-Time Analyzer	Detect potential conflicts caused by interrupts and job priorities
Source Conversion Program	Convert source code to execute on different computer
Assembler, Compiler, Loader	Verify correct assembly, compilation, loading
Memory Allocation Program	Provide variable name and initial value of each location in temporary memory
Drum Memory Dump Processor	Format and print selected portions of drum memory
Program Structure Analyzer	Confirm correct compilation of program
Interpretive Computer Simulation (ICS) of Flight Computer	Verify correct operation of flight program in its target computer
HOL Simulation of Flight Program	Evaluate flight program accuracy and correctness
ICS of Ground Program Computer	Verify correct operation of C ³ I program

Table 2. Software Tools Used on the IV&V Projects (continued)

Tool	Function
Data Base Preprocessor	Format and verify flight constants prior to simulation
Branch Analysis Program	Monitor program execution by recording the number of times each code segment is executed and each branch outcome occurs
<u>Project 3</u>	
Global Cross-Reference Generator	Generate cross-reference of program variables and labels
Source Comparison Program	Identify differences between source files
Interrupt Interceptor	Intercept and modify real-time interrupts
Software Monitor	Record the state of the system at pre-selected execution points
Console Monitor	Capture transient displays on CRT console and produce hard copy for later analysis
Time Mark Tool	Monitor and record all program requests made to timer; simulate additional requests
Real-Time Driver	Create real-time tasks and monitor task dispatching
Console Test Program	Send selected discretes to flight control console to test hardware interface
Macro Test Program	Verify correct operation of program macros
Data Capture Program	Write incoming telemetry data on tape
Plot Board Driver	Send data to plotboards to test hardware interface
Task Network Simulator	Simulate a network of real-time tasks to test task communication and queueing
Test Driver	Drive operating system and monitor behavior
Automated Flowcharter	Generate flowcharts from code

Table 2. Software Tools Used on the IV&V Projects (continued)

Tool	Function
Program Structure Analyzer	Perform symbolic execution and path analysis
Time Code Translator Test Program	Test hardware clock
Branch Analysis Program	Monitor program execution by recording the number of times each code segment is executed and each branch outcome occurs
Execution Tracer	Execute in conjunction with instrumented code to trace execution, display intermediate values, and verify assertions
Data Base Modifier	Modify data base values for program testing
Data Base Access Program	Permit batch access to data base for testing of individual routines
Sensor Data Extractor	Extract sensor data from history tapes for comparison and for use in driving plotboards
Routine Interface Verifier	Verify consistency and correctness of routine interfaces
Microfiche Plot Program	Generate microfiche plots of sensor data
History Tape Converter	Convert history tapes generated by one system to format suitable for testing another system
History Tape Perturber	Modify contents of history tape for program testing
Data Flow Analyzer	Perform analysis of interprocedural data flow to detect potential mishandling of data
Software Timer	Monitor execution time of selected modules
Address Locator	Determine address of any routine in program load module

Table 2. Software Tools Used on the IV&V Projects (continued)

Tool	Function
<u>Project 4</u>	
Source Comparison Program	Identify differences between source files
Global Cross-Reference Generator	Generate cross-reference of program variables and labels
Automated Flowcharter	Generate flowcharts from code
Program Structure Analyzer	Perform symbolic execution and path analysis
File Index and Source List Generator	Generate indexed source listing
History Tape Dump	Format and print contents of program history tape
History Tape Modifier	Modify history tape for program testing
Load Module Comparison Program	Ensure that the load module tested by IV&V matches the load module certified for operational use
<u>Project 5</u>	
Environmental Simulator	Simulate electromagnetic pulse environment
Electronic Warfare Simulator	Simulate flight computer and interfacing equipment
Core Image Comparator	Compare core image before and after simulation to detect memory destruction
Core Image Reformatter	Reformat core image for use on another computer
Computer Interface Program	Permit interfacing of two computers
Patch Processor	Apply patches to a given core image
HOL Simulation of Program	Evaluate algorithms used in program

3. GENERAL RESULTS

Table 3 identifies the types of data collected from the five IV&V projects. Analysis of this data was performed with the aid of the Statistical Package for the Social Sciences (SPSS), a collection of statistical programs developed by the University of Chicago (Reference 1).^{*} Results specific to IV&V's effects on software reliability, maintainability, and cost/productivity are presented in Sections 4, 5, and 6, respectively. General results useful as background for these findings are presented in this section.

3.1 Number of Anomalies Found

IV&V results are of two general types:

- The detection of anomalies
- Assurance of the absence of anomalies of particular types in given segments of code or documentation

Both types of results contribute to the determination of software quality. The first type, however, not only determines software quality but affects it as well by bringing about the correction of software faults. To evaluate IV&V's effect on software reliability and maintainability, therefore, the study focused on the anomaly detection aspect of IV&V, looking at the number and types of anomalies detected and the impact and resolution of these anomalies.

A total of 1575 anomalies were reported by the five IV&V projects. The number reported by each project was as follows:

- Project 1: 249
- Project 2: 325
- Project 3: 510
- Project 4: 175
- Project 5: 316

To normalize these figures, they were compared with the number of machine language instructions generated by the programs examined. The results are shown in Figure 1. The overall average was 4.6 anomalies per thousand machine language instructions. Project 3 varied most dramatically from this average, with 13.4 anomalies per thousand machine instructions.

Since different projects operated under different charters regarding analysis of specifications and other documents, a similar analysis was performed using only the code anomalies found. A total of 802 code anomalies were reported, broken down as follows:

^{*}N.T. N. H., et al., Statistical Package for the Social Sciences, McGraw Hill, 1975.

Table 3. Data Collected From the IV&V Projects

- Data concerning each anomaly reported by IV&V
 - Location (specification, code, etc.)
 - Type of problem
 - Probable effects if left uncorrected
 - Severity
 - Detection date
 - Detection method
 - Resolution
 - Resolution date
- Data concerning each IV&V project
 - Objectives
 - Schedule
 - Man-loading
 - Relationship with developer
 - Tools and techniques used
 - Cost
- Data concerning each development project
 - Schedule
 - Man-loading
 - Development practices used
 - Test results
 - Software operational performance
 - Software maintenance requirements
 - Cost

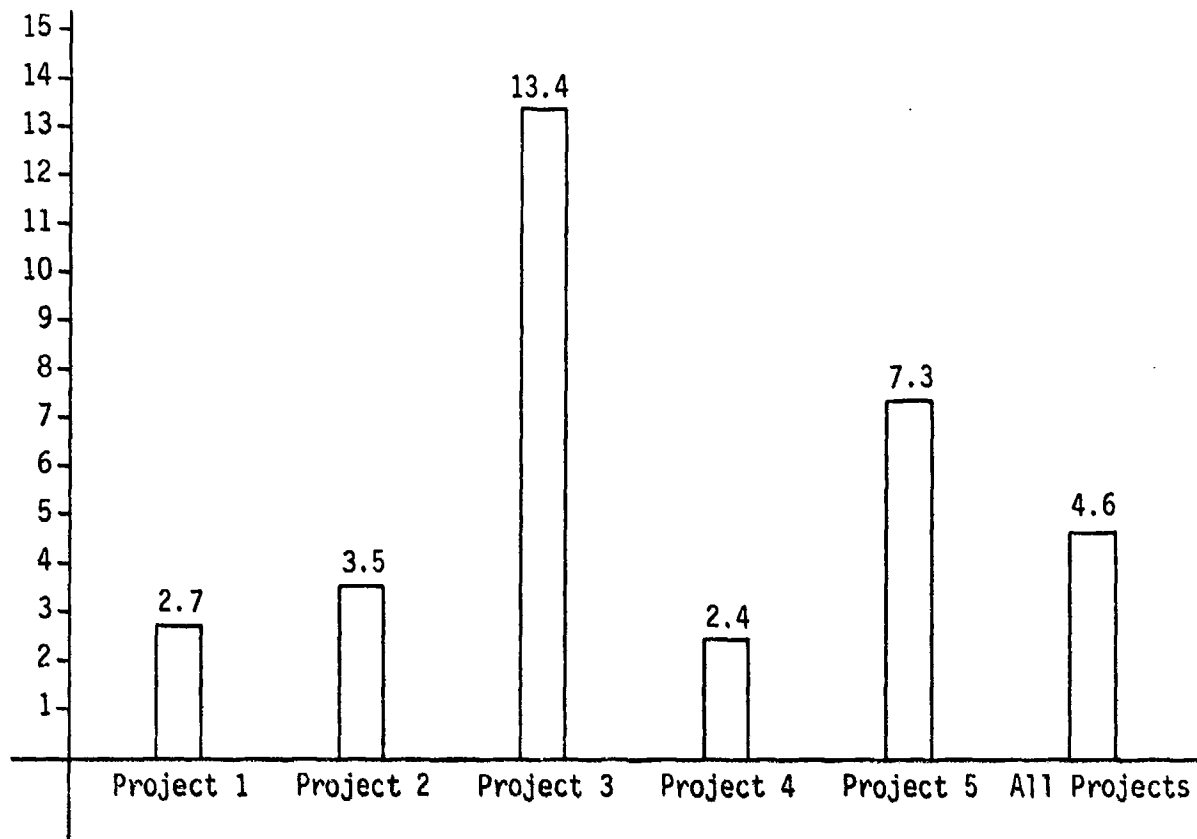


Figure 1. Anomalies Per Thousand Machine Instructions

- Project 1: 193
- Project 2: 205
- Project 3: 111
- Project 4: 100
- Project 5: 143

A comparison of these figures with the number of machine language instructions in each program is shown in Figure 2. These results are surprisingly uniform, averaging 2.2 code anomalies per thousand machine instructions. The somewhat higher figures for Project 5 probably result from the experimental nature of the development effort, with many different versions of the program being tried over the course of the development. It is interesting to speculate that the low figure shown for Project 4 results from its use of modern programming practices, but the sample size is too small to support such a conclusion. The overall average of 2.2 code anomalies per thousand machine instructions is slightly higher than the 2.0 figure observed by Rubey in a study of IV&V results performed in 1975 (Reference 2).*

3.2 Distribution of Anomalies Among Development Materials

Figure 3 indicates, for each project, the number of anomalies found in:

- Requirement specifications before code delivery
- Design specifications before code delivery
- Code
- Requirement specifications after code delivery
- Design specifications after code delivery
- User documentation and other materials

The analysis was intended to determine where and when development problems were likely to be found by IV&V. Instead, it illustrates the high dependence of IV&V results on IV&V and development project characteristics.

The results for Projects 1 and 2 clearly reflect their focus on ensuring compliance of the code with the requirements. The only surprising aspect is the significant number of requirement anomalies reported on Project 2. The results for Project 3 reflect its attention to all aspects of IV&V, the unavailability of detailed design materials for a full design verification, and its strong emphasis on documentation verification. Results for Project 4 reflect the reporting of requirement, design, and documentation anomalies in meetings and letters rather than anomaly reports. Project 5 results show its de-emphasis on requirement verification and its performance of design verification, code verification, testing, and documentation verification. The change of charter imposed on the Project 5 development effort, making it an experimental rather than standard development project, makes it impossible to determine whether the performance of design verification would have decreased the number of anomalies found in the code.

*Rubey, R. J., "Quantitative Aspects of Software Validation," Proceedings of the International Conference on Reliable Software, April 1975, pp. 246-251.

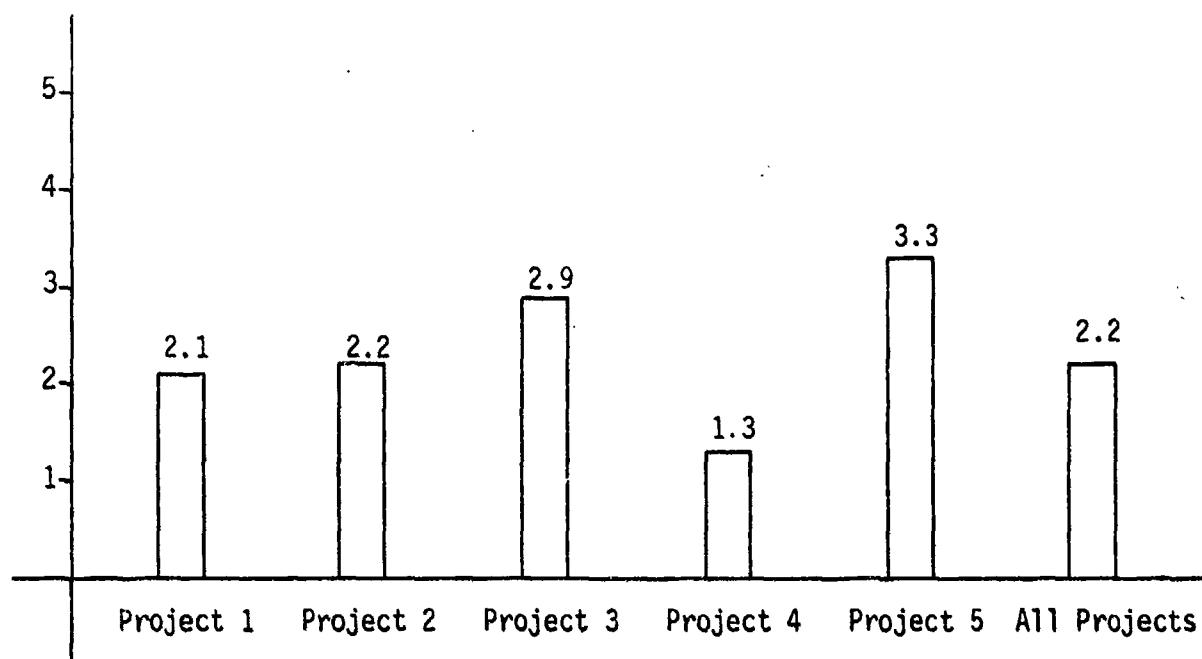


Figure 2. Code Anomalies Per Thousand Machine Instructions

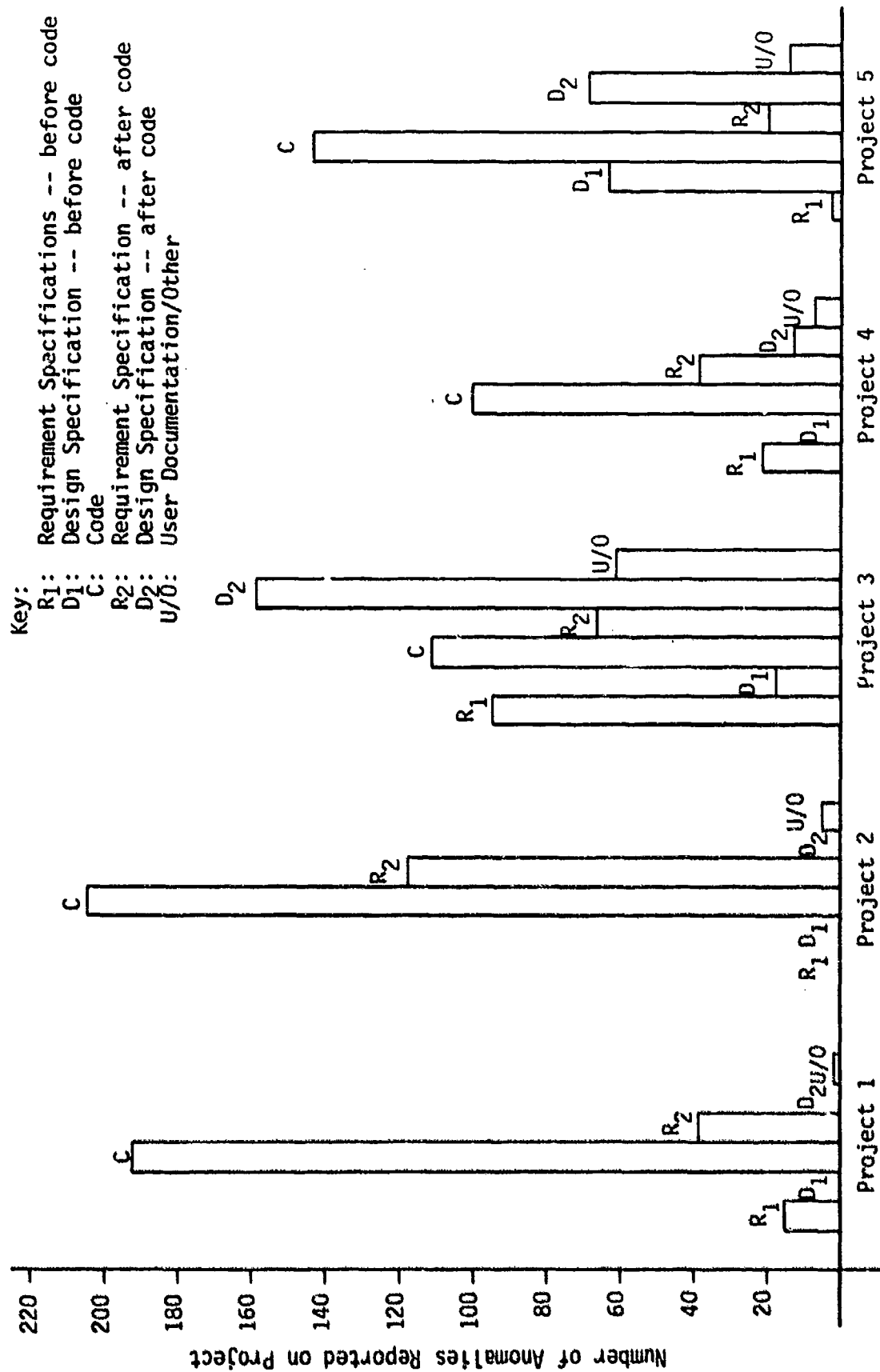


Figure 3. Development Materials In Which Anomalies Were Found

Of all five projects, only Project 3 reported a significant number of anomalies in the user documentation. The other projects were not chartered to evaluate these materials, and reported only those anomalies they happened to detect in trying to use the documents themselves.

3.3 Anomaly Categories

Table 4 shows the number of anomalies reported in each category for each project and for the study as a whole. Notable results are as follows:

- Among requirement anomalies, incorrect and incomplete requirements predominate, accounting for 72% of all requirement anomalies.
- Among before-code design specification anomalies, reported almost solely by Project 5, anomalies concerned with choice of algorithm/mathematics, data definition, and data handling predominate.
- Among code anomalies:
 - Projects 1, 2, and 5 had as their most prevalent category "choice of algorithm or mathematics," a design-oriented category.
 - All projects reported a considerable number of data handling problems.
 - Overall results show that code anomalies fell into the following categories, in decreasing order of frequency:
 - o Choice of algorithm or mathematics (30%)
 - o Data handling (24%)
 - o Interfaces, I/O (11%)
 - o Requirement/design compliance; data definition (each 7%)
 - o Other code problems (6%)
 - o Sequence of operations (6%)
 - o Timing, interruptibility (5%)
 - o Presentation, standards compliance (4%)
- For documentation anomalies in the after-code design specification, prevalent categories were incorrectness, incompleteness, and, for Project 3, presentation and standards compliance.

Correlation of these results with development project characteristics led to the following observations.

3.3.1 New Development vs. Modification

The Project 2 development represented a modification to the Project 1 software. The considerably higher number of requirement anomalies for Project 2 may reflect a less rigorous requirements definition activity on the

Table 4. Number of Anomalies Reported in Each Category

Anomaly Category	Project					
	1	2	3	4	5	All
Requirement Specification Anomalies						
R1. Incorrect Requirements	15	55	73	16	4	163
R2. Inconsistent Requirements	9	13	16	7	17	62
R3. Incomplete Requirements	22	27	54	29	6	138
R4. Other Requirement Problems	7	18	15	3	--	43
R5. Presentation; Standards Compliance	2	3	4	1	--	10
Total	55	116	162	56	27	416
Before-Code Design Specification Anomalies						
D1. Requirement Compliance	--	--	10	--	1	11
D2. Choice of Algorithm, Mathematics	--	--	5	--	11	16
D3. Sequence of Operations	--	--	--	--	7	7
D4. Data Definition	--	--	--	--	19	19
D5. Data Handling	--	--	--	--	18	18
D6. Timing, Interruptibility	--	--	--	--	0	0
D7. Interfaces, I/O	--	--	--	--	8	8
D8. Other Design Problems	--	--	1	--	0	1
D9. Presentation; Standards Compliance	--	--	1	--	0	1
Total	0	0	17	0	64	81
Code Anomalies						
C1. Requirement, Design Compliance	13	6	26	9	2	56
C2. Choice of Algorithm, Mathematics	79	69	11	17	46	222
C3. Sequence of Operations	8	4	12	3	18	45
C4. Data Definition	6	22	1	21	6	56
C5. Data Handling	34	62	25	24	32	177
C6. Timing, Interruptibility	24	9	--	--	6	39
C7. Interfaces, I/O	23	21	9	16	10	79
C8. Other Code Problems	3	6	21	3	13	46
C9. Presentation; Standards Compliance	3	6	6	7	10	32
Total	193	205	111	100	143	752
After-Code Design Specification Anomalies						
P1. Incorrect Documentation	--	--	29	4	52	85
P2. Inconsistent Documentation	--	--	6	1	2	9
P3. Incomplete Documentation	--	--	57	4	10	71
P4. Other Documentation Problems	--	--	11	3	1	15
P5. Presentation, Standards Compliance	--	--	56	--	4	60
Total	0	0	159	12	69	240
User Documentation Anomalies	0	2	59	5	0	67
Other Anomalies	1	2	2	2	13	19
Anomalies in All Categories	249	325	510	175	316	1575

modification activity than on the initial development. The two projects had approximately the same number of code anomalies, with Project 2 having considerably more anomalies in the categories of data definition and data handling, and considerably fewer in requirement/design compliance and timing/interruptibility.

3.3.2 Language Type

Project 5 addressed a program written entirely in assembly language. It is interesting to note that this project reported the most anomalies of any project in category C3: "Sequence of operations," an area notoriously more difficult in assembly language than in higher order language. No other language-related results were observed.

3.3.3 Modern Programming Practices

The system evaluated by Project 4 was developed using modern programming practices. A possible correlation is that Project 4 reported the fewest anomalies of any project in category C3: "Sequence of operations." This result may be attributable to the use of program design language and structured programming. No other trends were observed that could be attributed to modern programming practices.

3.4 Anomaly Effects

Figure 4 indicates the number of anomalies on each project that had the potential to affect software reliability, maintainability, efficiency, and usability. The numbers may exceed project totals because of the potential for multiple effects.

As with anomaly location, discussed in Section 3.2, anomaly effects reflect each project's charter and objectives. Projects 1 and 2 were concerned almost solely with software reliability. As a result, over 90% of the anomalies reported on these projects affected reliability. The other projects were concerned not only with reliability, but with maintainability, efficiency, and usability as well. Their results present a more balanced picture.

Project 3 was unique in reporting considerably more maintainability than reliability anomalies. This was the result of its emphasis on documentation analysis. The number of reliability and maintainability anomalies for Project 4 were almost the same. Project 5 reported slightly more reliability than maintainability anomalies.

Overall results show that 65% of all anomalies affected reliability, 54% affected maintainability, 4% affected efficiency, and 6% affected usability. On all projects, anomalies affecting efficiency and usability accounted for only a small percentage of the total number of anomalies. Sections 4 and 5, which focus on reliability and maintainability anomalies, therefore discuss nearly all of the anomalies reported.

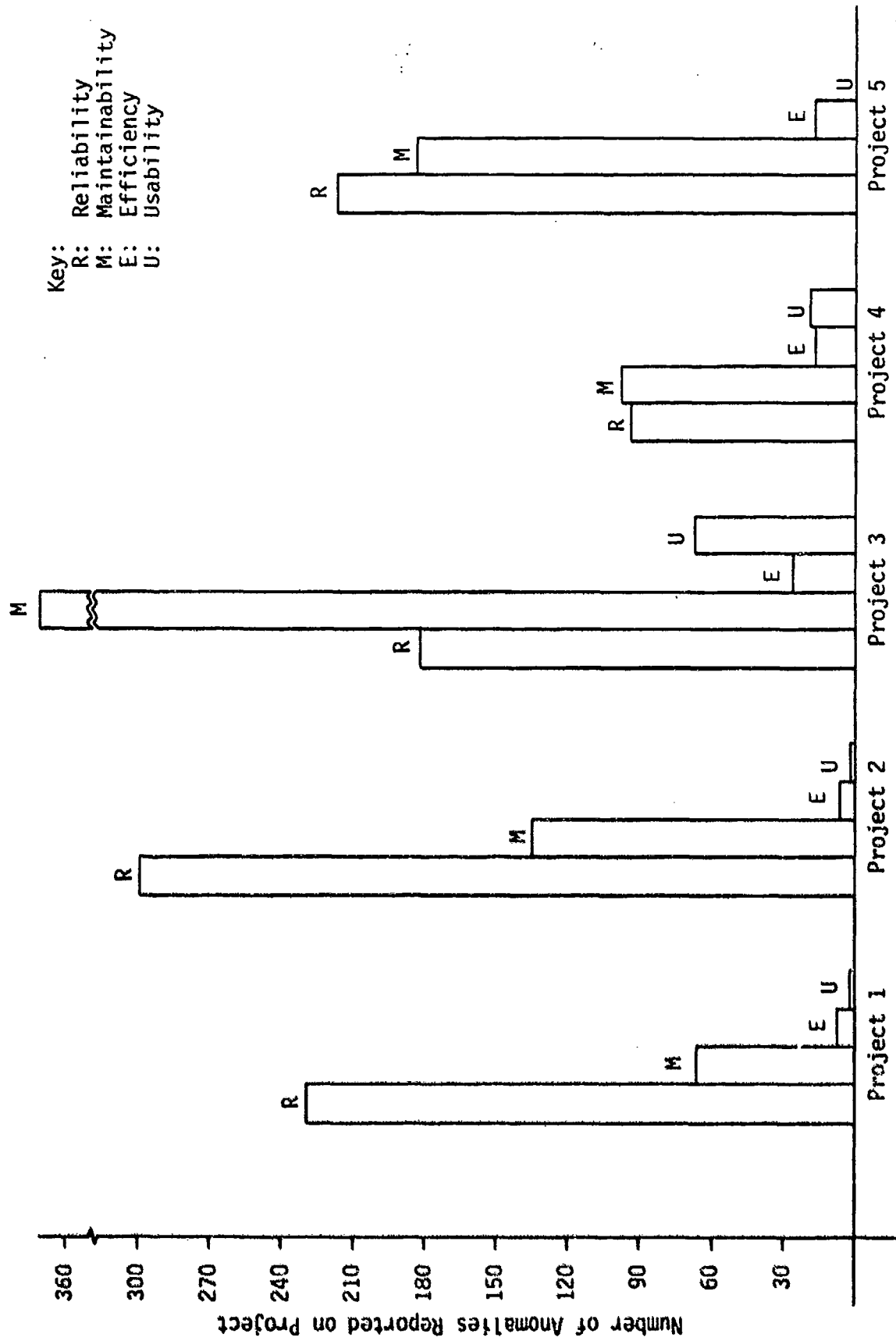


Figure 4. Predicted Effects of the Anomalies Reported

3.5 Anomaly Severity

Figure 5 indicates the number of anomalies on each project that had severity ratings High, Medium, Low, and Unknown. The overall results for the five projects show that approximately a tenth of the anomalies received High ratings, a fourth were rated Medium, and two thirds were rated Low. While the precise meanings of these ratings vary from one project to another, they are generally considered to have the following interpretation for the types of programs considered here:

- High: Threat to life or property
- Medium: Serious threat to mission objectives
- Low: Degraded system performance or non-operational effect

The seriousness of these consequences indicates the significance of the 110 High- and 404 Medium-severity anomalies reported.

There is a connection between the type of anomalies reported on a project and the severity rating results. Projects 3, 4, and 5, which reported a significant percentage of maintainability anomalies, show higher percentages of Low ratings than Projects 1 and 2, which concentrated on reliability problems. To arrive at a more accurate comparison, the same analysis was performed using only the anomalies concerned with program code. Figure 6 shows the results. The overall figures show that over a tenth of all code anomalies received High ratings, 41% received Medium ratings, and just over half received Low ratings. Again, the results varied significantly from one project to another. Extremes were Project 1, which reported 21% High, 48% Medium, and 30% Low, and Project 4, with 1% High, 12% Medium, and 87% Low. The other projects had severity ratings closer to the overall average.

3.6 Phase of Anomaly Detection

Figure 7 shows the number of anomalies detected during each development phase. The results indicate the degree to which IV&V contributed to early detection of anomalies. Over half of the anomalies were reported before the developer's testing phase. Project 4 had the most dramatic results, with 89% of all anomalies reported before development testing. Project 5's results are also impressive, with 78% of all anomalies detected before the testing phase. Sections 5 and 6 discuss the benefits of early detection.

3.7 Anomaly Report Acceptance

Figure 8 shows the percentage of anomaly reports on each project that:

- Were accepted by the program office as valid
- Were accepted with changes
- Were rejected by the program office
- Were withdrawn by the IV&V contractor or superseded by other reports
- Had unknown acceptance

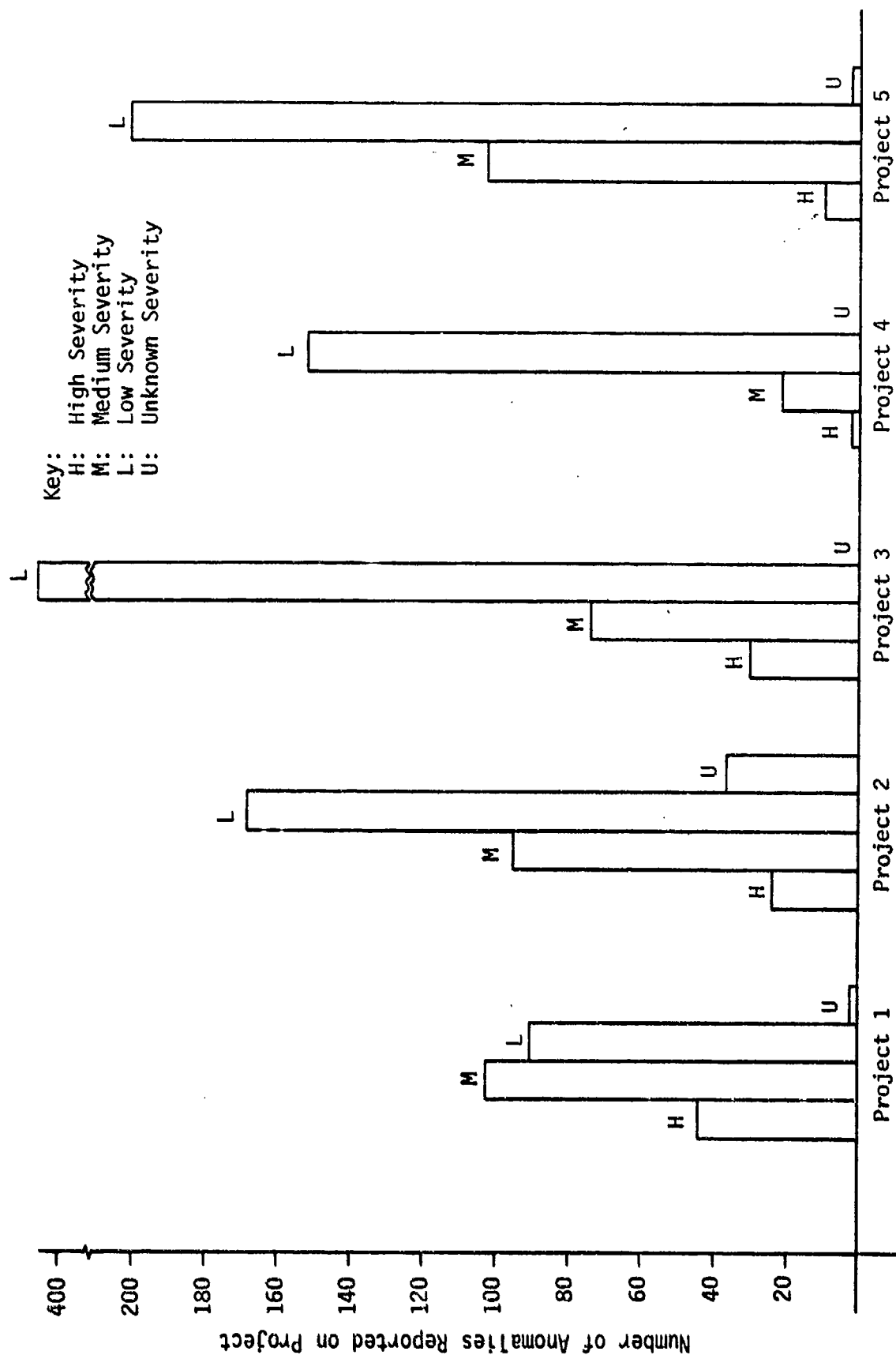


Figure 5. Severity Ratings of the Anomalies Reported

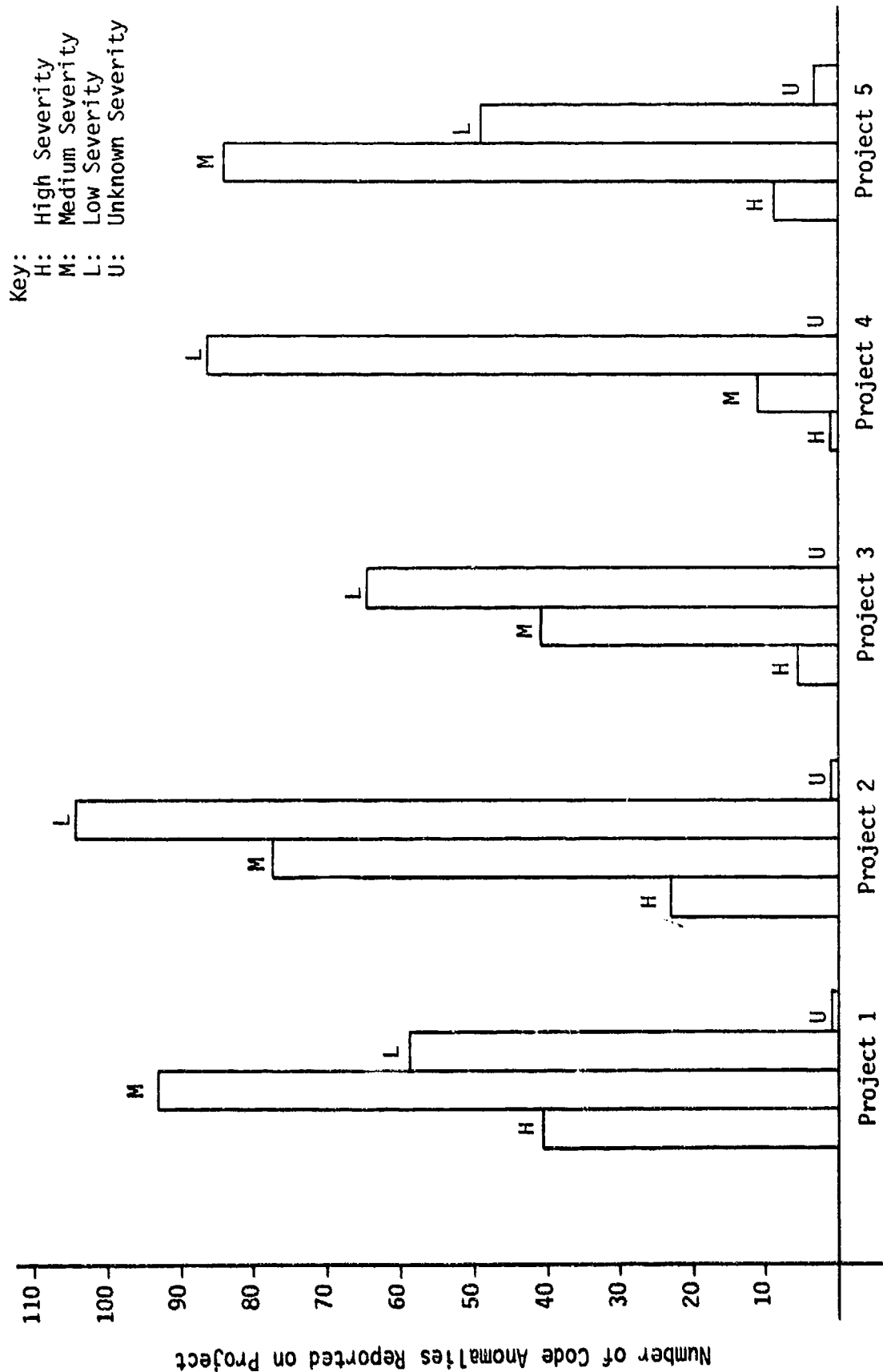


Figure 6. Severity Ratings of Code Anomalies

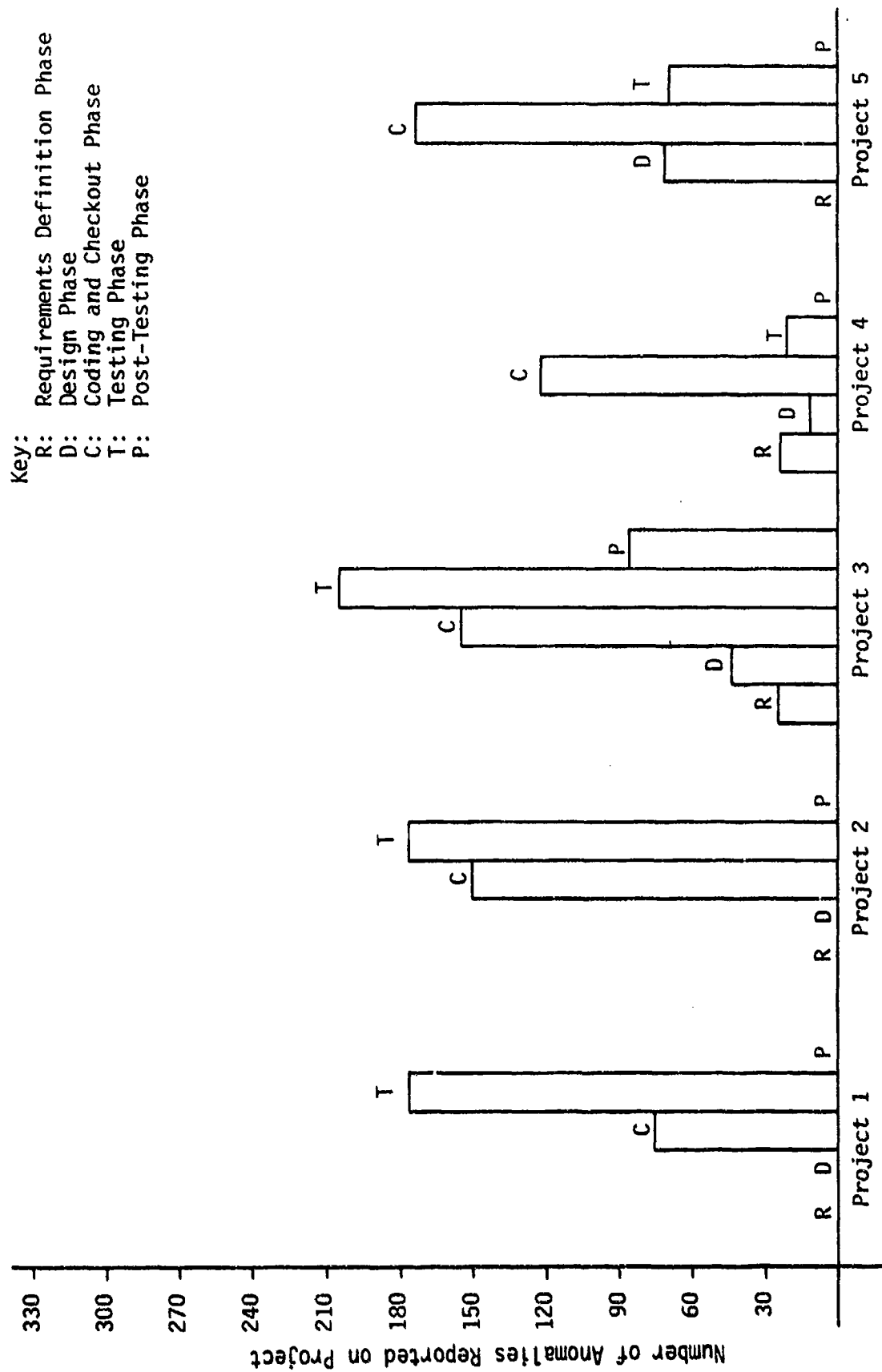


Figure 7. Anomalies Found in Each Development Phase

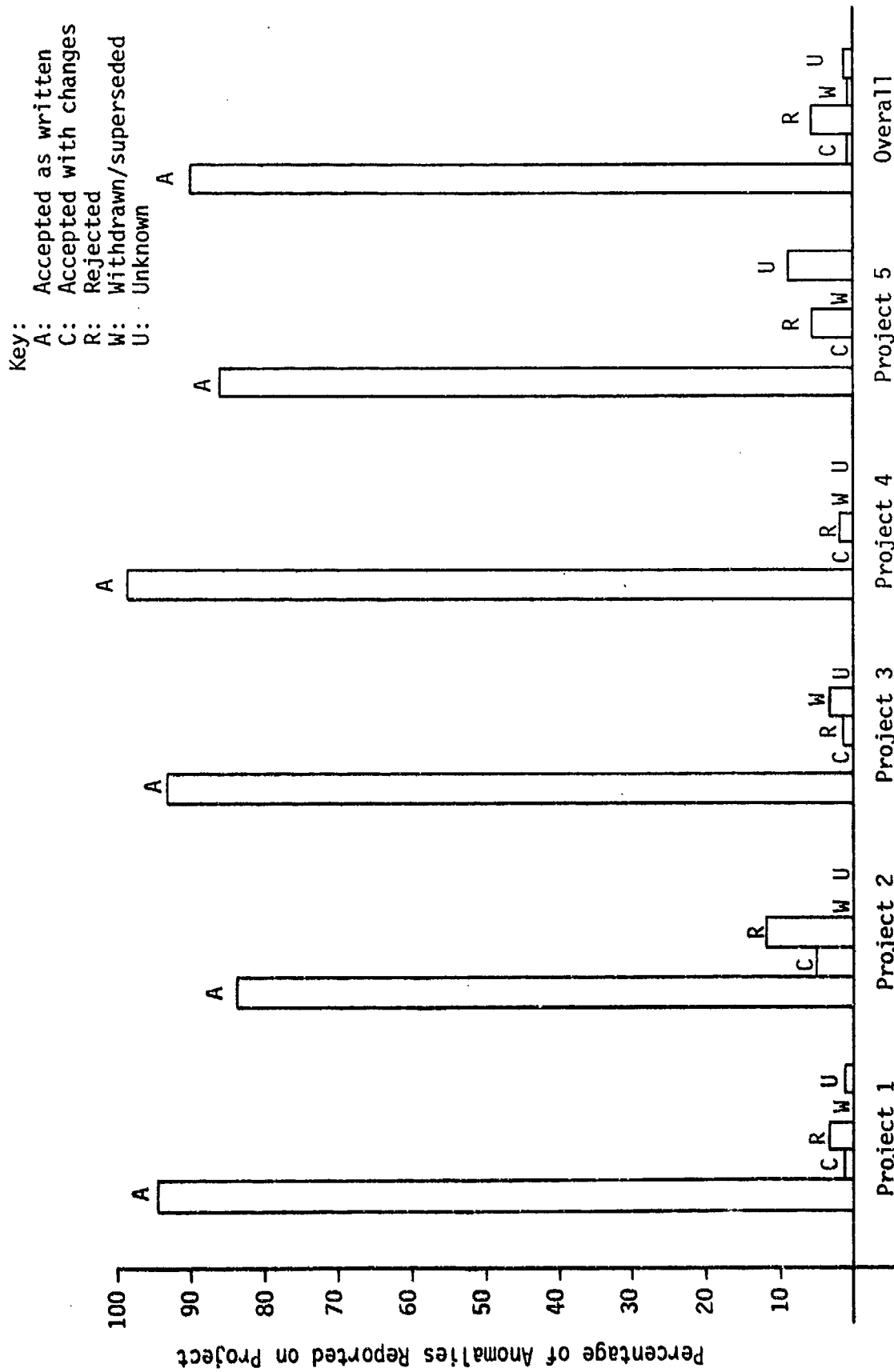


Figure 8. Anomaly Report Acceptance

The results indicate the degree to which IV&V results were both valid and relevant to the software development effort. The acceptance rate on all projects was high, ranging from 83% to 98%. Taking into consideration anomaly reports that were accepted with changes, the acceptance rate ranged from 88% to 98%. Of the anomaly reports for which acceptance was known, 93% were accepted, an indication of the high validity of IV&V results.

3.8 Anomaly Resolution

Figure 9 shows the percentage of anomalies on each project for which:

- Action was taken
- Action was not taken
- Resolution was open or unknown at the time of the study

Anomalies were considered to have been acted on if they were fixed during the project, fixed in a program update, negated by an unrelated change, or dealt with by a work-around solution. Anomalies were considered not to have been acted on if they were rejected, withdrawn, superseded, or accepted but left unchanged. Anomalies were considered to have open or unknown resolution if resolution was deferred to a program update that had not taken place at the time of the study or if resolution could not be determined.

Projects 1 through 4 have similar profiles, indicating a very high percentage of anomalies acted on. These results indicate that IV&V results were not only valid but were sufficiently important to require corrective action. The atypical figures for Project 5 are attributable to the experimental nature of the development project. Anomalies reported in the different program versions did not necessarily require correction. These figures are not typical of most IV&V projects.

3.9 Data Relationships

Tables 5 and 6 present the results of statistical analyses examining the relationships between selected anomaly characteristics. The chi-square statistical procedure was used. A high chi-square value indicates the possibility that the two variables are statistically related. The procedure assumes that there is no association, then computes the probability of observing in repeated samples a relationship less pronounced than that in the current sample. When this probability is less than 5%, there is statistical evidence that the two variables are related; when it is less than 1%, the statistical evidence is even stronger.

The chi-square statistics measure the degree of relationship between each pair of variables. This relationship may or may not be one of cause and effect. The nature of the relationship may be determined by further examination of the data.

Table 5 shows the relationship between anomaly severity ratings and anomaly location, effects, and development phase when detected. The results indicate a significant relationship between severity and these other anomaly characteristics. Specific results revealed by examination of the data are as follows:

key: A: Acted on
 N: Not acted on
 O/U: Resolution open/unknown

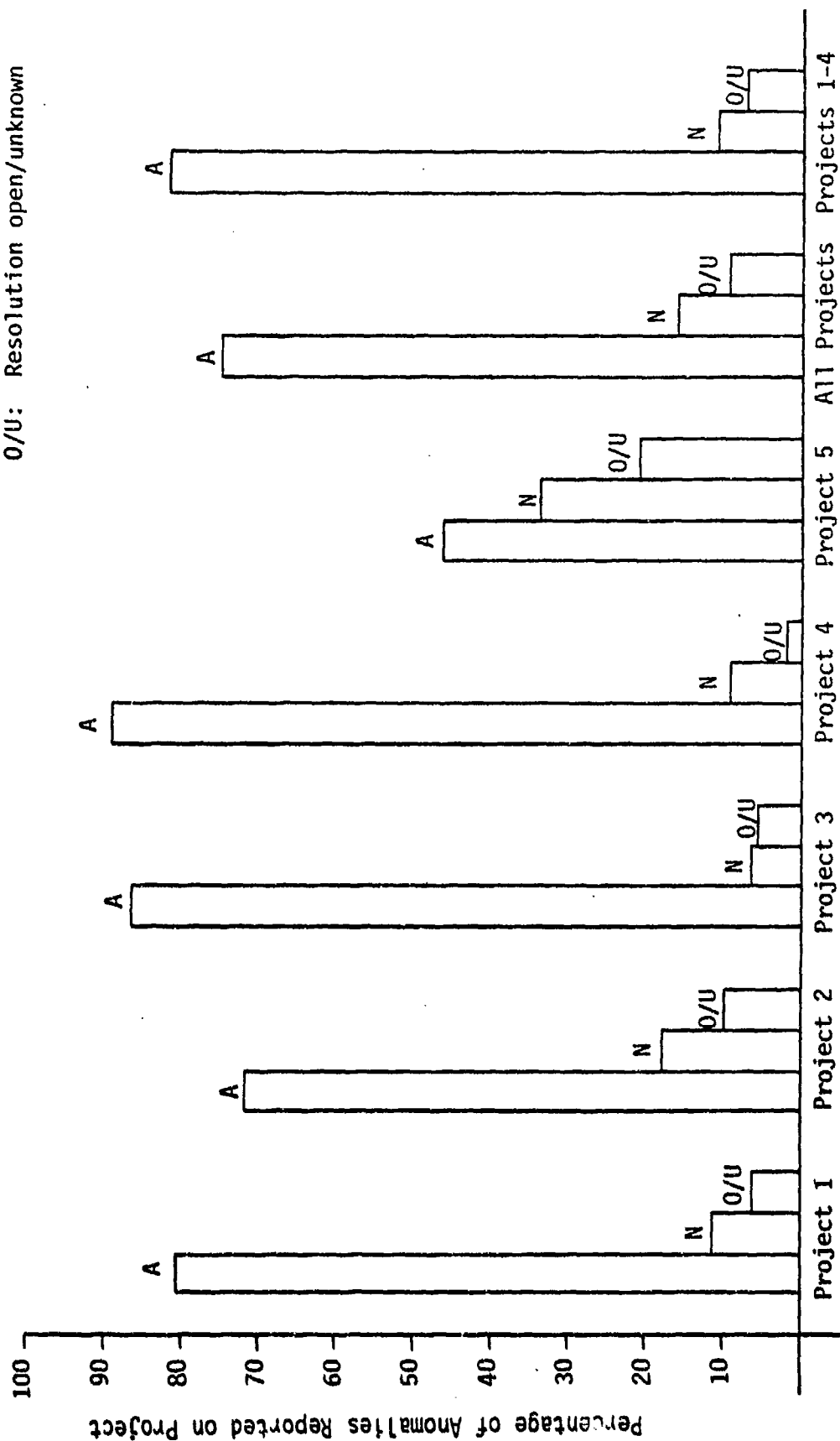


Figure 9. Anomaly Resolution

Table 5. Anomaly Severity Relationships

Note: ** = Significant at 1% level
 * = Significant at 5% level

Analysis		Chi-Square Value
Severity vs. Location (High, Medium, Low)	vs. (Requirements Specification, Before-Code Design Specification, Code, After-Code Design Specification, User Documentation)	267.766**
(High or Medium, Low)	vs. (Code, Other)	24.167**
(High, Medium or Low)	vs. (Code, Other)	214.136**
Severity vs. Primary Effect (High, Medium, Low)	vs. (Operation, Maintainability, Development, Verifiability, Usability, Other)	188.411**
(High or Medium, Low)	vs. (Maintainability, Other)	173.723**
(High, Medium or Low)	vs. (Maintainability, Other)	37.868**
(High or Medium, Low)	vs. (Operation, Other)	168.272**
(High, Medium or Low)	vs. (Operation, Other)	40.257**
Severity vs. Operational Effect (High, Medium, Low)	vs. (Correctness, Accuracy, Security, Efficiency, Other)	244.821**
(High or Medium, Low)	vs. (Correctness, Other)	46.617**
(High, Medium or Low)	vs. (Correctness, Other)	1.803
(High or Medium, Low)	vs. (Accuracy, Other)	41.233**
(High, Medium or Low)	vs. (Accuracy, Other)	3.084
(High or Medium, Low)	vs. (Efficiency, Other)	12.297**
(High, Medium or Low)	vs. (Efficiency, Other)	6.743**
Severity vs. Development Phase When Detected (High, Medium, Low)	vs. (Requirements Definition, Design, Coding, Testing)	36.538**
(High or Medium, Low)	vs. (Requirements or Design, Coding or Testing)	4.496*
(High, Medium or Low)	vs. (Requirements or Design, Coding or Testing)	0.472
(High or Medium, Low)	vs. (Testing, Other)	10.731**
(High, Medium or Low)	vs. (Testing, Other)	0.0

Table 6. Anomaly Resolution Relationships

Note: ** = Significant at 1% level
 * = Significant at 5% level

Analysis		Chi-Square Value
Resolution vs. Location (Corrected, Not Corrected, vs. Open/Unknown)	(Requirements Specification, Before-Code Design Specification, Code, After-Code Design Specification, User Documentation)	175.014**
	(Code, Other)	18.194**
Resolution vs. Primary Effect (Corrected, Not Corrected, vs. Open/Unknown)	(Operation, Maintainability, Development, Verifiability, Usability, Other)	25.011**
	(Operation, Other)	3.447
	(Maintainability, Other)	4.446*
Resolution vs. Operational Effect (Corrected, Not Corrected, vs. Open/Unknown)	(Correctness, Accuracy, Security, Efficiency, Other)	25.698**
	(Correctness, Other)	4.852*
	(Accuracy, Other)	6.667**
	(Efficiency, Other)	1.430
Resolution vs. Development Phase When Detected (Corrected, Not Corrected, vs. Open/Unknown)	(Requirements Definition, Design, Coding, Testing)	76.166**
	(Requirements or Design, Coding or Testing)	2.527
	(Testing, Other)	19.376**
Resolution vs. Severity (Corrected, Not Corrected, vs. Open/Unknown)	(High, Medium, Low)	19.480**
	(High or Medium, Low)	1.649
	(High, Medium or Low)	17.913**

- Code anomalies are more likely to be rated High or Medium than anomalies in other development materials.
- Anomalies in the after-code design specification and user documentation are almost always rated Low.
- Anomalies affecting reliability are the only type likely to be assigned High ratings.
- Most anomalies with maintainability as their primary effect are rated Low.
- Anomalies detected during the requirements definition phase are more likely to be rated High than those detected later.
- Anomalies detected during the coding and testing phases have severity ratings very close to the overall average of 7.2% High, 26.4% Medium, 66.4% Low.

Table 6 shows the relationship between various anomaly characteristics and anomaly resolution. Here again, significant relationships exist. Discounting results attributable to the atypical resolution pattern of Project 5, the following results can be observed from the data:

- Anomalies in all categories are far more likely to be acted on than not.
- Anomalies affecting maintainability and usability, while seemingly less significant than those affecting reliability, have an even higher probability of being acted on than reliability anomalies; anomalies concerned with efficiency have a lower probability.
- Not surprisingly, anomalies with High severity have the greatest probability of being acted on; the probabilities for Medium and Low anomalies are approximately equal.
- Anomalies detected during the coding and checkout phase of development are the most likely to be acted on.

4. RESULTS CONCERNING SOFTWARE RELIABILITY

The primary concern of IV&V is software reliability, defined by Boehm as the extent to which software can be expected to perform its intended functions satisfactorily (Reference 3).^{*} Included within the scope of reliability are:

- Operational Correctness: Ensuring that the software performs all intended functions satisfactorily and performs no unintended functions
- Operational Accuracy: Ensuring that mathematical functions are performed with the required accuracy/precision
- Operational Security: Ensuring that the program is free of unauthorized coding and incorporates all required measures to prevent access to software and data by unauthorized persons

The major difficulty in evaluating IV&V's effect on software reliability is the possibility that the developer may eventually have detected some or all of the problems reported by the IV&V agency without the latter's help. The fact that IV&V was the first to find them proves that:

- IV&V is capable of detecting development problems.
- IV&V provides visibility into the development process.
- IV&V finds problems earlier than development testing and may therefore prevent the schedule slips and cost overruns that result from late detection.

It does not necessarily prove that without the aid of IV&V, these problems would have gone undetected into the operational environment.

The ideal experiment for evaluating IV&V's effect on software reliability would be to have two groups of equally experienced and talented programmers working in equivalent development environments develop the same program using the same methods and tools. An IV&V group would be assigned to one of the development efforts, and the resulting programs would be compared for reliability. If the software that had undergone IV&V was more reliable than that developed without it, it could be concluded that IV&V did indeed have a positive effect.

The IV&V study was forced to take a far more limited approach, consisting of surveying the literature for relevant results and examining the data from the five IV&V projects in light of these research findings. The results of these activities are described in the following paragraphs.

^{*}Boehm, B. W., et al., "Characteristics of Software Quality," TRW Software Series TRW-SS-73-09, Dec. 1973.

4.1 Relevant Findings in the Literature

A number of studies have noted the effects of submitting a program to two or more test and evaluation groups in succession. Proceedings of a TRW symposium on software development (Reference 4)* reported that on a large development project, each successive phase of testing followed the same pattern. Faults were found at a high rate at the beginning of the phase, then at lower and lower rates as the phase continued. When the program was turned over to a new test group for the next testing phase, the detection rate jumped up sharply and the pattern began anew. The report theorized that the different techniques of each test group resulted in the renewed fault detection rate.

Thayer reported similar findings in a study of five large development efforts (Reference 5).† He identified as the cause of this phenomenon the expanded test objectives and fresh viewpoint of each successive test group. In his study, successive test groups sometimes found more faults than their predecessors had.

Two other findings of the Thayer study are also worthy of note. The first is that each test group detected faults that should have been detected in previous test phases. That is, in addition to those faults detected because of expanded test objectives, each test group detected faults within the scope of previous test efforts. The fresh viewpoint and different test techniques of each group were considered to be the factors here.

The second finding was the tendency of each test group, and in particular each test analyst, to report several faults of a similar type over a period of a day or two. Having detected a certain type of fault, the analyst made a specific search for that type of problem in other parts of the program. Thayer states that this tendency can have very positive effects on the rate and completeness of fault discovery, especially if the analyst is intimately familiar with all of the code produced by a given programmer.

Studies on the effects of various tools and techniques are also relevant. A study by Shooman and Bolsky (Reference 6)* found that a large proportion of program faults can be detected by code inspection without resorting to computer testing. A study by Rubey (Reference 2) found that analysis methods detect faults earlier than testing methods but that both methods are needed to

*Proceedings of the TRW Symposium on Reliable, Cost-Effective, Secure Software, March 1974, pp. 5.13-5.17.

†Thayer, T. A. et al., Software Reliability Study, RADC-TR-76-238, Feb. 1976.

*Shooman, M. L., and Bolsky, M. I., "Types, Distribution, and Test Correction Times for Programming Errors," Proceedures of the International Conference on Reliable Software, April 1975, pp. 34-357.

find all types of faults. Finally, Dana and Blizzard (Reference 7)* indicate that certain tools and techniques are most effective in detecting each type of fault.

A third set of results concern the benefits of early detection on program reliability. Research reported by Finfer (Reference 8)† indicates that:

- The reliability of a system is greatly affected when problems of one development phase are allowed to go uncorrected into subsequent phases.
- Design errors found in integration and system testing have a much greater impact on reliability than if they had been detected during the design phase.
- "Crash" development and remediation efforts generally result in poor system design and poor-quality software.

The implications of these findings for IV&V include the following:

- The fresh viewpoint, independent objectives, and specialized tools and techniques offered by IV&V can be expected to disclose software faults not detected by developer testing.
- The manual analysis techniques used by IV&V can be expected to disclose software faults not detected by developer testing.
- IV&V analysis and testing combined can be expected to detect faults in all categories.
- The early detection of problems provided by IV&V can provide the time needed for effective redesign, thereby improving program reliability.
- The IV&V analyst's intimate familiarity with the program undergoing evaluation should make possible the detection of whole classes of related faults.

The last phenomenon is a recognized aspect of IV&V. Often called the "clone effect," it accounts for the detection of numerous anomalies on most IV&V projects (Reference 9).‡

*Dana, J. A., and Blizzard, J. D., The Development of a Software Error Theory to Classify and Detect Software Error, Logicon Report HR-74012, May 1974.

†Finfer, M. C., Software Data Collection Study, Volume III: Data Requirements for Productivity and Reliability Studies, RADC-TR-76-329 Vol. III, June 1976.

‡Radatz, J. W., Ramsey, O. C., and McKillop, T. L., NSCCA/PATE Guidebooks, Volume III, Logicon Report R:SED-80204-III, June 1980.

4.2 Project Results

The key questions addressed by the study concerning IV&V's effects on software reliability were as follows:

- How many of the anomaly reports submitted by IV&V had an effect on program reliability?
- In what development materials were the anomalies located?
- What types of problems did they involve?
- What aspects of reliability did they affect?
- How severe were their consequences?
- When were they found?
- What was the operational reliability of the completed programs?

The following paragraphs discuss these issues.

4.2.1 Number of Anomaly Reports Affecting Reliability

Of the 1575 anomaly reports submitted on the IV&V projects, 1023 were concerned with software reliability. Broken down by project, the numbers were as follows:

- Project 1: 229
- Project 2: 300
- Project 3: 183
- Project 4: 95
- Project 5: 216

Only a subset of these reports had an actual effect on program reliability, namely, those that were accepted as valid by the program office and acted on by the developer. Figures 10 and 11 show the percentage of anomaly reports that met these criteria.

Figure 10 indicates program office acceptance of the anomaly reports concerned with reliability. On the average, 89% were accepted as written, an additional 2% were accepted with changes, 7% were rejected, 1% were withdrawn or superseded, and for 1%, acceptance was unknown. Project 4 had the highest acceptance rate, with 98%.

Figure 11 indicates the action taken on reliability anomalies. For reasons described in Section 3.8, the resolution seen for Projects 1-4 is more typical of IV&V projects than that shown for Project 5. The average for these four projects was 79% acted on, 14% not acted on, and 7% unknown or pending.

There is no way of knowing how many of these anomalies would have been detected by the developer without IV&V. The results of the literature search imply that some at least would not have been. For purposes of the study, the following assumption was made: Any report that was concerned with program reliability, accepted as valid by the program office, and acted on by the developer represents an improvement in program reliability attributable to IV&V.

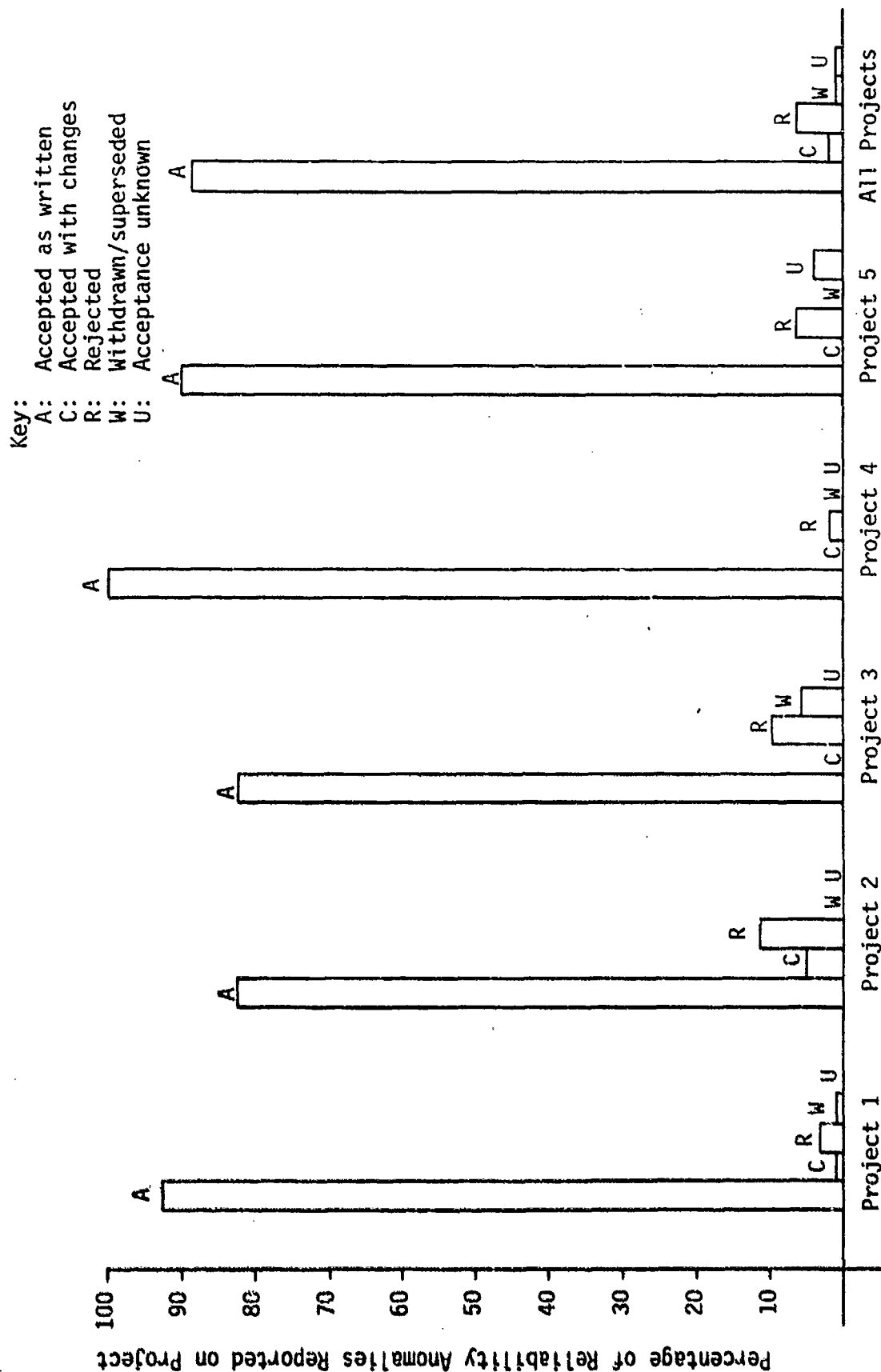


Figure 10. Acceptance of Anomaly Reports Concerned With Reliability

Key: A: Acted on
 N: Not acted on
 O/U: Resolution open/unknown

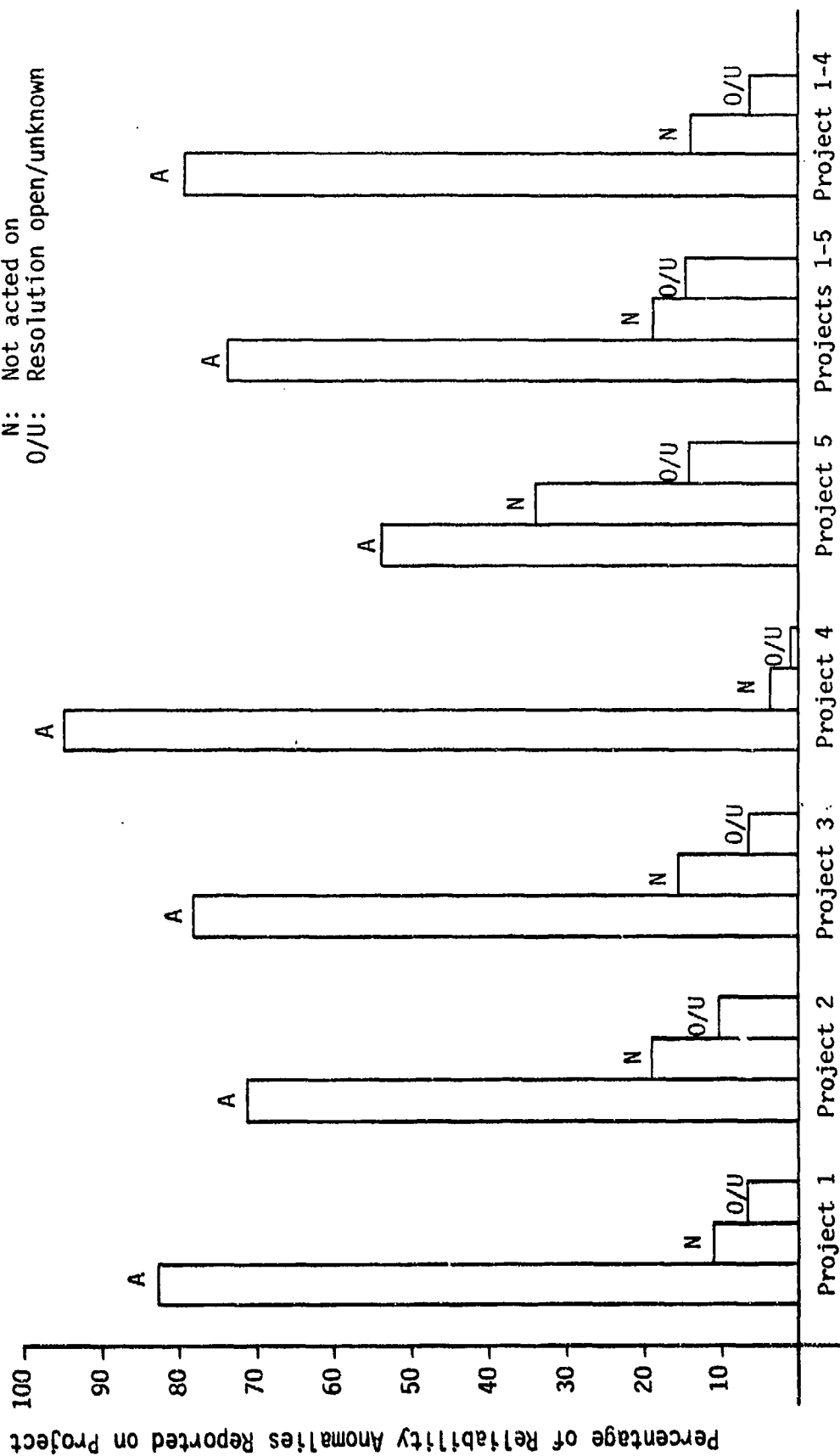


Figure 11. Resolution of Anomalies Concerned With Reliability

There were 748 such anomaly reports. For convenience, the anomalies they describe are hereafter referred to as "corrected reliability anomalies." The breakdown of these anomalies by project was as follows:

- Project 1: 188
- Project 2: 216
- Project 3: 139
- Project 4: 90
- Project 5: 115

To normalize these figures, they were compared with the number of machine language instructions generated by the programs examined. The results are shown in Figure 12. Project 3 had the highest number, with 3.6 per thousand machine language instructions; Project 4 had the lowest, with 1.2. On the average, IV&V resulted in the correction of 2.2 reliability anomalies per thousand machine language instructions.

4.2.2 Anomaly Location

Anomalies affecting reliability could be found in requirement specifications, before-code design specifications, code, or other materials such as trade study reports. Figure 13 shows the number of corrected reliability anomalies found in each of these development materials. Overall, 33% were found in requirement specifications, 5% in before-code design specifications, 61% in code, and 1% in other materials. On Project 5, the only project to perform a standard design verification, over a fourth of the corrected reliability anomalies were in the before-code design specification. The other projects reported most or all of the anomalies in requirement specifications and code.

4.2.3 Anomaly Categories

Table 7 indicates the number of corrected reliability anomalies found in each anomaly category. Significant results are as follows:

- IV&V resulted in the correction of 245 requirement anomalies that would have affected reliability. In 38% of these cases, the requirements were incorrect; in another 28%, they were incomplete. In 21% the requirements were inconsistent; in 12% they were ambiguous, unfeasible, or otherwise unsatisfactory for software reliability.
- IV&V resulted in the correction of 448 code anomalies that would have affected reliability. Over a third concerned an incorrect or unsatisfactory choice of algorithm or mathematics for the program. Nearly a fourth concerned incorrect handling of program data. An additional 11% were concerned with incorrect interfaces or program input/output.
- IV&V methods resulted in the detection of code anomalies in all categories. It could not be determined how many of these anomalies resulted from the "clone effect."

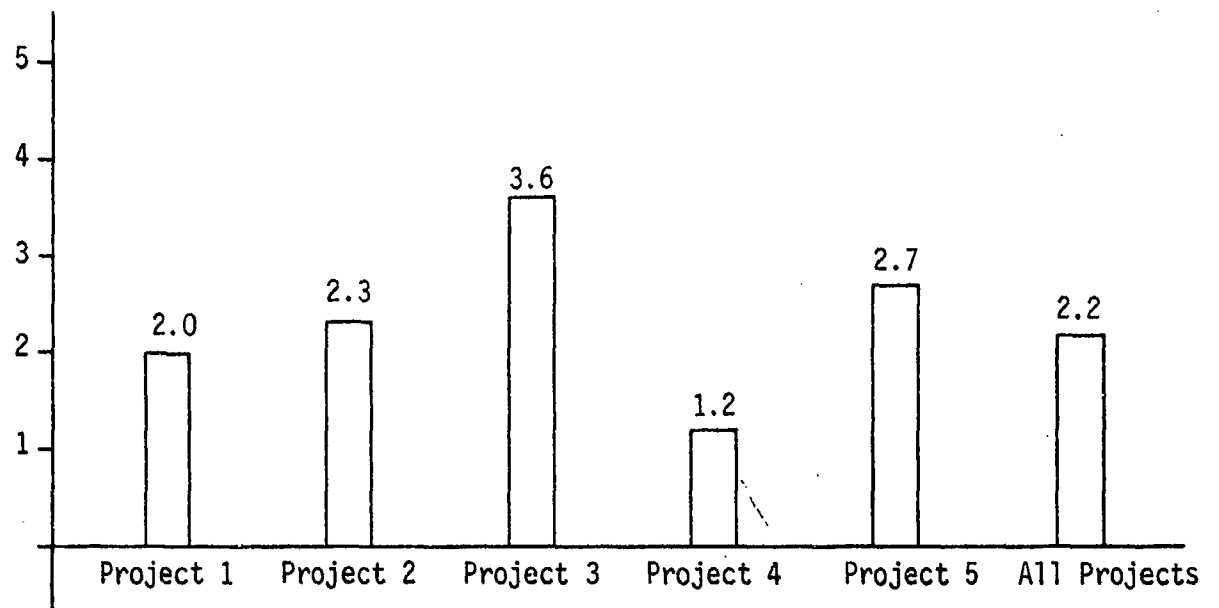


Figure 12. Corrected Reliability Anomalies Per Thousand Machine Instructions

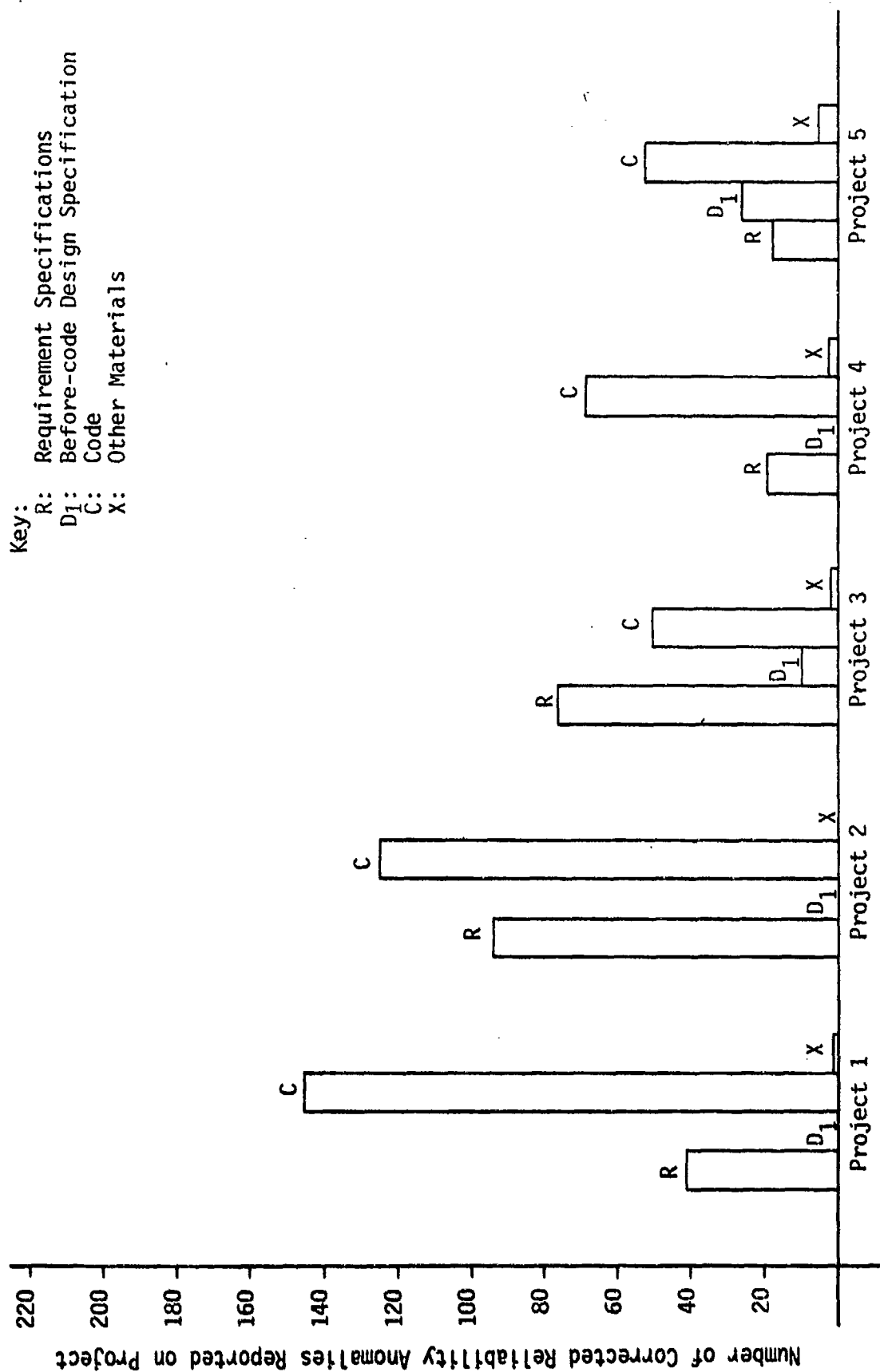


Figure 13. Development Materials In Which Corrected Reliability Anomalies Were Found

Table 7. Number of Corrected Reliability Anomalies in Each Category

Anomaly Category	Project					All
	1	2	3	4	5	
Requirement Specification Anomalies						
R1. Incorrect Requirements	12	47	32	3	--	94
R2. Inconsistent Requirements	8	9	12	6	17	52
R3. Incomplete Requirements	16	22	21	9	2	70
R4. Other Requirement Problems	5	15	8	1	--	29
R5. Presentation; Standards Compliance	N/A	N/A	N/A	N/A	N/A	N/A
Total	41	93	73	19	19	245
Before-Code Design Specification Anomalies						
D1. Requirement Compliance	--	--	8	--	1	9
D2. Choice of Algorithm, Mathematics	--	--	2	--	7	9
D3. Sequence of Operations	--	--	--	--	5	5
D4. Data Definition	--	--	--	--	1	1
D5. Data Handling	--	--	--	--	14	14
D6. Timing, Interruptibility	--	--	--	--	--	--
D7. Interfaces, I/O	--	--	--	--	2	2
D8. Other Design Problems	--	--	--	--	--	--
D9. Presentation; Standards Compliance	N/A	N/A	N/A	N/A	N/A	N/A
Total	--	--	10	--	30	40
Code Anomalies						
C1. Requirement, Design Compliance	5	3	14	5	1	28
C2. Choice of Algorithm, Mathematics	71	46	7	15	24	163
C3. Sequence of Operations	7	3	9	3	10	32
C4. Data Definition	6	16	1	18	1	42
C5. Data Handling	22	34	16	16	16	104
C6. Timing, Interruptibility	18	6	--	--	2	26
C7. Interfaces, I/O	17	14	5	9	6	51
C8. Other Code Problems	--	1	1	--	--	2
C9. Presentation; Standards Compliance	N/A	N/A	N/A	N/A	N/A	N/A
Total	146	123	53	66	60	448
After-Code Design Specification Anomalies	N/A	N/A	N/A	N/A	N/A	N/A
User Documentation Anomalies	N/A	N/A	N/A	N/A	N/A	N/A
Other Anomalies	1	--	3	5	6	15
Anomalies in All Categories	188	216	139	90	115	748

4.2.4 Aspects of Reliability That Were Affected

The key aspects of software reliability are operational correctness, accuracy, and security. Figure 14 indicates the distribution of each project's corrected reliability anomalies into these three areas. Totals may exceed the number of anomalies reported because of multiple effects.

By far the most prevalent aspect was operational correctness. Most of the anomalies for Projects 1, 2, and 3, and all for Projects 4 and 5 fell into this category. This preponderance is partly because the other two aspects of reliability do not apply to all types of software. Accuracy applies primarily to programs that perform calculations for which various degrees of accuracy or precision can be achieved. Anomalies concerned with this aspect of reliability were reported for Projects 1, 2, and 3. Security applies to software that can be threatened with unauthorized alteration or misuse. This aspect was limited to Projects 1 and 2, and accounted for only a small percentage of the anomalies on these projects. The greatest effect of IV&V lies in assuring that the subject program operates as expected.

4.2.5 Anomaly Severity Ratings

Figure 15 indicates the severity ratings assigned to the corrected reliability anomalies. The overall figures show that about a tenth of the anomalies received High ratings, a third were rated Medium, half were rated Low, and for 4%, the severity was unknown. Extremes were exhibited by Project 1, on which over two-thirds had High or Medium ratings, and by Project 4, on which 85% had Low ratings. Projects 2, 3, and 5 fell closer to the overall average.

In the context of reliability, these ratings have the following general interpretation:

- High: threat to life or property
- Medium: serious threat to mission objectives
- Low: degraded system performance

The seriousness of the High and Medium impacts and the fact that nearly half of the anomalies had these ratings indicates the importance of these IV&V results.

4.2.6 Phase of Anomaly Detection

Figure 16 shows the number of corrected reliability anomalies detected during each development phase. Nearly two-thirds of the anomalies were reported before development testing. On Project 4, 93% of the anomalies were reported before the testing phase; on Project 3, 81%. All projects except Project 1 reported well over half of their corrected reliability anomalies before development testing.

The significance of these findings lies in the results reported by Finfer: Early detection of anomalies provides time for effective redesign, thereby improving program reliability.

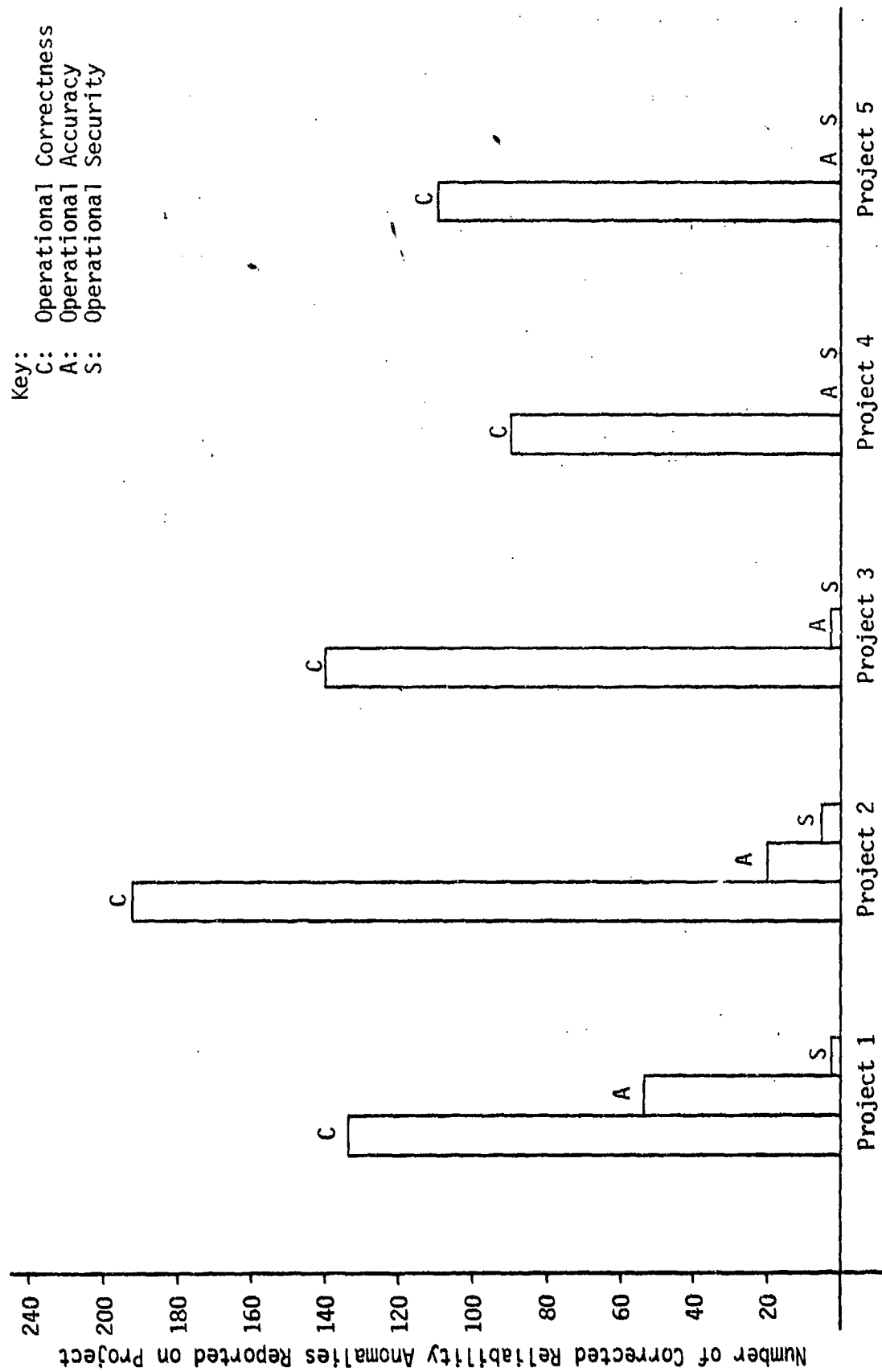


Figure 14. Number of Anomalies Affecting Each Aspect of Reliability

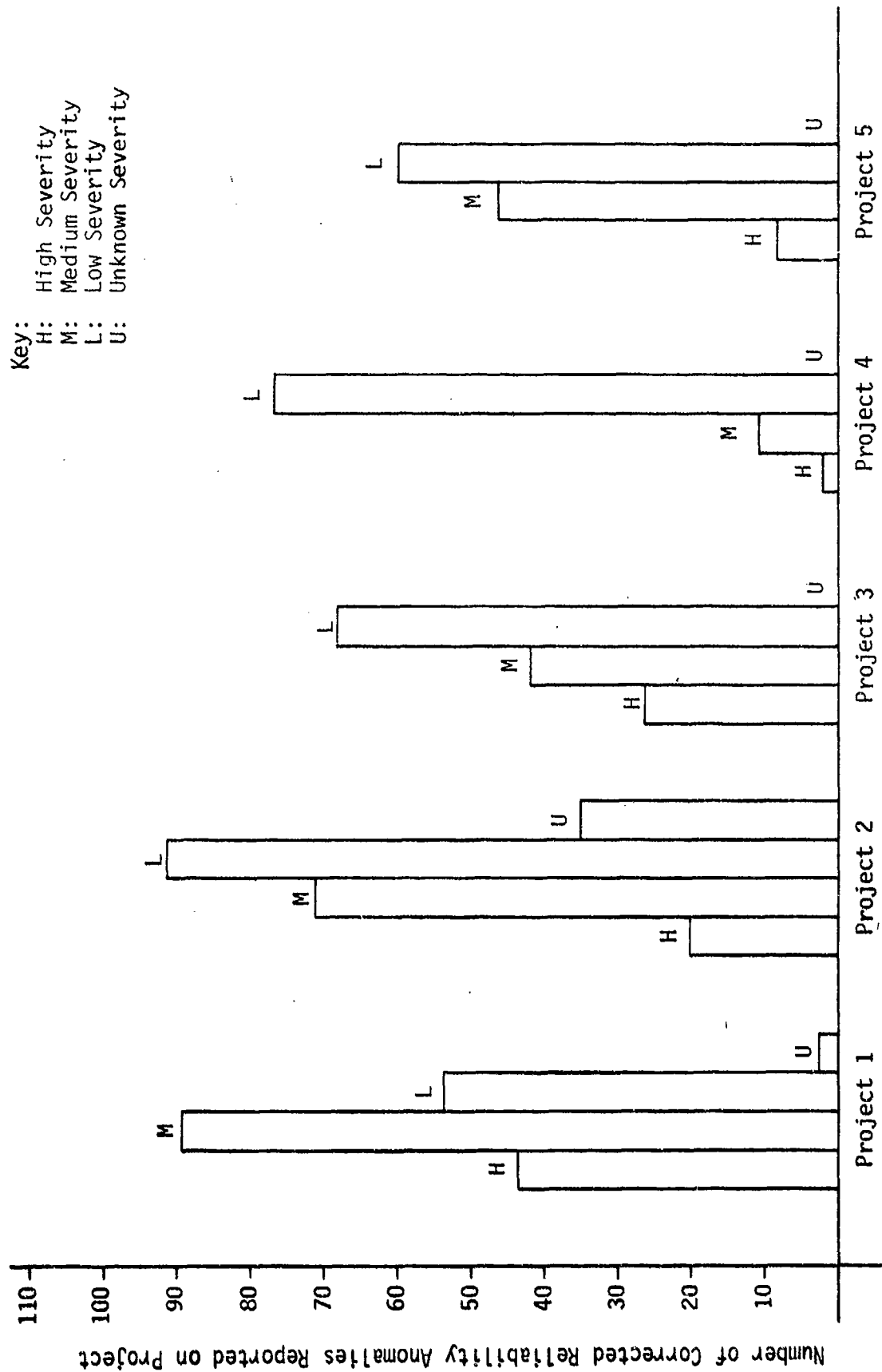


Figure 15. Severity Ratings of Corrected Reliability Anomalies

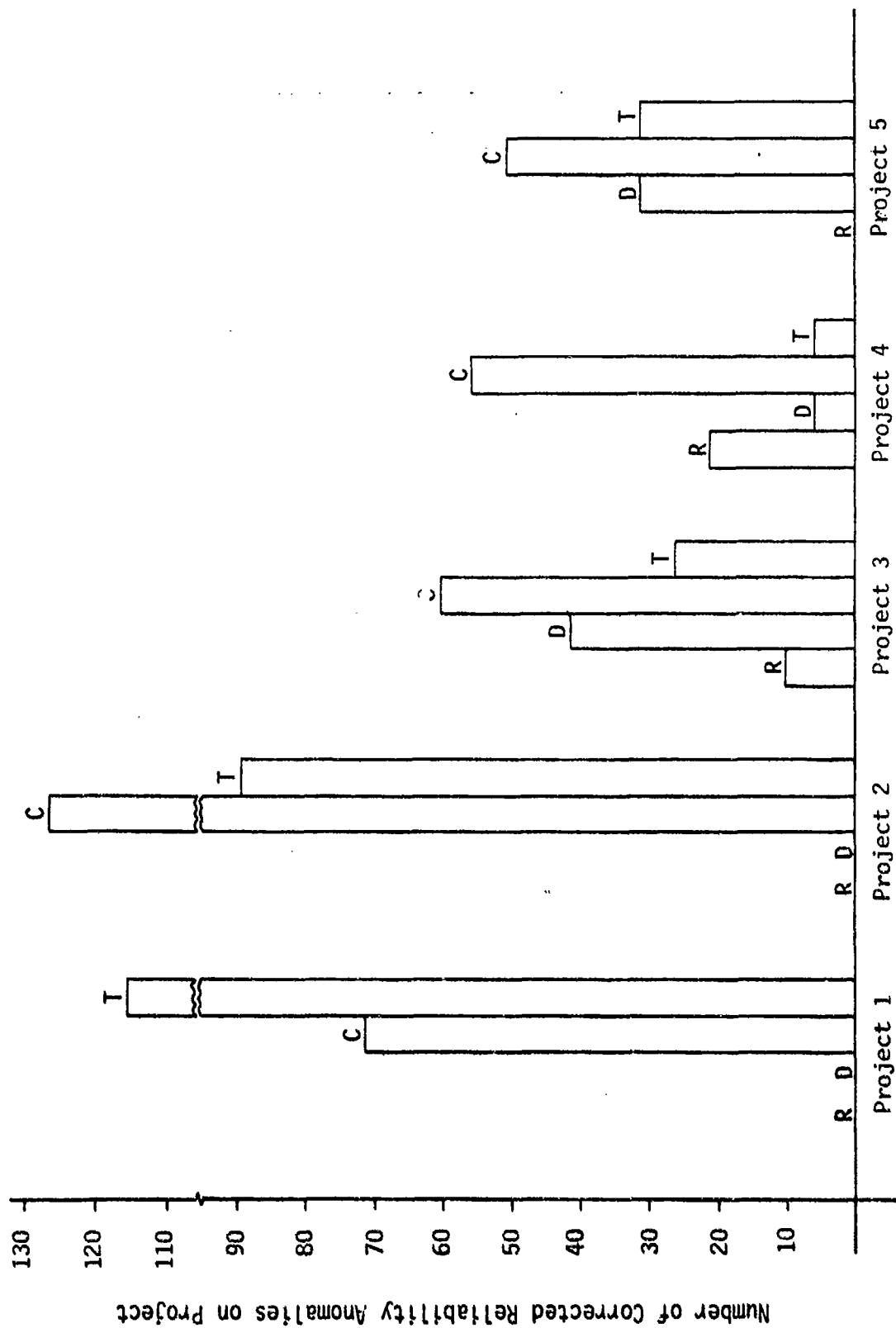


Figure 16. Development Phase In Which Corrected Reliability Anomalies Were Detected

4.2.7 Operational Performance of the Completed Programs

To assess the reliability of the completed programs, the following questions were considered by the development project questionnaire:

- How many problems have been reported since the program became operational?
- If the program has been modified, was it due to operational problems, requirement changes, or other reasons?

The responses were as follows:

- For the Project 1 software, four problems were reported in operational use. When the software was modified, however, the purpose was to respond to requirement changes rather than to correct any operational problems.
- For the Project 2 software, no problems were reported in operational use. The software was modified to respond to requirement changes.
- No usable responses were given for the Project 3 software; records of its performance were combined with those of interfacing programs and could not be separated out.
- For the Project 4 software, no operational problems had been reported, and the software had not yet been modified.
- The Project 5 software had not yet been put into operational use; its operational performance and maintenance needs were therefore unknown.

For the three projects for which data was available, therefore, none had required modification to correct a reliability problem encountered in the operational environment. There is no way to establish that IV&V was responsible for this high reliability. It is safe to say, however, that with an average of 150 anomaly reports per project having a direct bearing on the improvement of reliability, IV&V made a significant contribution.

5. RESULTS CONCERNING SOFTWARE MAINTAINABILITY

The overall cost of a software product may be far greater than the cost to develop it. Figures cited by Miller (Reference 10),* Fife (Reference 11),† and others indicate that software maintenance cost--the cost of modifying a program after it has become operational--may account for up to 70%-75% of its total life cycle cost. A 1976 paper by Prokop (Reference 12)* stated that two out of every three Navy programmers and computer systems analysts were involved in maintaining existing software. Nolan and Robinson (Reference 13)** have found that all data processing organizations eventually reach a stage in which 70% of the effort is devoted to maintenance activities.

Modifying existing software is a difficult, error-prone process. A study by McGonagle (Reference 14)†† reports that 19% of all errors detected in the software of one organization resulted from unexpected side effects to other changes. In Reference 15,** Boehm reports that even for small modifications (1 to 10 instructions), the chance of a successful first run is at best 50%, and for larger changes, the success rate decreases steadily to about 15%. Lehman (Reference 16)*** states that software tends to become more and more complex with each change, making each modification more difficult than the last.

Increasing awareness of both the likelihood and the difficulty of software maintenance has resulted in new attitudes toward software development.

*Miller, C. R., "Software Maintenance and Life Cycle Management," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 53-59.

†Fife, D. W., "Software Management Standards," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 63-80.

*Prokop, J., Computers in the Navy, Annapolis, MD, Naval Institute Press, 1976.

**Robinson, D. G., "Beyond the Four Stages: What Next," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 187-201.

††McGonagle, J. D., A Study of a Software Development Project, James P. Angerson and Co., Sept. 1971.

**Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp. 48-59.

***Lehman, M. M., "Evolution Dynamics--A Phenomenology of Software Maintenance," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 313-323.

Delivered software is no longer viewed as a finished product not intended for change. Instead, it is assumed that the operational environment will be dynamic and that software will be required to change along with it. The result is increasing emphasis on software that is not only reliable, but maintainable as well.

The following approach was taken to investigate the effect of IV&V on software maintainability:

- Identify from the literature software attributes that have been shown to contribute to maintainability.
- Formulate hypotheses about IV&V's potential to affect these attributes.
- Analyze the results of the five IV&V projects in light of these hypotheses.

The results of these activities are described in the following paragraphs.

5.1 Software Attributes That Contribute to Maintainability

Software maintenance may be performed to remove or correct a software fault, to add new features or capabilities, to delete unused or undesirable features, or to adapt the software to hardware changes (Reference 17).^{*} Regardless of the motivation for change, however, the maintenance process consists of understanding the existing software, making the needed changes, and revalidating the modified software (Reference 18).[†] Software that has been designed, coded, and documented in a way that facilitates these tasks is said to be "maintainable."

According to Peercy, the three basic attributes of maintainable software are:

- Understandability: The ease with which the purpose and organization of the software can be grasped
- Modifiability: The ease with which changes can be incorporated once the nature of the desired change has been identified
- Testability: The extent to which the software supports evaluation of its performance

^{*}Peercy, D. E., "A Software Maintainability Evaluation Methodology," Proceedings of the AIAA 2nd Computers in Aerospace Conference, Oct. 1979, pp. 315-325.

[†]Boehm, B. W., "Software Engineering," IEEE Transactions on Computers, Dec. 1976, pp. 1226-1241.

To these three characteristics Neil and Gold (Reference 19)* add:

- Portability: The ease with which a software product can be transferred from one computer environment to another

Specific software features contribute to each of these attributes. Features that contribute to understandability include complete, accurate documentation, good traceability between code and requirements, and code and design that are modular, self-descriptive, noncomplex, and consistent. Features that contribute to modifiability include data structures designed to allow for ease of expansion and change, code and data structures that minimize the side effects of changes, and documentation that corresponds to the code and is modular in nature. Features that contribute to testability include software structures that isolate the effects of changes, program instrumentation, and complete, accurate documentation. Features that contribute to portability include device independence, use of higher order language, and minimization of interfaces with other systems. Appendix C identifies more specifically a variety of features that contribute to software maintainability.

5.2 IV&V's Potential for Improving Maintainability

Software maintenance is rarely performed by the original programmer. More typically, a person unfamiliar with the program must study the code and its documentation until he understands the program well enough to make the needed changes and to devise test cases to requalify the program.

The similarity of this process to the IV&V process is striking. The IV&V analyst must study the documentation and code until he becomes sufficiently familiar with the program to follow the programmer's thought processes, detect logical flaws, identify situations that the programmer may have failed to consider, and devise test cases to thoroughly test the program.

The objectives of the maintenance programmer and the IV&V analyst differ. The maintenance programmer wants to modify the program, the IV&V analyst to evaluate it. The similar preparations that both must make to perform these functions, however, suggest that the IV&V analyst is in an excellent position to assess software maintainability. The following paragraphs explore this hypothesis by addressing each of the four major aspects of maintainability and the special case in which IV&V is applied to a maintenance effort. A concluding paragraph discusses indirect effects of IV&V's assessment of software reliability.

5.2.1 Understandability

Understandability is as crucial to the IV&V analyst as it is to the maintenance programmer. In the analyst's efforts to become familiar with the requirement and design specifications, he notices incompleteness, inconsis-

*Neil, G., and Gold, H. I., Software Acquisition Management Guidebook: Software Quality Assurance, ESD-TR-77-255, Aug. 1977.

tencies, inaccuracies, ambiguities, unclear presentation, and other problems of this type because they hamper his own efforts to understand the software system. Similarly, during the detailed code analysis that is central to IV&V, such problems as inadequate or incorrect comments, unstructured or unmodular code, complex constructs, and obscure logic present the IV&V analyst with the same difficulties they would present the maintenance programmer. Assessing understandability is therefore a natural part of IV&V; the findings can be reported if the program office so chooses.

5.2.2 Modifiability

According to Peercy, modifiability consists of understandability plus expandability, where expandability is the extent to which a physical change to information, computational functions, data storage, or execution time can be easily accomplished. Software features that contribute to expandability include a reasonable margin of storage space and processing time, extra fields in data files, parameterization of constants and data structure sizes, and documentation that will easily accommodate change.

Although few if any IV&V projects have been chartered to evaluate software for modifiability, anomalies concerning this trait are often reported under the category "poor programming practices," and the potential to expand this evaluation certainly exists. Data bases that are being examined for accuracy could be simultaneously evaluated for expandability. Code being examined for correctness and efficiency could also be evaluated against a set of criteria known to enhance expandability. Documentation being examined for correctness, completeness, and consistency could also be evaluated for the type of modularity and traceability that enhance expandability. Drawing from a list such as that given in Appendix C, a set of explicit modifiability criteria could be developed against which the software was to be evaluated. Manual analyses of code, data, and documentation would then include these criteria along with those normally used. Special tools to evaluate certain features might also be developed.

5.2.3 Testability

Peercy defines testability as understandability plus instrumentation, where instrumentation is the extent to which software contains embedded test aids or has been implemented to allow the use of external test aids. Embedded aids might include assertions or execution monitoring statements; external test aids might include drivers, monitors, simulators, or test case generators.

An important aspect of IV&V requirements verification is evaluation of each requirement for testability. Considered in the evaluation are the clarity, quantifiability, and feasibility of each requirement. While evaluation of the design and code for testability has not traditionally been included in the IV&V charter, here again, the potential exists. If the code has been instrumented with assertions indicating expected conditions on inputs, outputs, program variables, or other program aspects, the analyst could evaluate the completeness and quality of these embedded test aids. If it does not contain such instrumentation, recommendations could be made for ways to include assertions or to accommodate external test aids. Software structures could be

evaluated for their ability to isolate the effects of change. These and other testability criteria could be developed, and deviations could be reported in anomaly or other types of reports.

5.2.4 Portability

Portability contributes to maintainability by reducing the need for modification when new equipment is introduced or when the software is transferred to a new environment. Important aspects of portability are the use of a higher order language and minimization of equipment dependencies. Even when these measures are adopted, however, portability is difficult to achieve. Differences in computer word sizes make compatibility between some systems very difficult to achieve. Small segments of assembly language tend to appear in programs that are supposed to be written in higher order language. Higher order languages, despite their claims of portability, have been adapted to particular computer systems.

From the point of view of IV&V, portability is a trait that can be evaluated quite effectively. Guidebooks exist (e.g., Reference 20)* which identify higher order language constructs that are truly hardware independent. Criteria can be established to evaluate device independence. Evaluation of code for portability could be included as part of the code verification process.

5.2.5 IV&V of Maintenance Efforts

The preceding sections were concerned with software development. Another aspect of the maintainability issue is ensuring that software undergoing maintenance does not become less maintainable than it was before.

In his survey of software maintenance technology (Reference 21),† Donahoo cites the following four issues as the major concerns in software maintenance:

- 1) Lehman's "Law of Increasing Entropy": The complexity of a program tends to increase with each modification, making maintenance more difficult each time, unless specific effort is applied to stop this trend
- 2) The tendency for correction of one problem to cause others to appear
- 3) The question of how much of the program to retest after modification

*Georghiou, D. L., Guidelines for Programming in Portable Fortran, Logicon Report No. DS-R78069, Sept. 1978.

†Donahoo, J. D., A Review of Software Maintenance Technology, RADC-TR-80-13, Feb. 1980.

- 4) The tendency for program documentation not to be updated to reflect the changes made

Issues 2 and 3 are concerned with the reliability of the modified software. These issues are always addressed in the IV&V of maintenance efforts. Issue 4, concerned with continued maintainability, is also inherent in the IV&V process, through the documentation verification activity. While Issue 1 has not traditionally been included in the IV&V charter, it has been addressed informally with the reporting of poor programming practices. By directing the design and code verification activities to report all unwarranted increases in complexity, such as those caused by artificial localization of changes, the program office could focus attention on this problem and ensure that the resulting software was not only as reliable, but also as maintainable as it was before.

5.2.6 Indirect Effects on Maintainability

IV&V's evaluation of software reliability has the added effect of enhancing the maintainability of the subject program. The improved reliability of the software makes it less likely to require modification once in the operational environment, and the early detection of anomalies provided by IV&V allows time for effective redesign rather than "kluge" solutions, which can make the software overly complex, nonmodular, and difficult to understand.

5.3 Project Results

Data from the projects surveyed provided answers to the following questions:

- How many of the anomaly reports submitted by IV&V had a direct effect on software maintainability?
- What development materials did they involve?
- What types of problems did they report?
- What were their severity ratings?
- What aspects of maintainability would have been affected?
- What were the indirect effects resulting from reliability evaluation?

The following paragraphs discuss these issues.

5.3.1 Number of Anomaly Reports Affecting Maintainability

Of the 1575 anomalies reported on the IV&V projects, 854 were concerned with software maintainability. The number of these anomalies on each project was as follows:

- Project 1: 66
- Project 2: 135
- Project 3: 371
- Project 4: 97
- Project 5: 185

Many of these anomalies had maintainability as one of several effects. Such anomalies were usually reported for some reason other than maintainability and had maintainability as a secondary effect. It was instructive, therefore, to single out the anomalies that had maintainability as their only effect. There were 347 such anomalies, broken down as follows:

- Project 1: 10
- Project 2: 9
- Project 3: 211
- Project 4: 47
- Project 5: 70

This breakdown shows dramatically the results of different project objectives. Projects 1 and 2 were concerned almost solely with software reliability. They reported maintainability anomalies only when these might result in reliability problems in future program versions. Project 3 performed extensive documentation analysis aimed at detecting maintainability problems. Project 4 was discouraged from reporting documentation problems in anomaly reports. Project 5 performed some documentation analysis, but had limited emphasis on maintainability.

Of the 854 anomaly reports cited above, those that were accepted by the program office and acted on by the developer had a direct effect on software maintainability. Figures 17 and 18 indicate program office acceptance of the two types of maintainability reports. Figure 17 shows that for all maintainability reports, an average of 90% were accepted, 3% were rejected, 0% were withdrawn or superseded, and for 6% acceptance was unknown. Figure 18 shows that for anomaly reports concerned solely with maintainability, an average of 94% were accepted, 1% were rejected, 0% were withdrawn or superseded, and for 5% acceptance was unknown. Projects 3 and 4 had 100% acceptance of maintainability-only reports. The projects that were less concerned with maintainability had somewhat lower rates. Even on these projects, however, acceptance was high enough to indicate the overall validity of the findings.

Figures 19 and 20 indicate the action taken on the two types of maintainability anomalies. Figure 19 shows a pattern similar to that observed for reliability anomalies, namely, Projects 1-4 exhibiting similar profiles and Project 5 being markedly different due to the experimental nature of the development project. For Projects 1-4, an average of 80% of the anomalies were acted on, 9% were not, and 10% had resolution still open or unknown. Figure 20 shows the resolution of anomalies that had maintainability as their only effect. Projects 2, 3, and 4 show high rates of corrective action; Projects 1 and 5 show lower rates. Overall, 79% of these anomalies were acted on, 10% were not, and resolution of 11% was still open or unknown at the time of the study. Thus, while maintainability anomalies may have had a lower priority

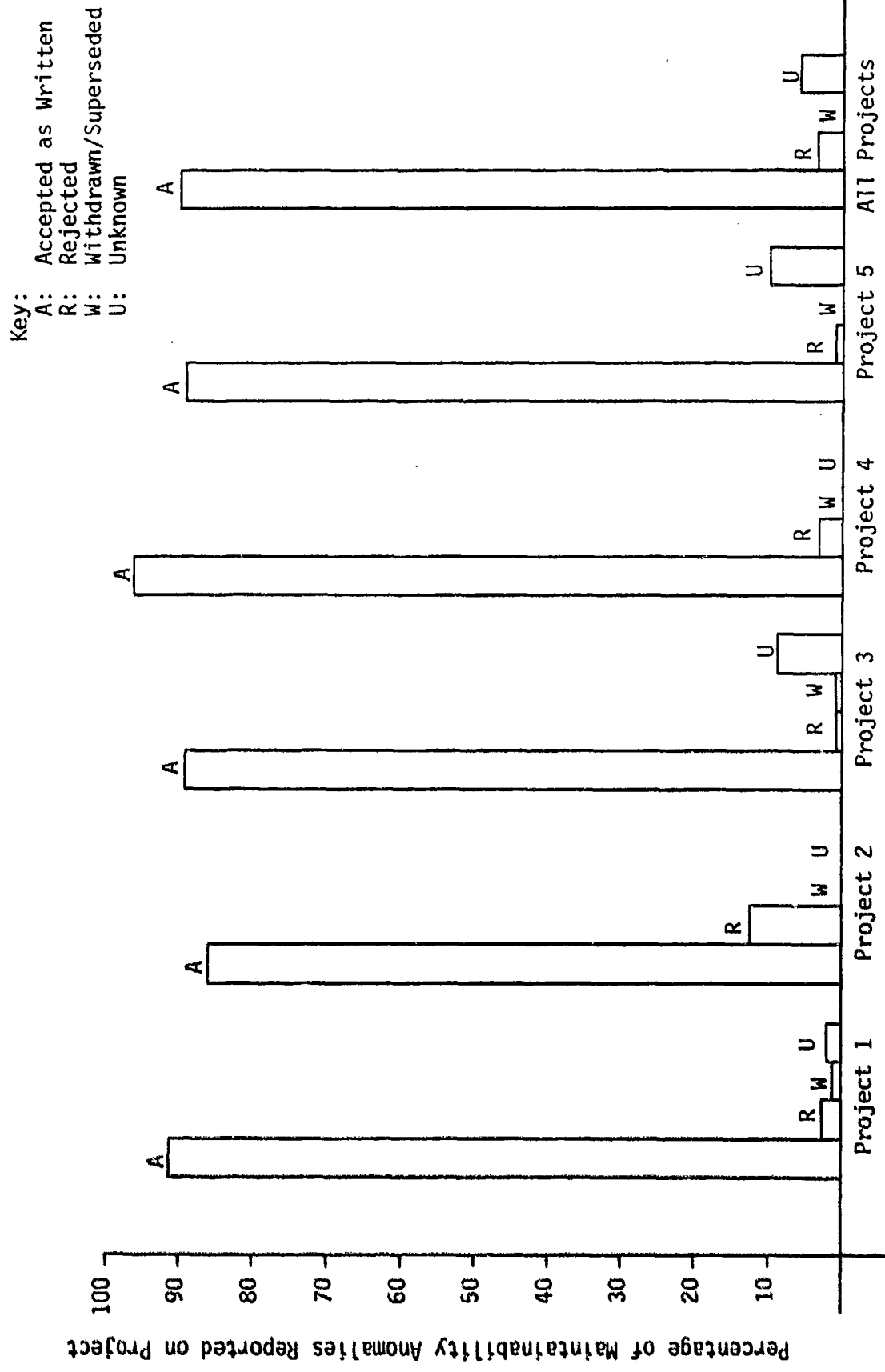


Figure 17. Acceptance of Anomaly Reports Concerned With Maintainability

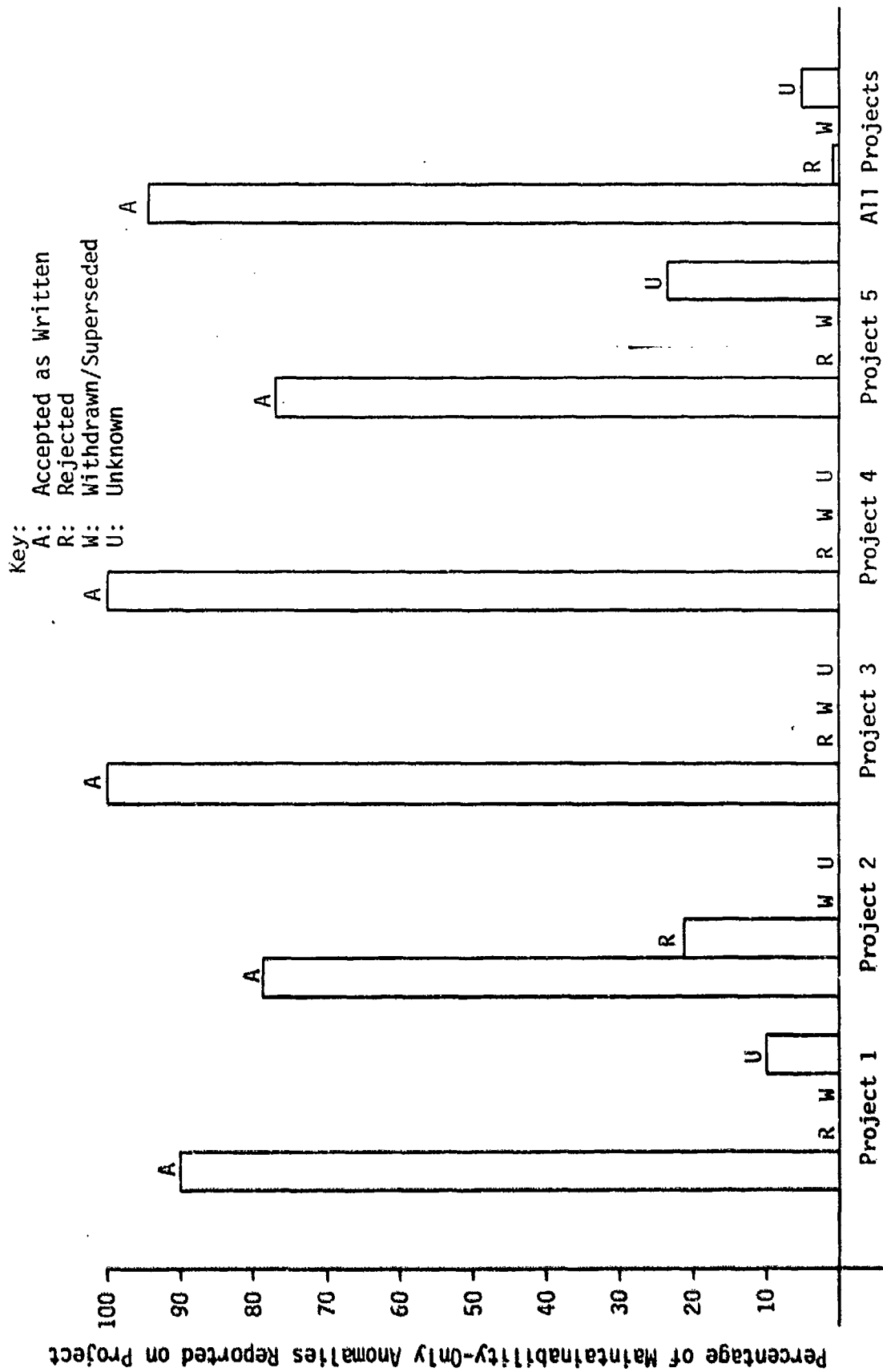


Figure 18. Acceptance of Anomaly Reports Concerned Solely With Maintainability

Key: A: Acted on
 N: Not acted on
 O/U: Resolution open/unknown

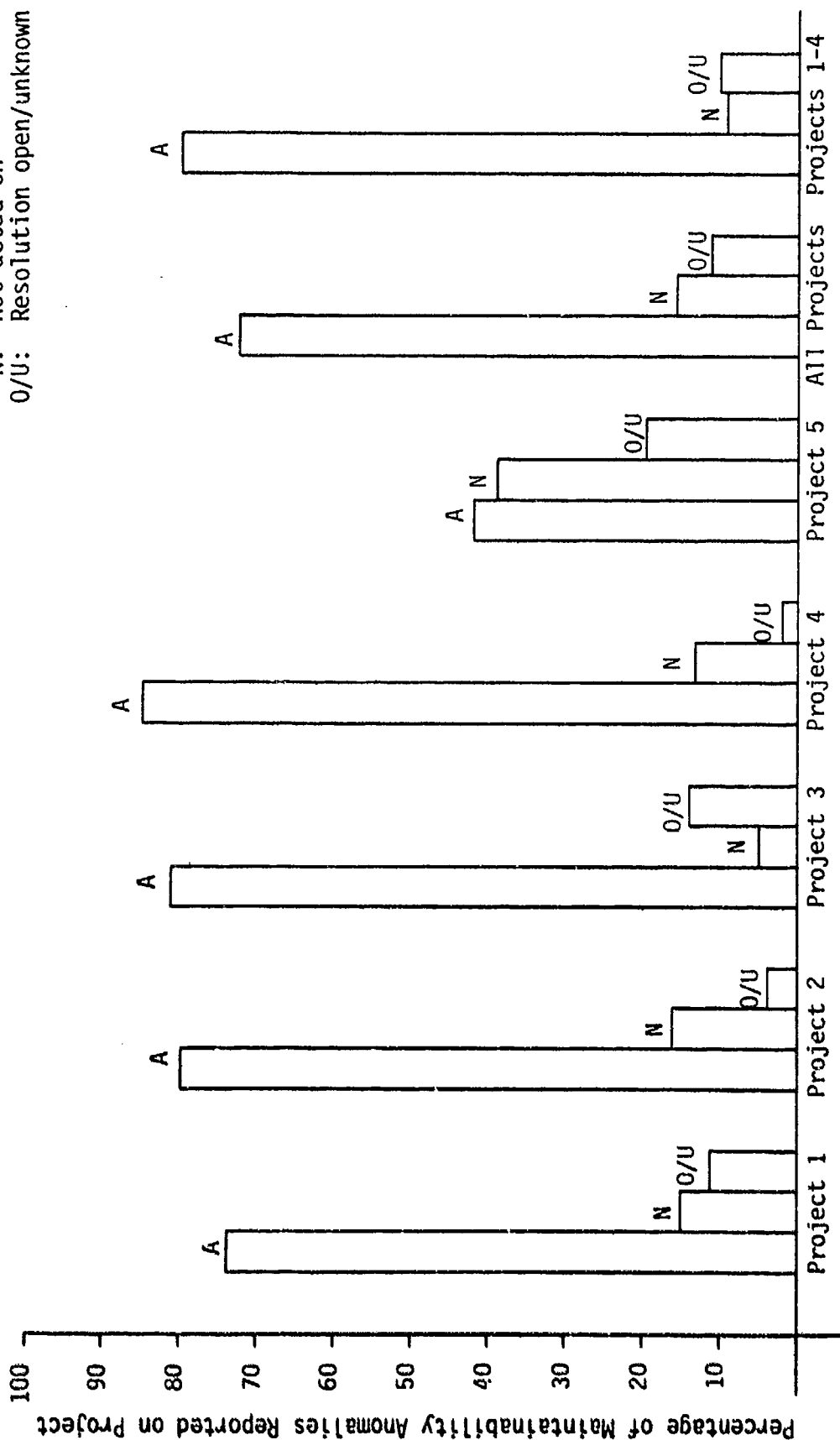


Figure 19. Resolution of Anomalies Concerned With Maintainability

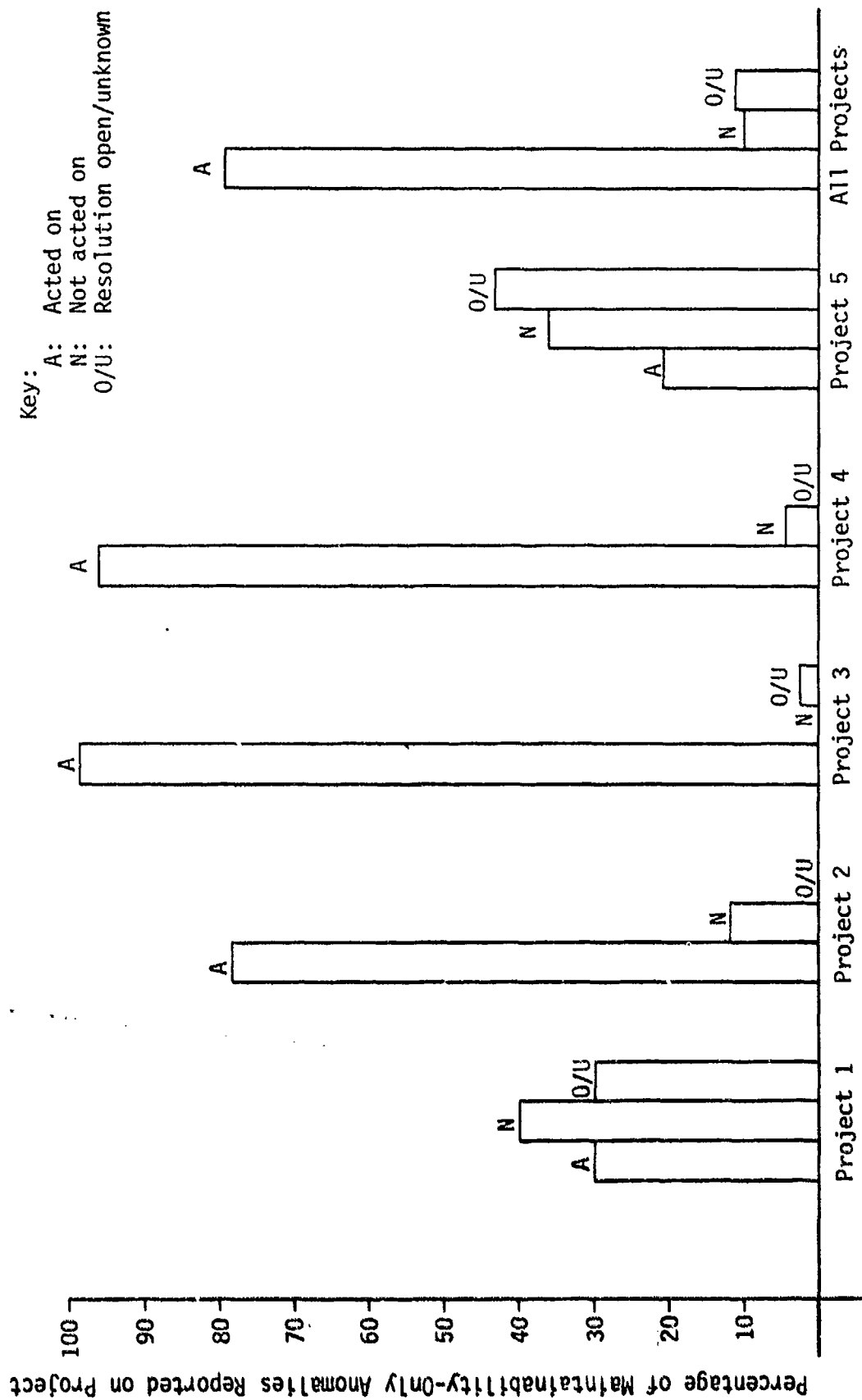


Figure 20. Resolution of Anomalies Concerned Solely With Maintainability

than those concerned with reliability, they were considered important enough on three of the projects to have a high rate of corrective action.

The total number of anomaly reports that were concerned with maintainability, accepted as valid, and acted on by the developer was 645. These reports represent the direct contribution of IV&V to the maintainability of the subject programs. The breakdown of these anomalies, hereafter referred to as "corrected maintainability anomalies," was as follows:

- Project 1: 48
- Project 2: 108
- Project 3: 330
- Project 4: 82
- Project 5: 77

Singling out the anomalies that affected maintainability only, the total was 276, broken down as follows:

- Project 1: 3
- Project 2: 7
- Project 3: 207
- Project 4: 45
- Project 5: 14

These anomalies are hereafter referred to as "corrected maintainability-only anomalies." The remainder of the analysis is concerned with these two sets of anomalies.

5.3.2 Anomaly Location

Anomalies affecting maintainability may be found in requirement specifications, before-code and after-code design specifications, code, user documentation, and other materials. Figure 21 shows the number of corrected maintainability anomalies and corrected maintainability-only anomalies found in each of these materials.

Over half of the corrected maintainability anomalies were in requirement specifications. Correction of these anomalies enhanced maintainability by making it easier to understand the functions and organization of the program and to devise new test cases. A third of the anomalies were in the before-code and after-code design specifications. Correction of these problems made it easier to grasp the program design and to determine how to make the needed changes and devise test cases. Twelve per cent of the anomalies were in the code itself. Their correction resulted in code that was traceable to requirements, more efficient, less complex, better commented, and easier to understand, modify, and retest.

5.3.3 Anomaly Categories

Table 8 indicates the number of corrected maintainability anomalies found in each anomaly category. The left-hand set of figures applies to all such

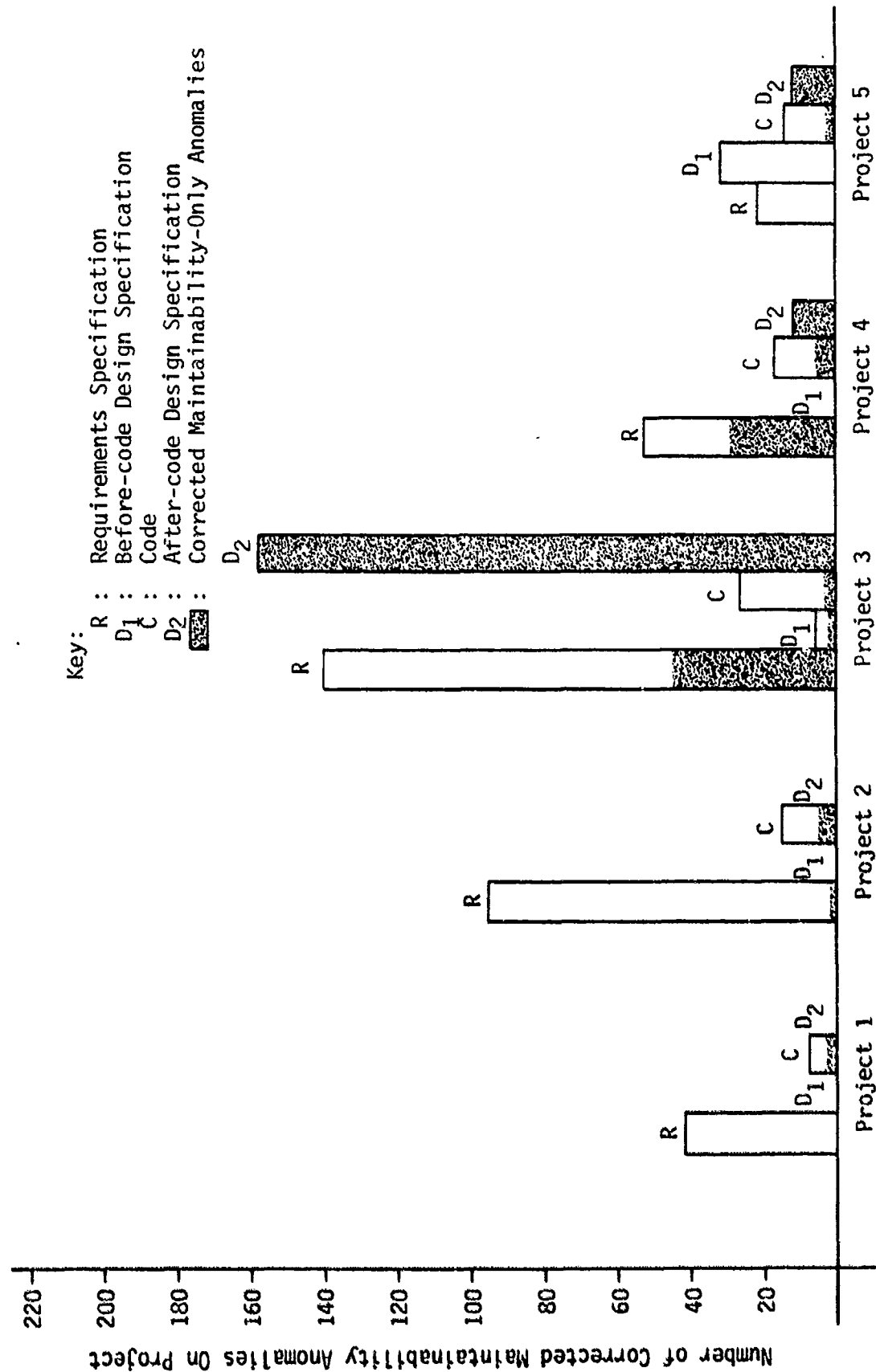


Figure 21. Development Materials In Which Corrected Maintainability Anomalies Were Found

Table 8. Number of Corrected Maintainability Anomalies in Each Category

Anomaly Category	Corrected Maintainability Anomalies						Corrected Maintainability-Only Anomalies					
	Project						Project					
	1	2	3	4	5	ATT	1	2	3	4	5	ATT
Requirement Specification Anomalies												
R1. Incorrect Requirements	14	49	66	15	1	145	--	1	18	12	--	31
R2. Inconsistent Requirements	9	9	13	7	17	55	--	--	1	--	--	1
R3. Incomplete Requirements	14	22	46	27	3	112	--	--	23	14	--	37
R4. Other Requirement Problems	4	13	11	3	--	31	--	--	2	2	--	4
R5. Presentation; Standards Compliance	--	1	4	1	--	6	--	--	1	1	--	2
Total	41	94	140	53	21	349	--	1	45	29	--	75
Before-Code Design Specification Anomalies												
D1. Requirement Compliance	--	--	4	--	1	5	--	--	--	--	--	--
D2. Choice of Algorithm, Mathematics	--	--	--	--	7	7	--	--	--	--	--	--
D3. Sequence of Operations	--	--	--	--	5	5	--	--	--	--	--	--
D4. Data Definition	--	--	--	--	1	1	--	--	--	--	--	--
D5. Data Handling	--	--	--	--	14	14	--	--	--	--	--	--
D6. Timing, Interruptibility	--	--	--	--	2	2	--	--	--	--	--	--
D7. Interfaces, I/O	--	--	--	--	--	--	--	--	--	--	--	--
D8. Other Design Problems	--	--	1	--	1	2	--	--	1	--	--	1
D9. Presentation, Standards Compliance	--	--	--	--	--	--	--	--	--	--	--	--
Total	--	--	5	--	31	36	--	--	1	--	--	1
Code Anomalies												
C1. Requirement, Design Compliance	3	1	13	8	1	26	2	--	--	--	--	2
C2. Choice of Algorithm, Mathematics	2	3	--	--	--	5	--	1	--	--	--	1
C3. Sequence of Operations	--	--	1	--	--	1	--	--	1	--	--	1
C4. Data Definition	--	3	1	3	--	7	--	2	--	--	--	2
C5. Data Handling	1	1	--	--	1	3	1	1	--	--	1	3
C6. Timing, Interruptibility	--	--	--	--	--	--	--	--	--	--	--	--
C7. Interfaces, I/O	--	--	--	1	--	1	--	--	--	--	--	--
C8. Other Code Problems	1	2	10	1	11	25	--	--	--	--	--	--
C9. Presentation, Standards Compliance	--	4	2	4	1	11	--	2	2	4	1	9
Total	7	14	27	17	14	79	3	6	3	4	2	18
After-Code Design Specification Anomalies												
P1. Incorrect Documentation	--	--	29	4	11	44	--	--	29	4	11	44
P2. Inconsistent Documentation	--	--	6	1	--	7	--	--	6	1	--	7
P3. Incomplete Documentation	--	--	57	4	1	62	--	--	57	4	1	62
P4. Other Documentation Problems	--	--	10	3	--	13	--	--	10	3	--	13
P5. Presentation, Standards Compliance	--	--	56	--	--	56	--	--	56	--	--	56
Total	--	--	158	12	12	182	--	--	158	12	12	182
User Documentation Anomalies												
Other Anomalies												
Total Anomalies in All Categories	48	108	330	82	77	645	3	7	207	45	14	276

anomalies; the right-hand set applies to the anomalies concerned with maintainability alone.

There were 349 requirement specification anomalies that would have affected maintainability if they had not been corrected. Most prevalent among these were incorrect requirements, which accounted for 42%, and incomplete requirements, which accounted for 32%. Seventy-five of these anomalies had maintainability as their only effect. These anomalies, reported only on Projects 3 and 4, concerned cases in which the requirement specifications had not been updated to reflect the program as implemented. Again, the two most prevalent categories were incorrect and incomplete requirements.

Design verification performed by Projects 3 and 5 disclosed 36 anomalies that would have affected maintainability if they had not been corrected. Most prevalent were errors in the design specification's descriptions of program data handling. Only one of the 36 anomalies had maintainability as its only effect. This anomaly concerned a questionable design feature to be incorporated in a future update.

Seventy-nine code anomalies would have affected maintainability if they had not been corrected. Most prevalent were cases in which code was not traceable to the requirements or design and cases of inefficient or extraneous code, shown under Category C8. Eighteen of the code anomalies had maintainability as their only effect. Most prevalent here were cases involving incorrect and incomplete comments and violations of development practices. Also included were several latent errors, that is, cases in which the program was coded incorrectly, but happened to work satisfactorily in the current version.

There were 182 corrected maintainability anomalies in after-code design specification, all having maintainability as their only effect. Most prevalent were instances of incomplete documentation. Problems with presentation of information and incorrect documentation were also common.

5.3.4 Anomaly Severity

Severity ratings for all five projects were based on the impact an anomaly would have on program performance. As a result, anomalies that had as their only effect decreased maintainability were almost always rated Low. The few exceptions resulted from the fact that severity ratings were assigned to entire anomaly reports, rather than to parts thereof, and maintainability anomalies occasionally received a higher severity rating by association with anomalies affecting reliability. Severity ratings for the anomalies that had maintainability as one of several effects are not meaningful here because the ratings were generally assigned on the basis of other effects.

5.3.5 Maintainability Attributes Affected

Section 5.1 identified as the four basic attributes of maintainable software understandability, modifiability, testability, and portability. It is interesting to ask which of these attributes were affected by the detection and correction of maintainability anomalies on the five IV&V projects.

The four basic attributes are not mutually exclusive. As described in Section 5.2, modifiability consists of understandability plus expandability; testability consists of understandability plus instrumentation. Any anomaly report that contributes to understandability therefore contributes to modifiability and testability as well.

To determine more precisely the effects of the maintainability anomalies, each one was examined in terms of its effect on understandability, expandability, instrumentation, and portability. These traits are mutually exclusive and therefore more readily analyzed.

The answer in almost all cases was understandability. Maintainability anomalies were almost universally concerned with:

- Documentation that did not describe program requirements or design completely or accurately
- Code characteristics that would make it difficult for a maintenance programmer to grasp program logic or to trace the logic to requirements or design

Only 10 anomalies had an effect other than understandability. These anomalies, concerned with poor programming practices or latent errors, affected expandability. They would make changes more difficult than necessary or make it possible for program changes to have unexpected side effects.

The predominant effect of IV&V on software maintainability was therefore increased program understandability, and therefore enhanced modifiability and testability as well. The analysis showed that IV&V can be an effective tool for increasing software maintainability.

5.3.6 Indirect Effects of Reliability Evaluation

Section 4 described 748 reliability anomalies corrected as a result of IV&V. Correction of these anomalies improved not only program reliability, but also program maintainability. By detecting the 748 problems before the software went into the operational environment, IV&V helped to prevent the need for corrective maintenance of the software. By detecting 447 of the problems before development testing, IV&V allowed the developer extra time for anomaly correction, reducing the possibility of poorly designed corrections that could hinder maintenance efforts.

The four basic attributes are not mutually exclusive. As described in Section 5.2, modifiability consists of understandability plus expandability; testability consists of understandability plus instrumentation. Any anomaly report that contributes to understandability therefore contributes to modifiability and testability as well.

To determine more precisely the effects of the maintainability anomalies, each one was examined in terms of its effect on understandability, expandability, instrumentation, and portability. These traits are mutually exclusive and therefore more readily analyzed.

The answer in almost all cases was understandability. Maintainability anomalies were almost universally concerned with:

- Documentation that did not describe program requirements or design completely or accurately
- Code characteristics that would make it difficult for a maintenance programmer to grasp program logic or to trace the logic to requirements or design

Only 10 anomalies had an effect other than understandability. These anomalies, concerned with poor programming practices or latent errors, affected expandability. They would make changes more difficult than necessary or make it possible for program changes to have unexpected side effects.

The predominant effect of IV&V on software maintainability was therefore increased program understandability, and therefore enhanced modifiability and testability as well. The analysis showed that IV&V can be an effective tool for increasing software maintainability.

5.3.6 Indirect Effects of Reliability Evaluation

Section 4 described 748 reliability anomalies corrected as a result of IV&V. Correction of these anomalies improved not only program reliability, but also program maintainability. By detecting the 748 problems before the software went into the operational environment, IV&V helped to prevent the need for corrective maintenance of the software. By detecting 447 of the problems before development testing, IV&V allowed the developer extra time for anomaly correction, reducing the possibility of poorly designed corrections that could hinder maintenance efforts.

The four basic attributes are not mutually exclusive. As described in Section 5.2, modifiability consists of understandability plus expandability; testability consists of understandability plus instrumentation. Any anomaly report that contributes to understandability therefore contributes to modifiability and testability as well.

To determine more precisely the effects of the maintainability anomalies, each one was examined in terms of its effect on understandability, expandability, instrumentation, and portability. These traits are mutually exclusive and therefore more readily analyzed.

The answer in almost all cases was understandability. Maintainability anomalies were almost universally concerned with:

- Documentation that did not describe program requirements or design completely or accurately
- Code characteristics that would make it difficult for a maintenance programmer to grasp program logic or to trace the logic to requirements or design

Only 10 anomalies had an effect other than understandability. These anomalies, concerned with poor programming practices or latent errors, affected expandability. They would make changes more difficult than necessary or make it possible for program changes to have unexpected side effects.

The predominant effect of IV&V on software maintainability was therefore increased program understandability, and therefore enhanced modifiability and testability as well. The analysis showed that IV&V can be an effective tool for increasing software maintainability.

5.3.6 Indirect Effects of Reliability Evaluation

Section 4 described 748 reliability anomalies corrected as a result of IV&V. Correction of these anomalies improved not only program reliability, but also program maintainability. By detecting the 748 problems before the software went into the operational environment, IV&V helped to prevent the need for corrective maintenance of the software. By detecting 447 of the problems before development testing, IV&V allowed the developer extra time for anomaly correction, reducing the possibility of poorly designed corrections that could hinder maintenance efforts.

- Hypothesize for the selected factors which ones would be affected positively and which negatively by IV&V
- Tie in study results to test the hypotheses with actual data and to quantify results where possible

The results of the first three steps are given in Table 9. The following paragraphs discuss these results.

6.1 Factors Affecting Development Cost/Productivity

Table 9 identifies 125 cost/productivity factors described in the literature. They have been divided into 11 basic categories:

- The nature of the software to be developed
- Special requirements imposed on the software
- The quality and stability of the requirements
- The quality and stability of the design
- The quality and stability of the code
- Personnel and organization
- Development methodology used
- Development facilities available
- Project management
- Amount of nonproductive activity performed
- Cost factors unrelated to productivity

The left-most column identifies one or more references in which each factor was found. When more than two references cited a given factor, only the first few encountered were included in the table.

6.2 Factors Affected by IV&V

The right-hand columns of Table 9 present the study's hypotheses as to the potential effect of IV&V on each of the cost/productivity factors. The possible responses are:

- "Positive," meaning that IV&V has the potential to increase programmer productivity or decrease cost with respect to the factor
- "Negative," meaning that IV&V has the potential to decrease programmer productivity or increase cost with respect to the factor
- "None," meaning that little or no effect could be envisioned under normal circumstances

The most striking finding is that for 90 of the 125 factors, IV&V was considered to have no effect at all. This finding shows that the benefits of IV&V can be obtained without placing a significant amount of overhead on the development process. Ruled out were all factors under "the nature of the software to be developed" and most factors under "special requirements imposed on the software," "personnel and organization," "development facilities," and "costs

Table 9. Factors Affecting Development Cost/Productivity

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
A. <u>The Nature of the Software to be Developed</u>				
23, 24	A-1. Type of application			X
23, 25	A-2. Degree of innovation required			X
25, 26	A-3. Size of the software			X
8, 27	A-4. Data base size/complexity			X
25	A-5. Number and complexity of I/O formats			X
23	A-6. Data management techniques to be used			X
23	A-7. Multiple-site installation			X
15, 25	A-8. Extent of decentralization and number of interfaces			X
23, 27	A-9. Real-time requirements			X
28	A-10. Reimplementation of existing software			X
25	A-11. Number of other components and sub-systems being developed concurrently as part of the system			X
B. <u>Special Requirements Imposed on the Software</u>				
23, 26	B-1. Quality requirements			
	• Reliability requirements			X
	• Maintainability requirements			X
	• Efficiency requirements			X
	• Integrity requirements			X
	• Usability requirements			X
	• Testability requirements			X
	• Portability requirements			X
	• Reusability requirements			X
	• Interoperability requirements			X
	• Transportability requirements			X
23, 25	B-2. Requirements for special displays and interfaces with special equipment			X
23	B-3. Testing requirements			X
8, 27	B-4. Documentation requirements		X	
27	B-5. Percentage of code to be delivered			X

Table 9. Factors Affecting Development Cost/Productivity (continued)

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
25, 27	B-6. Overall constraints on program design			X
23, 27	B-7. Constraints on program size			X
23, 27	B-8. Constraints on program speed			X
8	B-9. Requirement to install the system at a site other than the development site			X
C. <u>Quality and Stability of the Requirements</u>				
27	C-1. Customer experience:			
	o With the application			X
	o With data processing			X
23, 27	C-2. User participation in requirements definition			X
25, 27	C-3. Programmer participation in requirements definition			X
23	C-4. Effectiveness of communication among user, customer, developer, maintainer	X		
8, 23, 26, 29	C-5. Completeness, accuracy, and clarity of requirement specifications	X		
23, 30	C-6. Level of change in requirements during development	X		X
23, 30	C-7. Phase in which requirement changes occur	X		
D. <u>Quality and Stability of the Design</u>				
23	D-1. Accuracy of translation from requirements to design	X		
8, 23	D-2. Quality of the design	X		
15, 23	D-3. Amount of changes to the design	X		
8, 27, 30	D-4. Timing of design changes	X		

Table 9. Factors Affecting Development Cost/Productivity (continued)

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
E. <u>Quality and Stability of the Code</u>				
23	E-1. Accuracy of translation from design to code	X		
8, 23	E-2. Quality of the coded program	X		
15, 23	E-3. Amount of code changes needed	X		
8, 27, 30	E-4. Timing of code changes	X		
F. <u>Personnel and Organization</u>				
	F-1. Development team experience			
27, 28, 31	● With similar applications			X
23, 27	● With the computer hardware			X
27	● With the language to be used			X
15, 26	F-2. Programmer ability			X
25, 27	F-3. Programmer participation in requirements definition			X
25, 26, 29	F-4. Amount of training required			X
28, 29	F-5. Development team organization and size			X
8, 25	F-6. Development team stability			X
31	F-7. Development team morale			X
29	F-8. Development team cooperation			X
15	F-9. Development team objectives	X		
26	F-10. Appropriateness of man-loading			X
31	F-11. Availability of personnel when needed			X
23, 26	F-12. Percentage of support personnel		X	
G. <u>Development Methodology</u>				
23, 24, 26	G-1. Programming language used	X		
23, 26	G-2. Use of modern programming practices			

Table 9. Factors Affecting Development Cost/Productivity (continued)

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
27, 32	• Top-down development	X		
30	• Program design language	X		
23	• HIPO diagrams	X		
15, 27	• Structured programming	X		
18, 23	• Programming support library			X
23, 27	• Chief programmer team			X
27, 33	• Design and code inspection			X
33	• Unit development folders			X
	G-3. Compliance with well-defined standards	X		
8	G-4. Quality assurance practices followed			X
33, 34	G-5. Configuration management of the software and documentation	X		
15, 33	G-6. Quality of test plan			X
23	G-7. Avoidance of hands-on batch testing			X
31	G-8. Debugging style			X
8, 25	G-9. Error reporting and correcting procedures	X	X	
<u>H. Development Facilities</u>				
	H-1. Availability of computer hardware			
23	• Late selection of target computer			X
15, 23	• Concurrent development of hardware and software			X
25	H-2. Suitability of target computer			
23	• Sufficient speed			X
15, 23	• Sufficient memory size			X
23	H-3. Development and target computers differ			X
25, 27	H-4. Ease of access to computer facilities			
23	• Use of operational site for development			X
23	• Use of another organization's development facilities			X
25	• Need to share computing facilities		X	
8	• Proximity of computing facilities			X

Table 9. Factors Affecting Development Cost/Productivity (continued)

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
8	● Proximity of computing facilities			X
	H-5. Complexity of computer facilities			
23	● Number of different development locations			X
25	● Number of different computers			X
	H-6. Computer response time			
15, 28	● Mode of operation: batch vs. on-line			X
8	● Computer throughput rate			X
25	● Time lost to maintenance			X
25	● Probability of computer overload			X
29, 31	H-7. Computer system reliability			X
25, 29	H-8. Computer system usability			X
	H-9. Support software and development tools:			
23, 25	● Availability when needed			X
29, 31	● Quality			X
29	H-10. Adequacy of technical reference materials			X
29	H-11. Suitability, comfort of work environment			X
25, 29	H-12. Cooperation and responsiveness of support services			X
27	H-13. Classified security environment		X	
25	H-14. Availability of data for the data base			X
	<u>I. Amount of Non-Productive Activity Performed</u>			
15, 25	I-1. Travel			X
	I-2. Meetings, interfaces			
24, 26	● Internal			X
27	● With customer		X	
25	● With other agencies		X	

Table 9. Factors Affecting Development Cost/Productivity (continued)

References	Factor	Potential Effects of IV&V		
		Positive	Negative	None
30	I-3. Documentation preparation		X	
8, 22	I-4. Training			X
24	I-5. Company business			X
22, 24	I-6. Paperwork		X	
24, 25	I-7. Delays for needed materials, components, concurrence, etc.			X
30, 8, 6	I-8. False starts; need for development	X		
30, 35, 36	I-9. Software defect removal	X		
	<u>J. Project Management</u>			
8	J-1. Type of contract			X
26, 29	J-2. Feasibility of schedule			X
8, 26	J-3. Allocation of resources to each phase			X
23, 26	J-4. Completion of activities within their allotted phase	X		
23, 34	J-5. Effectiveness of cost monitoring			X
24, 29, 31	J-6. Effectiveness of progress monitoring	X		
24, 29	J-7. Effectiveness of personnel management			X
8, 34	J-8. Adequacy of formal reviews and audits	X		
	<u>K. Costs Unrelated to Productivity</u>			
	K-1. Computer hardware			X
23, 25	K-2. Secondary resources (computer time, documentation reproduction, travel, etc.)		X	
25	K-3. Equipment, office space			X
25	K-4. Simulation and test facilities			X
25	K-5. Special security-related equipment			X

unrelated to productivity." For these factors, the presence of an IV&V agency either made no difference at all, or could make a difference only in unlikely circumstances.

Of the remaining factors, IV&V was considered to have a potentially positive effect on 27 and a potentially negative effect on 9. On one factor--error reporting and correcting procedures--both a positive and negative effect could be foreseen. The following paragraphs discuss these effects.

6.2.1 Positive Effects of IV&V

The study hypothesized a positive effect on 27 of the cost/productivity factors. These factors and IV&V's effects on them are discussed below. Related topics have been discussed together for brevity.

6.2.1.1 Quality and Stability of Requirements: In Reference 8, Finfer states that the stability and quality of requirement specifications may be the key factor in programmer productivity. Supporting this position is Doty's software estimation guide (Reference 23), which states that:

- Vague operational requirements can be expected to decrease productivity by 35% on command and control applications and 50% on scientific applications.
- The effects of changing requirements can be as high as a 95% decrease in programmer productivity.

Equally dramatic are figures cited by Wolverton (Reference 30), which indicate that a requirement defect not corrected during the requirements definition phase is:

- 2-1/2 times more costly to correct during design
- 5 times more costly to correct during coding and checkout
- 36 times more costly to correct during test and integration

Improving the quality of requirement specifications is one of the key objectives of IV&V. Requirements verification focuses on the clarity, completeness, correctness, and consistency of requirements. It can point out omissions; identify unfeasible, questionable, and ambiguous requirements; detect errors; point out inconsistencies; and identify problems in the way that the requirements have been documented. The increased visibility provided by this analysis can improve communication among the user, customer, and developer; help the user to define his requirements more precisely so that later changes will be unnecessary; and result in vastly improved requirement specifications. By performing this verification in parallel with the requirements definition process, IV&V can ensure that requirement changes are made early, preventing the major cost impacts associated with later changes.

6.2.1.2 Quality and Stability of the Design: According to Doty (Reference 23):

- 60% of all errors discovered in testing are caused by faulty design.

- These errors can result in cost increases of up to 100%.

Emphasizing the importance of early detection, Wolverton (Reference 30) cites figures stating that a design change costs, on the average, \$977 to correct during code and checkout and \$7136 during test and integration (1975 figures).

The design verification performed as part of IV&V can have a major impact on design quality and stability. It ensures that all requirements have been properly translated into design and that all functions included in the design are traceable to approved requirements. It evaluates the choice of algorithms, the design logic, the data definitions, and all aspects of the design. By performing this verification during the design phase, IV&V can detect design errors before they are implemented in code and improve the quality of the design materials used as input to the coding phase.

6.2.1.3 Quality and Stability of the Code: The coding process almost always involves some detailed design beyond that specified in the design materials. As a result, new faults can be introduced at this stage. Figures cited by Finfer (Reference 8) indicate that coding errors not detected until the testing stage can be from 2 to 5 times as costly to correct as those detected during coding and checkout. Once again, early detection decreases cost.

The code verification activity of IV&V is specifically designed to detect anomalies in preliminary versions of code. It relies upon inspection rather than program execution to ensure agreement between the design and code and to check for faults in logic, data definition, data usage, interfacing, and so on. By detecting such problems before the program has been integrated for testing, IV&V can decrease the cost of error correction.

This approach is substantiated by Shooman and Bolsky (Reference 6). In a study of error detection methods, they found that a large percentage of faults can be found by code inspection alone and that this method is cheaper by a ratio of 25 to 1 than techniques involving machine testing.

6.2.1.4 Development Team Objectives: In Reference 37,* Weinberg reports that the objectives of a software development team exert a significant influence on programmer productivity. In his experiments, development teams were given the same program specification but were assigned different criteria for success, namely, speed of program execution for one team, speed of program development for the other. The results showed that the different objectives produced markedly different results.

The application of this principle to IV&V lies in the program office's ability to foster an attitude of cooperative competition between the developer and the IV&V agency. If the developer knows that late deliveries, incompleteness, inconsistency, and incorrectness are going to be reported by the IV&V agency, he will tend to be more careful to meet deadlines and more thorough and careful in his work, resulting in ultimate productivity gains.

*Weinberg, G. M., "The Psychology of Improved Programming Performance," Data-mation, Nov. 1972.

6.2.1.5 Programming Language, Development Practices, and Configuration Management Procedures Used: The programming language and development and configuration management practices used on a project have been shown to have a significant effect on programmer productivity. In Reference 23, Doty states that development of a program in assembly language rather than in higher order language can decrease programmer productivity by 75% and that use of modern programming practices can result in a 67% increase in productivity for large programs.

The inspection methods used by IV&V permit effective detection of development standard violations. These may include questionable use of assembly language, incorrect or ineffective use of modern programming practices, violations of accepted development practices, inadequate configuration control procedures, and so on. The IV&V agency can alert the program office to these problems so that adjustments can be made and the full productivity potential of these methods can be achieved.

6.2.1.6 Error Correction Procedures, False Starts, and Defect Removal: In Reference 30, Wolverton cites studies indicating that:

- One of the major factors affecting productivity is false starts.
- Two-thirds to four-fifths of programmer time is spent in eliminating faults that were introduced earlier.

Supporting these studies are the findings of Miyamoto (Reference 36) which indicate that on the average, 16.56 days were needed to correct each fault in a software system under study.

IV&V can have a major impact on both false starts and defect removal time. False starts can be reduced by the improvement in requirements and design resulting from IV&V. Defect removal can be decreased through a combination of three factors:

- Improved requirements and design mean that fewer defects are introduced into the code in the first place.
- Unlike development testing, which identifies the effects of a program failure but not the program fault causing it, IV&V identifies the specific problem in the code, thereby reducing the "find-and-fix" cycle of defect removal.
- Early detection decreases the time and effort required to correct problems because faults are found before the modules have been integrated and tested.

6.2.1.7 Schedule Compliance and Progress Monitoring: In Reference 24, Brooks states that schedule overruns result not from major calamities, but from the cumulative effects of day-to-day slippage. To prevent such problems, it is crucial that all of the activities and milestones planned for each development phase be completed in that phase and not be allowed to extend into the next. Completion of activities and milestones, however, can be

difficult to assess. Timely deliveries of products that are inaccurate or incomplete may provide the program office with a false picture of overall project status. The schedule impacts of such deliveries may not surface until late in the development process.

The IV&V agency's technical evaluation yields, as a side effect, a continual assessment of the overall quality and status of the development project. Late deliveries, inadequate materials, failure of promised access to development facilities, and other signals can indicate to the IV&V agency that the development may not be on schedule. The IV&V agency can then alert the program office to the potential problem so that corrective measures can be taken.

6.2.1.8 Adequacy of Formal Reviews and Audits: Formal reviews and audits are the decision points of the development process. They provide the program office with the opportunity to review the developer's progress and to approve or disapprove development products before the next development or life cycle phase begins. IV&V can make a significant contribution to these reviews and audits by providing the program office with an independent evaluation of the materials being reviewed and of the review or audit itself. This evaluation can help the program office to determine whether the development is proceeding satisfactorily or changes need to be made. Informed decisions on these issues can prevent costly schedule overruns.

6.2.2 Negative Effects of IV&V

Of the 125 cost/productivity factors identified in Table 9, IV&V has a potentially negative effect on 9. The following paragraphs discuss these factors and IV&V's effect on them.

6.2.2.1 Secondary Resource Expenditures: The developer is required to supply the IV&V agency with copies of specifications, specification change pages, machine-readable source code, computer listings, trade study reports, and so on. Generation of these deliverables requires computer time, use of document reproduction facilities, paper, and other resources. These requirements impose overhead costs on the development effort.

6.2.2.2 Documentation Requirements: In order to perform requirements verification and design verification in parallel with the requirement and design phases, the IV&V agency requires preliminary deliveries of requirement and design materials. To provide these materials, the developer may be required to deliver documents that would not otherwise be put into deliverable form. These added documentation requirements, while having possible long-range productivity benefits, take time away from the development effort.

6.2.2.3 Need To Share Computing Facilities: On some projects, the IV&V agency conducts some or all of its testing on the same computer facilities used by the developer. These facilities may be government furnished equipment provided for the use of both contractors, a system test bed, the operational site, or, in rare cases, the developer's own facilities. In some cases, the sharing may involve support software, test software, and test equipment as well.

The requirement to share these facilities with the IV&V agency can make computer facilities less available for development testing, increase turn-around time, and complicate the computer scheduling process. The scope of these effects can be controlled, but they can decrease development productivity.

6.2.2.4 Classified Security Environment: Providing the IV&V agency with development materials and sharing computer facilities with IV&V analysts become more complex when the program being evaluated is classified. Transfer of materials from one agency to the other involves formal security procedures involving not only the development and IV&V teams but security personnel as well. Computer scheduling must take into account not only the amount of time needed by each agency, but the types of jobs that can be executed concurrently. A classified security environment always affects programmer productivity to some extent; the presence of the IV&V agency adds one further complication.

6.2.2.5 Error Reporting and Correcting Procedures: "Error reporting and correcting procedures" was the one cost/productivity factor considered to be affected both positively and negatively by IV&V. The positive aspects were addressed in Section 6.2.1.6. On the negative side is the time required for anomaly report processing. Each report must be studied and understood. A decision must be made as to its validity and as to the cost effectiveness of various alternatives for action. These recommendations must then be conveyed to the program office, a process that may involve discussions including the IV&V agency.

If the anomaly report concerns a problem that would have surfaced during program development or operation, development time is saved by having it pointed out in an anomaly report. If the report is incorrect or out of the scope of the development project, anomaly report processing time clearly outweighs any benefits gained. If the report concerns a maintainability problem that would not arise in the development process, its processing would tend to decrease development productivity but have a positive effect on life cycle cost. The balance between positive and negative effects, therefore, is complex. Section 6.3.7 examines this question quantitatively, using results from the IV&V projects.

6.2.2.6 Customer Interface. Walston and Felix (Reference 27) identify "customer interface complexity" as a factor in programmer productivity. They report a 42% decrease in productivity when this interface is more complex than normal. Though not explained in the reference, it is assumed that this complexity is determined by such factors as the degree of customer involvement in the development process and the amount of time spent in interfacing with the customer.

IV&V can increase the complexity of the customer interface by giving the customer more visibility into the development process. The developer may be required to report on the status of anomaly reports, present arguments regarding the validity of anomaly reports, respond to the customer regarding IV&V requests for deliverables, respond to the customer regarding IV&V comments about the development process, arrange for time on shared computer facilities, etc. In the long run, a better informed customer enhances

productivity; in the short run, however, these added demands take time from the development process.

6.2.2.7 Interfaces With Other Agencies: Nanus and Farr (Reference 25) state that the number of agencies with which the developer must deal and their level of experience with system development have an impact on programmer productivity. The need to interface with the IV&V agency falls within this category.

The interface with the IV&V agency may include discussions of anomaly report validity; discussions of anomaly resolution; technical interchanges about deliverables, equipment, and other aspects of the development or IV&V efforts; and so on. While such interfaces are generally kept to a minimum to ensure the independence in outlook of the IV&V agency, they do require time of development team members.

6.2.2.8 Paperwork: Frederick Brooks (Reference 24) reports that 50% of a programmer's time may be devoted to activities other than programming. Machine down-time, unrelated assignments, meetings, sickness, personal time, training, paperwork, and so on account for the rest. IV&V's effects on meeting time were discussed in the last two paragraphs. IV&V may also increase the paperwork required from the developer. The increased paperwork may include written responses to anomaly reports, reports and logs concerning anomaly resolution, and other records required by the program office. All of these demands take time away from development tasks.

6.2.2.9 Support Personnel Required: In Reference 23, Doty reports that each 10% increase in support personnel relative to programmers and analysts results in a 25% decrease in productivity. This figure is based on an expected mix of 20% support personnel to 80% programmers/analysts. If requirements for deliverables, computer operators, and other services imposed by the presence of an IV&V contractor require the addition of support personnel, this factor would come into play in reducing productivity.

6.3 Project Results

Not all of the hypotheses set forth in Section 6.2 could be evaluated from the data collected. The data that were available, however, together with results from the literature, provided answers to the following questions:

- How did IV&V cost compare with development cost on the projects surveyed?
- To what extent did IV&V contribute to:
 - The quality and stability of requirements?
 - The quality and stability of the design?
 - The quality and stability of the code?
 - Compliance with development standards and configuration management procedures?
- What was the cost benefit of early error detection?

- What were the positive and negative effects of anomaly report processing on programmer productivity?

The following paragraphs discuss these results.

6.3.1 Comparison of IV&V Cost to Development Cost

Figure 22 compares IV&V cost with software development cost and with total acquisition cost. Discounting Project 5, for which cost figures were not available, IV&V project costs averaged 25% of development costs and 20% of total acquisition costs. These would appear to be good rules of thumb for IV&V cost estimation.

6.3.2 The Effect of IV&V on Requirement Quality and Stability

Of the IV&V projects surveyed, only one--Project 3--performed a standard requirements verification. Projects 1, 2, and 5 reported a number of requirement anomalies (118 for Project 2), but the phase in which they were reported was past the time at which significant cost/productivity benefits would be realized. Project 4 reported most of its findings in the weekly walk-throughs performed for the development project, making the effects of its requirements verification unavailable for the study.

Project 3 reported 96 requirement anomalies in the pre-code phases of the project, that is, early enough to make a difference in the quality and stability of the requirements. Sixty-seven more requirement problems were reported after code release.

The breakdown of the 96 requirement anomalies was as follows:

- Incorrect requirements: 38
- Inconsistent requirements: 14
- Incomplete requirements: 28
- Other content problems: 13
- Presentation, standards problems: 3

The effect of these anomaly reports was to clarify the program requirements, improve the quality of the requirement specifications used as input to the design process, prevent false starts resulting from inadequate understanding of the problem to be solved, and decrease the number and magnitude of requirement changes needed during subsequent development phases.

6.3.3 The Effect of IV&V on Design Quality and Stability

Of the IV&V projects surveyed, only Project 5 conducted a standard design verification. This analysis resulted in 75 anomaly reports, falling into the following categories:

- Requirement compliance: 1
- Choice of algorithm, mathematics: 12
- Sequence of operations: 9
- Data definition: 22

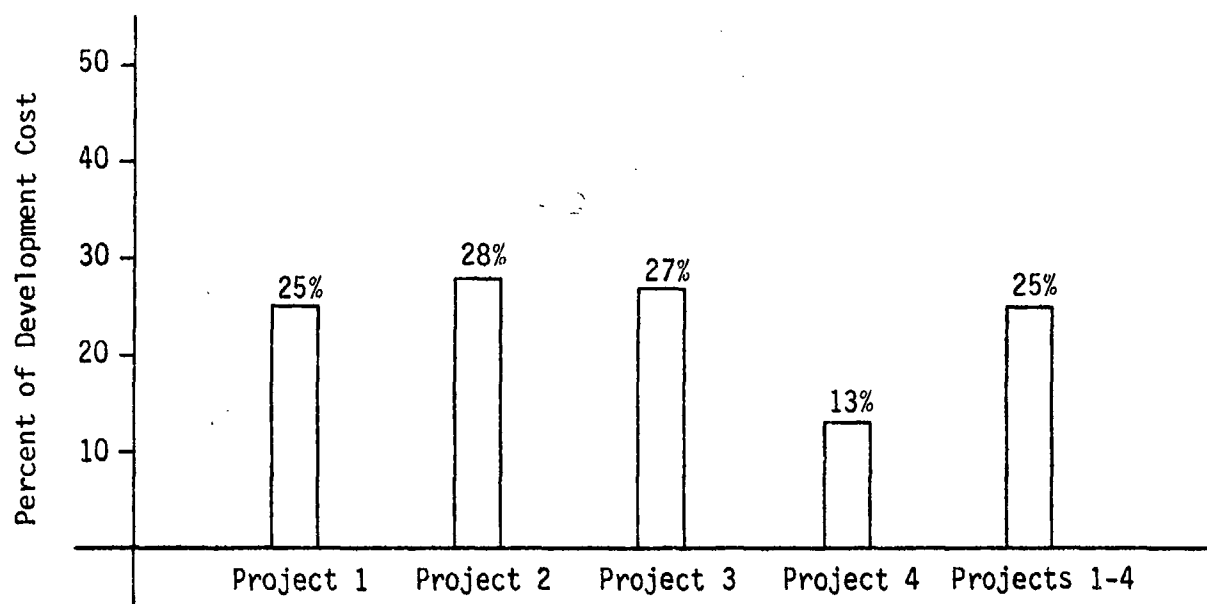


Figure 22a. IV&V Cost as a Percentage of Development Cost

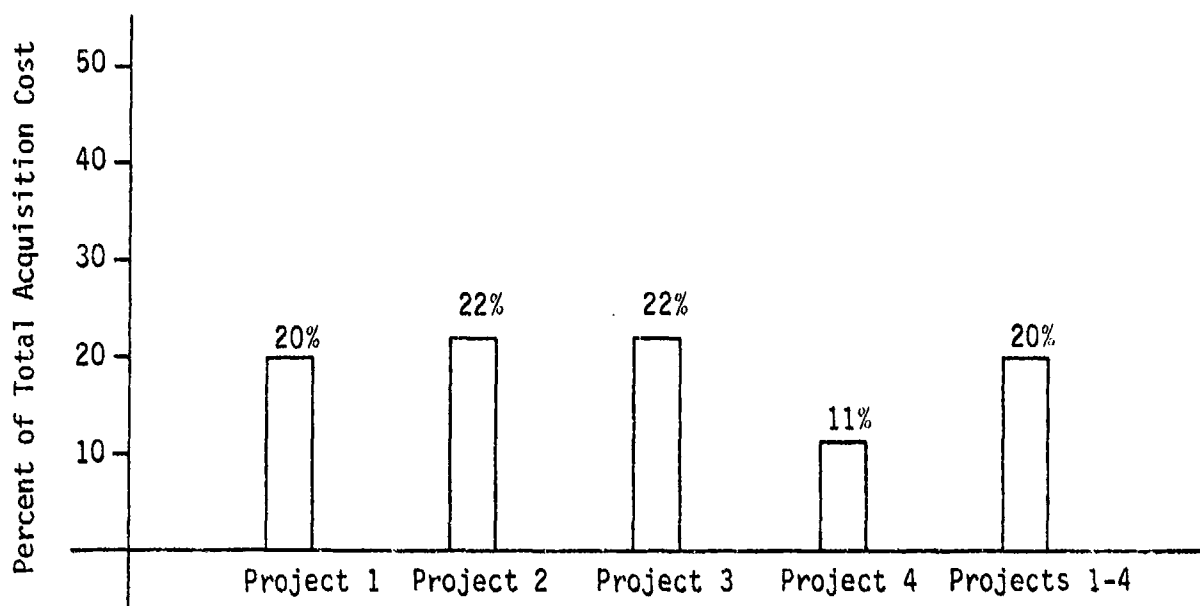


Figure 22b. IV&V Cost as a Percentage of Total Acquisition Cost

- Data handling: 23
- Interfaces, I/O: 8

The effect of these anomaly reports was to improve the quality of the design materials used for coding, allow design problems to be corrected before they were propagated into the code, decrease the number and magnitude of design changes needed once coding was under way, and decrease the time spent in false starts and defect removal during the coding and testing phases.

6.3.4 The Effect of IV&V on Code Quality and Stability

The 5 IV&V projects reported 329 code anomalies before program testing was under way, that is, early enough to have significant cost/productivity benefits. The number of such anomalies on each project was as follows:

- Project 1: 40
- Project 2: 66
- Project 3: 58
- Project 4: 83
- Project 5: 82

These anomalies fell into the following categories:

Category	Project					All
	1	2	3	4	5	
Requirement/design compliance	1	-	17	6	1	25
Choice of algorithm/mathematics	17	25	5	17	15	79
Sequence of operations	3	3	10	3	14	33
Data definition	3	10	1	17	5	36
Data handling	9	17	9	22	21	78
Timing, interruptibility	-	1	-	-	-	1
Interfaces, I/O	2	5	5	10	4	26
Other content problems	3	1	9	2	13	28
Presentation, standards compliance	2	4	2	6	9	23

The effect of these anomaly reports was to improve the quality of the program submitted for testing, allow for correction of coding problems before integration of modules made this process more difficult, decrease the number and size of program changes needed during testing, and decrease the time devoted to defect removal during the testing phase.

6.3.5 The Effect of IV&V on Standards Compliance

A total of 55 anomaly reports on the 5 projects concerned compliance with development standards and configuration management (CM) procedures. The breakdown of these reports by project was as follows:

<u>Project</u>	<u>Anomalies Concerning</u>	
	<u>Standards</u>	<u>CM</u>
1	2	-
2	8	-
3	31	3
4	1	-
5	9	1

The standards-compliance anomaly reports were concerned with violations of specified standards and violations of well-accepted programming practices. These anomaly reports alerted both the developer and the program office to potential problems in the production of code and documentation that could have resulted in development and maintenance problems. The configuration management anomaly reports concerned incorrect version identification of specification change pages. These reports helped to prevent the dissemination of different document versions bearing the same configuration control markings.

6.3.6 The Cost Benefits of Early Detection

A major objective of the IV&V study was to quantify the effects of IV&V on development cost/productivity. By applying cost/productivity figures found in the literature to IV&V results of the five projects surveyed, it was possible to obtain an estimate of the cost benefits associated with early detection of anomalies.

A basic question that had to be addressed before any analysis could take place was which anomaly reports to consider. Possibilities were:

- All of them
- Only those that were accepted and corrected
- Only those that affected reliability

Consideration of this question led to the following conclusions:

- An anomaly report has a cost/productivity benefit on the development effort if it reports a problem that would have surfaced later in the development or in operation and required correction at that time.
- The problems that would have surfaced later in the development effort or in operation and required correction at that time were those affecting program reliability.

- Reports concerning maintainability anomalies result in life cycle cost savings but not necessarily in development cost savings, the subject of the analysis.
- Invalid reports or those that were not acted on would not have a cost/productivity benefit.

To make the analysis meaningful, therefore, it was limited to those anomaly reports that were concerned with program reliability, accepted as valid by the program office, and acted on by the developer.

Two analyses were performed. The approach taken for the first was as follows:

- Select the anomalies meeting the criteria outlined above.
- Separate the selected anomalies according to the development phases in which they were detected.
- Apply the error-correction cost figures reported by Wolverton in Reference 30:
 - Correction during requirements definition: \$195
 - Correction during design: \$489
 - Correction during coding and checkout: \$977
 - Correction during test and integration: \$7136
- Assume that all of the anomalies would have been detected during development testing if IV&V had not been performed.
- Calculate the cost savings realized by detecting anomalies before the testing phase.

Table 10 shows the results of this analysis. In keeping with the assumption that all anomalies would have been detected during development testing, the anomalies detected during the testing phase are shown to have no cost benefit. Those detected earlier are shown to have cost benefits increasing with their distance from the testing phase. Total savings are shown in the right-most column.

The most striking finding is that even assuming that all anomalies would have been detected by the developer, substantial cost benefits can be demonstrated based on early detection alone. Even with 270 anomalies assumed to have no cost benefit at all, the figures show an average savings of \$601,613 per project. For Project 4, the savings exceed the cost of the IV&V project, despite the fact that most requirement and design anomalies were not documented in anomaly reports, so are not shown in the analysis. Project 3 savings come close to its cost. The lessons shown by the table are clear:

- The earlier anomalies are detected, the greater the cost benefits.

Table 10. Cost Benefits of Early Detection--Scenario 1¹

Pro- ject	Requirements Definition Phase				Design Phase				Coding and Checkout Phase				Testing Phase				
	No. of Anom- alies	Cost to Fix		Savings	No. of Anom- alies	Cost to Fix		Savings	No. of Anom- alies	Cost to Fix		Savings	No. of Anom- alies	Cost to Fix		Savings	Total Savings
		In Req Def.	In Testing			In Design	In Testing			In Coding	In Testing			Cost to Fix	Cost to Fix		
1	--	--	--	--	--	--	--	--	72	\$ 70,344	\$ 513,792	\$ 443,448	116	\$ 827,776	\$ 0	\$ 443,448	
2	--	--	--	--	--	--	--	--	127	\$124,079	\$ 906,272	\$ 782,193	89	\$ 635,104	\$ 0	\$ 782,193	
3	11	\$1,950	\$ 71,369	\$ 69,410	42	\$20,534	\$259,712	\$279,174	60	\$ 58,420	\$ 428,160	\$ 369,540	27	\$ 192,672	\$ 0	\$ 718,124	
4	22	\$4,250	\$156,992	\$152,702	6	\$ 2,934	\$ 42,316	\$ 39,382	56	\$ 54,712	\$ 309,616	\$ 344,904	6	\$ 42,816	\$ 0	\$ 537,488	
5	--	--	--	--	32	\$15,648	\$228,352	\$212,704	51	\$ 49,327	\$ 363,936	\$ 314,109	32	\$ 228,352	\$ 0	\$ 526,813	
11	32	\$6,240	\$228,352	\$222,112	80	\$39,120	\$570,880	\$531,760	366	\$357,582	\$2,611,776	\$2,254,194	270	\$1,926,720	\$ 0	\$3,008,066	

Notes:

1. This scenario assumes that all anomalies would have been detected in testing if I&V had not been performed
2. Anomaly counts represent those anomalies that required and received action for program reliability
3. Cost figures are based on the following (Reference 30):

Phase of Detection		Cost To Fix an Anomaly		Savings Over Waiting Until the Testing Phase	
Requirements Definition		\$ 195		\$6,941	
Design		\$ 489		\$6,547	
Coding and Checkout		\$ 977		\$6,159	
Testing		\$7,136		\$ 0	

- IV&V can pay for itself through early detection of errors.

The second analysis was identical to the first except that instead of assuming that all anomalies would have been detected during development testing, a scenario was adopted in which 10% of the anomalies went undetected into the operational environment and were found there. Figures published by Finfer (Reference 8) indicate that the cost to correct such problems is 15 times the cost if detected during coding and checkout. Combining this finding with Wolverton's cost figures produces an average of \$14,655 to correct an error once the program becomes operational.

Table 11 shows the results of this analysis. The average savings are \$714,097 per project, up from \$601,613 in the first analysis. Savings shown for Projects 3 and 4 exceed the cost of the IV&V efforts. Not shown is the fact that if even one of the errors not detected until operational use had a mission-threatening impact, the cost benefit of IV&V detection would be much higher. If one of these errors had a life-threatening impact, the benefit would be incalculable.

6.3.7 Cost/Productivity Effects of Anomaly Report Processing

The time required to analyze and respond to anomaly reports is the most conspicuous imposition of IV&V on development productivity. The question that arises is whether there are sufficient productivity gains associated with each report to offset the time that must be spent.

To answer this question, the anomaly reports for which resolution was known were divided into four groups:

- Reports that were concerned with reliability, declared valid, and acted on
- Reports that were declared invalid or that were withdrawn by the IV&V agency
- Reports that were declared valid and were acted on, but did not concern reliability
- Valid anomaly reports that were not acted on

The following paragraphs explore the cost/productivity impacts of each type of report.

6.3.7.1 Effects of Corrected Reliability Anomaly Reports: The first group of anomaly reports were those that were concerned with program reliability, judged valid by the program office, and acted on by the developer. These reports concerned anomalies that would have shown up either during program testing or during operational use, that is, anomalies that the programmer would have had to deal with eventually. The analysis assumed that all of the anomalies would have been detected during program testing rather than the more troublesome and expensive case of their showing up in operational use.

Table 11. Cost Benefits of Early Detection-- Scenario 2¹

Proj- ect	Requirements Definition Phase			Design Phase			Coding and Checkout Phase			Testing Phase		
	Cost To Fix			Cost To Fix			Cost To Fix			Cost To Fix		
	No. of Anom- alies	In Req. Def.	Savings	No. of Anom- alies	In Design	Savings	No. of Anom- alies	In Coding	Savings	No. of Anom- alies	In Testing	Savings
1	--	--	--	--	--	--	72	\$ 70,344	\$ 567,929	116	\$ 827,776	\$ 914,966
2	--	--	--	--	--	--	127	\$124,079	\$1,001,764	89	\$ 635,104	\$ 702,024
3	13	\$1,858	\$ 72,679	42	\$20,514	\$311,292	60	\$ 56,620	\$ 471,214	27	\$ 192,672	\$ 212,974
4	22	\$4,290	\$171,534	4	\$ 2,734	\$ 47,327	56	\$ 54,712	\$ 441,722	6	\$ 42,816	\$ 47,327
5	--	--	--	32	\$15,648	\$352,413	51	\$ 49,827	\$ 402,242	32	\$ 228,352	\$ 252,413
All	32	\$6,210	\$252,413	80	\$39,120	\$1,011,912	156	\$357,582	\$2,966,971	230	\$1,926,720	\$2,129,734
												\$203,014
												\$3,570,488

Notes:

1. This scenario assumes that 70% of the anomalies would have been detected in testing, 10% detected in operational use
2. Anomaly counts represent those anomalies that required and received action for program reliability
3. Cost figures are based on the following (References 2, 3):

Phase of Detection	Cost To Fix as Anomaly	Savings Over Waiting Until Testing Phase	Savings Over Waiting Until Operational Phase
Requirements Definition	\$ 195	\$6,941	\$14,460
Design	\$ 439	\$6,847	\$14,166
Coding and Checkout	\$ 977	\$6,159	\$13,678
Testing	\$ 7,116	\$ 0	\$ 7,519
Operational Use	\$14,555	--	\$ 0

In a paper on error remediation expenditures (Reference 35), Herndon and Keenan present the following equation:

Error remediation cost equals the sum of the costs of:

- Error handling:
 - Trouble report generation
 - Management analysis
 - Resolution form generation
 - Configuration control board actions
- Error analysis
- Error correction
- Retesting

They give as averages the following figures for the hours required for each of these activities:

Trouble report generation	.17 hours
Management analysis	.17 hours
Resolution form generation	.25 hours
Configuration control board action	.67 hours
Error analysis	0-12.0 (say 6) hours
Error correction	0-24.0 (say 12) hours
Retesting	<u>1.94 hours</u>
Total	21.2 hours

Other authors present considerably higher estimates. Miyamoto, for example, reports that the average "find-and-fix" cycle for program faults in a large system was 16.56 days (Reference 36). Multiplying this figure by a conservative 6 hours per day results in 99.36 hours, nearly 4.7 times the figure given by Herndon and Keenan.

The figures reported in the literature appear to depend upon the size and complexity of the software being developed. For the analysis of IV&V results, it was decided to use both sets of figures quoted above in order to represent the range of values reported in the literature. Incorporating Miyamoto's result into the Herndon-Keenan model produces the following estimate for error remediation:

Trouble report generation	.17 to .8 hours
Management analysis	.17 to .8 hours
Resolution form generation	.25 to 1.18 hours
Configuration control board action	.67 to 3.15 hours
Error analysis	6.0 to 28.2 hours
Error correction	12.0 to 56.4 hours
Retesting	<u>1.94 to 9.12 hours</u>
Total	21.40 to 99.36 hours

Shooman and Bolsky (Reference 6) add that the typical amount of computer time used to diagnose a problem is 0-4 runs, using 0-30 minutes. They cite as a mean .61 runs and 13.5 minutes (.225 hours) of computer time.

When a program fault is detected by IV&V rather than by the developer in program testing, the impact on remediation expenditure is as follows:

- Trouble Report Generation: No trouble report has to be generated by the development test team, resulting in a savings of .17 to .8 hours per anomaly.
- Error Analysis: The time needed for error analysis is significantly reduced because, unlike a trouble report, which generally describes only the symptoms of a fault as they were observed during testing, an anomaly report identifies the specific error made and often recommends corrective action. A conservative estimate would be that error analysis time is cut in half when starting with an anomaly report rather than a trouble report--a savings of 3.0 to 14.1 hours per anomaly.
- CPU Diagnostic Time: The computer time needed for diagnostic runs is also reduced because of the specific information provided in the anomaly report. Assuming a reduction by half results in a savings of .1125 computer hours per anomaly.
- Paperwork: Paperwork, in the form of anomaly response forms, special resolution forms, and so on, may or may not be greater than that required in response to trouble reports. Assuming that the time required for paperwork is increased by half results in an extra .125 to .59 hours per anomaly.
- Management Analysis: The time required to evaluate tradeoffs in correcting the problem may be greater in responding to anomaly reports than trouble reports because of IV&V and customer concurrence requirements. Assuming that the time is increased by half results in an extra .085 to .4 hours per anomaly.
- Other Factors: All other factors in the Herndon-Keenan model would be the same whether processing an anomaly report or a developer-generated trouble report.

Table 12 shows the results of this analysis. The cost figures discussed above have been multiplied by the number of anomaly reports on each project so that the total hours saved and the extra hours expended can be seen.

Despite the consistent use of conservative assumptions, the figures indicate that the processing of anomaly reports documenting problems that would have had to be corrected eventually actually saves programmer time. The estimated savings for each anomaly is 2.96 to 13.9 hours, plus 6.75 minutes of computer time. For Project 4, which had the smallest number of anomalies in this category, estimated savings in programmer time ranged from 267 to 1,252 hours. For Project 2, with the greatest number of anomalies, estimated savings ranged

Table 12. IV&V Effects on the Cost of Defect Removal

Project	No. of Anomalies ¹	Hours Saved			Extra Hours			Net Savings	
		% Generation ²	Error Analysis ³	Computer Time ⁴	Paperwork ⁵	Management Analysis ⁶	Personnel Hours	Dollars ⁷	Computer Hours
1	188	32-120	564- 2,651	21.2	24-111	15-75	556- 2,615	\$22,240-\$104,600	21.2
2	216	37-173	648- 3,046	24.3	27-127	18-86	640- 3,006	\$25,600-\$120,240	24.3
3	139	24-111	417- 1,960	15.6	17-82	12-56	412- 1,933	\$16,480-\$ 77,320	15.6
4	90	15-72	270- 1,269	10.1	11-53	7-36	267- 1,252	\$10,680-\$ 50,080	10.1
5	115	20-82	345- 1,672	12.9	14-68	10-46	341- 1,600	\$13,640-\$ 64,000	12.9
All	748	118-189	2,743-13,547	84.1	93-442	63-300	2,216-10,404	\$88,640-\$416,180	84.1

Notes:

1. Using reliability anomalies that were accepted and acted on
2. Assumes .17 to .18 hours saved for each anomaly report
3. Assumes 3.0 to 14.1 hours saved for each anomaly report
4. Assumes .1125 computer hours saved for each anomaly report
5. Assumes .125 to .55 extra hours for each anomaly report
6. Assumes .085 to .4 extra hours for each anomaly report
7. Assumes 140 per hour manpower cost

from 640 to 3,006 programmer hours. It should be noted that these findings do not take into account the additional cost advantages of early detection. They assume, in effect, that all anomaly reports were submitted in the testing phase.

6.3.7.2 Effects of Invalid Anomaly Reports: One way in which IV&V can waste programmer time is by submitting invalid anomaly reports. Reports may be invalid because:

- They identify faults where none exist.
- They recommend changes that are beyond the scope of the development effort.
- They recommend changes that are not necessary for satisfactory program performance.

The number and percentage of anomaly reports declared invalid on the five projects was as follows:

Project 1:	9	(4%)
Project 2:	40	(12%)
Project 3:	25	(5%)
Project 4:	3	(2%)
Project 5:	18	(7%)
Total	95	(6%)

An average of 6% of the anomalies reported were declared invalid. Project 2, with 12%, had the highest number. When questioned about this figure, project participants explained that there was disagreement between the developer and the IV&V team as to the degree of accuracy that could and should be achieved in program calculations, and the disagreement resulted in quite a few IV&V recommendations being declared invalid.

To quantify the effects of processing invalid anomaly reports, the following assumptions were made:

- The human and computer time spent in analyzing and rejecting an invalid report may be as great as the time spent in processing a valid one.
- The time spent for paperwork and management analysis may also be as great for an invalid report as for a valid one.
- The other costs associated with error remediation do not come into play in processing invalid reports.

These assumptions yield the following estimates for time expended on each invalid anomaly report:

- Error analysis: 3.0 to 14.1 hours
- Computer time: 6.75 minutes
- Paperwork: .375 to 1.77 hours
- Management analysis: .225 to 1.2 hours
- Total programmer time: 3.63 to 17.07 hours

Table 13 shows the results of these assumptions. The amount of time spent processing each invalid anomaly report was estimated to be from 3.6 hours to 17 hours. Thus on Project 2, with 40 such anomaly reports, the amount of time was 145 to 683 hours, costing \$5,800 to \$27,320. On Project 4, with only 3 such reports, the amount of time was 11 to 51 hours, costing \$440 to \$2,040.

Minimizing these figures should be a goal of all IV&V projects. The program office can support this effort by making both the developer and the IV&V contractor aware of the scope and nature of anomaly reports that will be considered valid.

6.3.7.3 Effects of Corrected Non-Reliability Anomalies: Anomalies that have been corrected but do not concern reliability are primarily maintainability anomalies. These anomalies are concerned with extraneous code, violations of development standards, incorrect documentation, and so on. These problems would not arise in program testing and therefore do not fit into the error remediation analysis presented above.

The number and percentage of such anomalies was as follows:

Project 1:	13	(5%)
Project 2:	18	(6%)
Project 3:	252	(49%)
Project 4:	64	(37%)
Project 5:	31	(10%)
All	378	(24%)

Nearly half of the anomalies on Project 3 and over a third of those on Project 4 fell into this category. Percentages for the other projects were 10% or less.

From the point of view of the developer, these anomaly reports may be considered pure overhead--they must be dealt with even though they do not help to remove operational problems that could impair developer testing. The cost benefits of these anomalies come into play only when the entire life cycle of the software is considered. Correcting them decreases the productivity of the development team, but increases the productivity of the maintenance team. No quantitative data on this tradeoff could be found.

6.3.7.4 Effects of Valid Reports That Were Not Acted On: Anomalies in the fourth category were those for which the program office decided that it was not cost-effective to make a correction even though the problem was real. The number and percentage of such anomalies was as follows:

Table 13. Cost Effects of Invalid Anomaly Reports

Project	Invalid Anomalies	Hours Spent				Total Impact			
		Report Analysis ¹	Computer Time ²	Paperwork ³	Management Analysis ⁴	Personnel		Computer Hours	
						Hours	Dollars ⁵		
1	9	27- 127	1.0	3-16	2-11	32- 154	\$ 1,280-\$ 6,160	1.0	
2	40	120- 564	4.5	15-71	13-48	145- 683	\$ 5,800-\$27,320	4.5	
3	25	75- 353	2.8	9-44	6-30	90- 427	\$ 3,600-\$17,080	2.8	
4	3	9- 42	0.3	1-5	1-4	11- 51	\$ 440-\$ 2,040	0.3	
5	18	54- 254	2.0	7-32	5-22	66- 308	\$ 2,645-\$12,320	2.0	
All	95	235-1,340	10.7	36-168	24-114	345-1,622	\$13,800-\$64,880	10.7	

Notes:

1. Assumes 3 to 14.1 hours per anomaly report
2. Assumes .1125 computer hours per anomaly report
3. Assumes .375 to 1.77 hours per anomaly report
4. Assumes .255 to 1.2 hours per anomaly report
5. Assumes \$40 per hour manpower cost

Project 1:	19	(8%)
Project 2:	19	(6%)
Project 3:	12	(2%)
Project 4:	12	(7%)
Project 5:	88	(28%)
All:	150	(10%)

Project 5, because of the experimental nature of the development, had an unusually high number of this type. The other projects show more typical results, averaging approximately 6%.

The cost impact of these anomaly reports could not be determined. On the negative side, they required processing time without producing any reliability or maintainability gains. On the positive side, they reported real problems that may have been encountered in testing, and therefore may have saved the time that would have been devoted to those problems. No way was found to quantify these effects.

6.3.7.5 Summary of Anomaly Report Processing Effects: Table 14 summarizes the contents of Sections 6.3.7.1-4. The primary conclusions of the analysis were as follows:

- The saving of programmer time on the valid, fixed, reliability anomalies far outweighs the time expended on invalid anomaly reports.
- The time spent fixing non-reliability anomalies results in long-range savings in the form of more maintainable software.
- Valid, unfixed anomalies have the mixed effect of requiring handling time but pointing out problems that the developer or maintainer may have to be aware of.

Table 14. Summary of Anomaly Report Processing Effects

<u>Anomaly Report Type</u>	<u>Number</u>	<u>Cost/Productivity Impact</u>
<u>Project 1:</u>		
Valid, fixed, reliability	188	Savings of 556-2,615 hours; 21.2 computer hours
Invalid	10	Expenditure of 32-154 hours; 1.0 computer hours
Valid, fixed, non-reliability	13	Development time expended; maintenance time saved
Valid, not fixed	21	Indeterminate effect
<u>Project 2:</u>		
Valid, fixed, reliability	216	Savings of 640-3,006 hours; 24.3 computer hours
Invalid	40	Expenditure of 145-683 hours; 4.5 computer hours
Valid, fixed, non-reliability	19	Development time expended; maintenance time saved
Valid, not fixed	19	Indeterminate effect
<u>Project 3:</u>		
Valid, fixed, reliability	138	Savings of 408-1,920 hours; 15.5 computer hours
Invalid	25	Expenditure of 90-427 hours; 2.8 computer hours
Valid, fixed, non-reliability	303	Development time expended; maintenance time saved
Valid, not fixed	19	Indeterminate effect
<u>Project 4:</u>		
Valid, fixed, reliability	90	Savings of 267-1,252 hours; 10.1 computer hours
Invalid	3	Expenditure of 11-51 hours; 0.3 computer hours
Valid, fixed, non-reliability	66	Development time expended; maintenance time saved
Valid, not fixed	12	Indeterminate effect
<u>Project 5:</u>		
Valid, fixed, reliability	115	Savings of 341-1,600 hours; 12.9 computer hours
Invalid	19	Expenditure of 66-308 hours; 2.0 computer hours
Valid, fixed, non-reliability	31	Development time expended; maintenance time saved
Valid, not fixed	88	Indeterminate effect
<u>All Projects:</u>		
Valid, fixed, reliability	747	Savings of 2,212-10,391 hours; 84.0 computer hours
Invalid	95	Expenditure of 345-1,622 hours; 10.1 computer hours
Valid, fixed, non-reliability	434	Development time expended; maintenance time saved
Valid, not fixed	150	Indeterminate effect

7. CONCLUSIONS AND RECOMMENDATIONS

The IV&V study analyzed the results of five IV&V projects to determine the effects of IV&V on software reliability, maintainability, and development cost/productivity. The following paragraphs present the study's conclusions and recommendations.

7.1 Study Conclusions

Conclusions resulting from the study were as follows:

- IV&V results depend significantly upon project objectives and directives--IV&V finds the types of problems it is directed to find.
- The primary emphasis of IV&V is on software reliability; software maintainability is deemphasized on many projects.
- While IV&V's effect on reliability cannot be quantified, the impact appears significant.
 - Each project detected an average of 150 anomalies that would have affected program reliability and that were deemed important enough for correction. This is equivalent to 2.2 such anomalies per thousand machine language instructions.
 - Thirteen percent of these anomalies were of High severity, indicating possible threat to life or property; another 35% were of Medium severity, indicating serious threat to mission objectives.
 - None of the programs that underwent IV&V have experienced operational problems that required modification.
- IV&V is being underutilized as a tool for improving software maintainability.
 - Software maintenance costs are approaching 75% of software life cycle cost.
 - This cost can be reduced by designing software with specific maintainability characteristics.
 - IV&V is ideally suited for evaluating these characteristics.
 - IV&V's current charter regarding maintainability is usually limited to evaluating program documentation and identifying latent errors; this role is often deemphasized.
 - On one project where maintainability was emphasized, IV&V detected 330 anomalies whose correction improved software maintainability.

- IV&V is cost-effective, especially if applied early.
 - IV&V costs average 25% of development cost and 20% of total software acquisition cost.
 - Of 125 factors known to influence programmer productivity, IV&V has no effect at all on 90, indicating the limited overhead that IV&V places on the development process.
 - Of the remaining 35 factors, IV&V has a positive effect on 27, a negative effect on 9 (on one factor, both a positive and negative effect could be seen).
 - IV&V increases programmer productivity by saving the time that would have been devoted to false starts and defect removal.
 - IV&V can pay for itself through the cost benefits provided by early detection of anomalies.

7.2 Recommendations for IV&V Planning and Management

Study results led to the following recommendations for increasing IV&V effectiveness on future projects:

- Concerning reliability:
 - Encourage independence of IV&V outlook and techniques by controlling the degree and type of contact between the developer and the IV&V agency.
 - Plan the IV&V project to allow for early detection of problems so that there is time for adequate redesign.
 - Ensure that corrections to anomalies are reverified by the IV&V agency.
- Concerning maintainability:
 - Direct the IV&V agency to evaluate the software for maintainability as well as for reliability.
 - Draw up a checklist of specific maintainability criteria such as those in Appendix C.
 - Schedule the development and IV&V efforts so that there is time after the conclusion of testing for the IV&V agency to evaluate final program documentation.
 - Treat maintainability anomalies as seriously as reliability anomalies; perhaps establish separate criteria for High, Medium, and Low severity ratings for these anomalies.

- Concerning cost/productivity:

- Stress early detection of anomalies.
- Include in the IV&V process requirements verification and design verification performed in parallel with the requirement and design development phases.
- Ensure delivery of preliminary requirement materials, design materials, and code so that IV&V can proceed in parallel with the development and provide feedback into each phase.
- Clarify IV&V scope to minimize the number of anomaly reports declared invalid.
- Minimize the overhead associated with anomaly report handling while still maintaining high visibility into anomaly report resolution.

7.3 Recommendations for Future Study

A number of interesting questions arose during the study that could not be answered with the data available. The following paragraphs identify these questions and provide some insight into the issues involved in answering them.

7.3.1 IV&V Productivity

Considerable attention has been devoted to measuring and improving programmer productivity. To our knowledge, however, no attempt has been made to measure IV&V productivity, or even to define it.

Programmer productivity is measured in terms of amount of output per unit of time, for example, source lines delivered per month. Attempts to transfer this measurement scheme to IV&V run into problems. The primary output of an IV&V team is anomaly reports. Can an analyst be required to find so many anomalies per week? Is the analyst who detects five anomalies in one week more productive than the analyst who detects only three, or in fact, the analyst who detects none? Are 10 Low-severity anomalies equivalent to 1 High? If 20 anomalies are detected in March and 10 in April, has IV&V productivity decreased 50%?

The answer is both "yes" and "no." Finding anomalies is what IV&V is about, but so is ensuring their absence. Thorough analysis of an error-free subroutine produces no anomaly reports, but cannot be considered a nonproductive activity. Its output, in effect, is a "stamp of approval" for the subroutine, no less valid a product of IV&V than anomaly reports.

If "anomaly reports per unit of time" is not a good measure of IV&V productivity, what is? It would seem to make more sense to measure IV&V productivity in terms of input processed than output produced: lines of code analyzed per week, pages of documentation evaluated per month, test procedures carried out per week, and so on.

IV&V productivity measured in this way could be expected to be influenced by many of the same factors that affect programmer productivity. Program size, complexity, application, and so on affect the difficulty of both developing a program and evaluating it. The same is true for specification quality, programming practices used, personnel experience, and many of the other factors identified in Table 9. One factor not experienced by the development team is the IV&V project's dependence upon the development schedule. When the development schedule slips and required products do not become available, the IV&V schedule necessarily slips as well. The degree of cooperation provided by the developer can also affect IV&V productivity.

Interesting questions concerning IV&V productivity, then, are:

- How should it be defined?
- How do factors that affect programmer productivity affect IV&V productivity?
- What other factors affect it?
- How do different tools, techniques, and procedures affect it?
- How do different development practices affect it?
- What are reasonable productivity rates to expect on a project?
- How can IV&V productivity be improved?

7.3.2 Effects of Modern Programming Practices

Modern programming practices have been incorporated only recently into the types of software verified by IV&V. As a result, the IV&V study was able to include only one such project, a sample clearly too small to permit conclusions to be drawn about the effects of modern programming practices on IV&V. Questions of interest are:

- What types of anomalies are found on projects using modern programming practices?
- How do they compare with the results of other projects?
- Are different IV&V techniques required for these projects?
- Can IV&V help to evaluate the effectiveness of various programming practices?

7.3.3 Effects of Program Characteristics on IV&V Results

The IV&V study had limited information about each program evaluated. Of considerable interest would be a study that related each anomaly to:

- The software function or module in which it was found
- The specific characteristics of that module, such as size, complexity, number of interfaces, etc.
- The development and testing methods used on the development project
- The development standards used on the project
- Other program characteristics

7.3.4 Effects of Various IV&V Tools and Techniques

Software tools can be a powerful aid to IV&V analysis and testing. They can detect the presence of certain types of problems, ensure the absence of others, and aid in the analysis and testing activities. An interesting subject for study would be the detection method used for each anomaly reported. Of particular interest would be:

- The relative effectiveness of manual versus automated analysis
- The relative effectiveness of specific tools and techniques
- The types of problems detected by each tool and technique
- The types of tools and techniques still needed by IV&V

7.4 Recommendations for Data Collection

IV&V and development projects generate enormous amounts of data, not all of which can be saved. The following recommendations are for IV&V recordkeeping procedures that would aid in future studies of this kind.

- Maintain a central anomaly report log for each program evaluated; record for each anomaly:
 - Anomaly report number
 - Anomaly report date and analyst
 - Anomaly location, including document version, program version, routine name, if applicable
 - Anomaly description
 - Anomaly category
 - Special circumstances surrounding the anomaly
 - Anomaly effects

- Anomaly severity
- Development phase when detected
- IV&V phase when detected
- Method or tool used for detection
- Anomaly acceptance or rejection, and date
- Anomaly resolution and date of resolution
- Materials changed and nature of changes
- Acceptability of resolution to the IV&V contractor
- Include a copy of each anomaly report in the log file.
- In a central log for each program, record for each routine:
 - Language used
 - Size
 - Function
 - Number of interfaces
 - I/O characteristics
- In a project management log, record:
 - Monthly man-loading for each IV&V activity
 - Identification of materials evaluated
 - Personnel assignments in terms of pages of documentation or lines of code to be evaluated, test procedures to be performed, etc.
 - Time required to complete each assignment
 - Number of anomalies resulting from each assignment
 - Project costs
 - Accurate schedules showing development deliverables, IV&V activities, IV&V deliverables
 - Statement of objectives
 - Criteria for assigning severity ratings

- List of tools used
- List of project participants

Information that could be recorded by the developer to aid in the analysis of IV&V effects would include:

- Accurate schedules
- Activities that must be performed to support IV&V
- Man-hours spent performing these activities
- Cost of paper, tapes, CPU time, etc., for deliverables to the IV&V contractor
- Development techniques used
- Programmer productivity figures
- Test results

This information about the IV&V and development efforts would be an invaluable contribution to future studies of this type.

APPENDIX A

PROJECT SELECTION

The project selection activity consisted of:

- Establishing selection criteria
- Identifying candidate projects
- Selecting the projects to be used

The following paragraphs describe these activities.

A1. SELECTION CRITERIA

The selection criteria used in the study came from two different sources. Basic criteria were set forth in the study's Statement of Work (SOW). These criteria stated that there were to be at least five IV&V projects, that each was to involve an unclassified Department of Defense (DoD) program, preferably a command, control, communication, and intelligence (C3I) application, and that each was to involve a program having at least 100,000 lines of source code.

To these basic requirements were added additional criteria aimed at ensuring:

- A balance of higher order language (HOL) and assembly language programs
- A balance of real-time and nonreal-time programs
- A balance of modern software engineering and traditional development practices
- A balance of initial development and modification efforts
- IV&V projects performed in parallel with, rather than subsequent to, the development effort
- Complete, rather than on-going, IV&V projects
- IV&V projects that had kept good records

Table A-1 summarizes the two sets of selection criteria.

A2. CANDIDATE PROJECTS

Thirty-five projects were considered as candidates for the study. Table A-2 summarizes their characteristics.

Table A-1. Project Selection Criteria

Criterion	Possible Traits	Desired Trait
<u>SOW Requirements</u>		
Contracting Agency	DoD/Non-DoD	DoD
Program Application	C ³ I/Other	C ³ I
Program Security Classification	Classified/Unclassified	Unclassified
Program Size	>100K/<100K source lines	>100K lines
<u>Additional Criteria</u>		
Programming Language Type	HOL/Assembly	Balance
Program Timing Characteristics	Real-time/Nonreal-time	Balance
Development Technique	Modern software engineering/ Other	Balance
Development Type	New development/Modification	Balance
IV&V Timing	In parallel with/After development	In Parallel
IV&V Status	Complete/Ongoing	Complete
IV&V Data Availability	Good/Fair/Poor/Incomplete	Good

Table A-2. Information About Candidate Projects

Project Number	DoD C-1	Source Lines	Classified	Language Type	RT/RTI	Modern S/W Eng	New/Mod	Parallel IV&V	Project Status	Data	Comments
1	Yes	Yes	?	Part	Asm	Both	No	No	Complete	Poor	Data unavailable
2	Yes	Yes	?	Part	Asm	Both	No	Yes	Complete	Poor	Data unavailable
3	Yes	Yes	93K	Part	HOL/Asm	Both	No	Yes	Complete	Good	Good candidate
4	Yes	Yes	93K	Part	HOL/Asm	Both	No	Yes	Complete	Good	Good candidate
5	Yes	No	23K	No	Asm	RT	No	No	Complete	Fair	Too small
6	Yes	No	23K	No	Asm	RT	No	Yes	Complete	Fair	Too small
7	Yes	No	23K	No	Asm	RT	No	Yes	Complete	Good	Too small
8	Yes	No	23K	No	Asm	RT	No	Yes	Complete	Good	Good data; too small
9	Yes	No	23K	No	Asm	RT	No	Yes	Complete	Good	Too small
10	Yes	No	7K	No	Asm	RT	No	Yes	Complete	Fair	Too small
11	Yes	No	7K	No	Asm	RT	No	Yes	Complete	Fair	Too small
12	Yes	No	176K	No	HOL/Asm	Both	No	Yes	Complete	Good	Good candidate
13	Yes	No	176K	No	HOL/Asm	Both	No	Yes	Complete	Good	Possibility
14	Yes	No	176K	No	HOL/Asm	Both	No	Yes	Ongoing	Incomplete	Too early in project
15	Yes	No	39K	No	HOL/Asm	RT	No	Yes	Complete	Good	Good data; too small
16	Yes	No	39K	No	HOL/Asm	RT	No	Yes	Complete	Good	Too small
17	Yes	No	39K	No	HOL/Asm	RT	No	Yes	Complete	Good	Too small
18	Yes	No	39K	No	HOL/Asm	RT	No	Yes	Complete	Good	Too small
19	Yes	No	90K	No	HOL/Asm	Both	Yes	Yes	Complete	Good	Good candidate

Table A-2. Information About Candidate Projects (continued)

Project Number	DoD C3I	Source Lines	Classified	Language Type	RT/NRT	Modern S/W Eng	New/Mod	Parallel IV&V	Project Status	Data	Comments
20	Yes No	90K	No	HOL/Asm	Both	Yes	Mod	Yes	Ongoing	Incomplete	Too early in project
21	Yes No	---	No	HOL/Asm	Both	Yes	New	Yes	Ongoing	Incomplete	Too early in project
22	Yes No	---	No	HOL/Asm	Both	Yes	New	Yes	Ongoing	Incomplete	Too early in project
23	Yes No	150K	No	HOL/Asm	RT	No	New	Yes	Finishing	Good	Good candidate
24	Yes No	10K	No	Asm	RT	No	New	No	Complete	Poor	Too small
25	Yes No	14K	No	HOL	RT	No	New	No	Complete	Poor	Too small
26	Yes No	1K	No	HOL	RT	No	New	No	Complete	Fair	Too small
27	Yes No	350K	Yes	HOL/Asm	NRT	No	New	No	Complete	Poor	Nonstandard IV&V
28	Yes No	350K	Yes	HOL/Asm	NRT	Yes	Mod	No	Complete	Poor	Nonstandard IV&V
29	Yes No	10K	Yes	HOL/Asm	NRT	No	Mod	Yes	Complete	Fair	Too small
30	Yes No	10K	Yes	HOL	NRT	No	New	Yes	Complete	Good	Too small
31	Yes No	10K	Yes	HOL	NRT	No	Mod	Yes	Complete	Good	Too small
32	Yes No	30K	No	Asm	RT	No	New	Yes	Complete	Good	Good data; too small
33	Yes No	---	Part	HOL/Asm	Both	Yes	New	Yes	Ongoing	Incomplete	Too early in project
34	Yes No	152K	No	HOL/Asm	NRT	Yes	New	Yes	Complete	Fair	Not a true IV&V
35	Yes No	210K	Part	HOL/Asm	Both	Yes	New	Yes	Ongoing	Incomplete	Too early in project

All 35 projects were performed for the DoD. Projects 1 through 4 involved C I applications; the others did not. The number of source lines ranged from a low of 1K to a high of 350K, with most single programs in the range of 10K to 40K lines. Approximately a third of the projects involved some classified components.

Eleven of the projects dealt with programs written entirely in assembly language. Four dealt with programs written entirely in higher order language. The remaining 20 involved programs or systems using both types of languages.

Twenty-nine of the projects dealt with real-time programs. Eight involved programs developed with modern software engineering practices. Approximately half involved initial development; the other half dealt with modifications to existing systems.

Twenty-eight of the IV&V projects were performed in parallel with the development effort, only seven were not. All but six of the projects were already completed. Sixteen of the projects were considered to have good data availability, seven fair, six poor; in six cases, data were incomplete because the projects were not yet complete.

A3. SELECTION PROCESS

Of the 35 candidate projects, 12 met the 100,000-line criterion. Since five of these were either nonstandard IV&V efforts or were in too early a stage for inclusion in the study, the project selection task consisted of picking five from among the seven eligible projects: 3, 4, 12, 13, 14, 19 and 23.

All of these candidates fulfilled certain of the criteria. All were DoD projects. All provided a balance of HOL and assembly language. Any subset would provide a balance of initial development and modification. All were performed in parallel with the development effort; all were complete except for Project 23, which was nearly so; and all were considered to have good availability of data.

One important consideration was the relationship of the projects to one another. Projects 3 and 4 involved two versions of the same system, as did Projects 12, 13, and 14. The seven projects taken together, therefore, represented only four different systems, and a selection that included at least one from each system was desirable. Other considerations were that Project 19 was the only one involving modern programming practices and that Projects 3 and 4 were the only ones involving the desired C³I application.

The final decision was to use Projects 3, 4, 12, 19 and 23. Projects 3 and 4 were both included because of the desirability of using C³I projects in the study. Project 12 was selected from among Projects 12, 13 and 14 because it represented the initial development of the system. Projects 19 and 23 rounded out the selection by bringing into the study the other two candidate systems. These five projects, in the order discussed, were renamed Projects 1 through 5 for the remainder of the study.

APPENDIX B

DATA COLLECTION

The data collection activity consisted of:

- Translating the study's objectives into specific questions that could be answered by the study
- Identifying the data needed to answer these questions
- Developing data collection forms
- Obtaining project records
- Recording relevant data
- Converting the data to machine-readable form

The following paragraphs describe these activities.

B1. KEY QUESTIONS

Table B-1 identifies the key questions identified for the study. These questions focus on the number and characteristics of anomaly reports affecting software reliability, the number and characteristics of anomaly reports affecting software maintainability, quantitative data concerning IV&V's effect on development cost and productivity, and ways in which project characteristics affect IV&V results.

B2. REQUIRED DATA

Analysis of the study's key questions revealed that three types of data were needed:

- Data concerning each anomaly reported by IV&V
- Data concerning IV&V project characteristics
- Data concerning development project characteristics

Table B-2 identifies the specific information identified for each of these categories.

B3. DATA COLLECTION FORMS

Three data collection forms were developed for the study, corresponding to the three types of data required:

- An anomaly questionnaire

Table B-1. Key Questions Identified for the Study

Questions concerning software reliability:

- How many of the anomaly reports submitted affected software reliability?
- What development materials did they involve?
- What types of anomalies were they?
- How severe were they?
- What aspects of reliability would they have affected?
- What was the operational reliability of the resulting software?

Questions concerning software maintainability:

- How many of the anomaly reports submitted had a direct affect on software maintainability?
- What development materials did they involve?
- What types of problems did they report?
- How severe were they?
- What aspects of maintainability would have been affected?
- What were the indirect effects of reliability anomalies on maintainability?

Questions concerning development cost and productivity:

- What was the average ratio of IV&V cost to development cost?
- What factors affected this ratio?
- In what ways did IV&V increase development cost?
- In what ways did IV&V decrease development cost?
- What was the cost effect of early detection?
- What was the overall cost impact of IV&V?

Questions concerning the improvement of IV&V effectiveness:

- How did different IV&V project characteristics affect IV&V results?
- How did different development project characteristics affect IV&V results?

Table B-2. Data Needed From Each Project

Data concerning each anomaly reported by IV&V:

- Location (specification, code, etc.)
- Type of problem
- Probable effects if left uncorrected
- Severity
- Detection date
- Detection method
- Resolution
- Resolution date

Data concerning each IV&V project:

- Objectives
- Schedule
- Man-loading
- Relationship with developer
- Tools and techniques used
- Cost

Data concerning each development project:

- Schedule
- Man-loading
- Development practices used
- Programmer productivity
- Test results
- Software operational performance
- Software maintenance requirements
- Cost

- An IV&V project questionnaire
- A development project questionnaire

Figures B-1, B-2, and B-3 illustrate these questionnaires.

In preparing the anomaly questionnaire, current literature concerning error classification was surveyed. Table B-3 presents a sampling of the error classification schemes found. A significant characteristic of many of these schemes was their focus on coding errors. Requirement and design errors were frequently relegated to a single category or ignored altogether. Because IV&V monitors the entire development process and reports anomalies in requirement and design specifications as well as code, these schemes were unsuitable for the study.

A notable exception to this coding orientation was the classification scheme devised by the Software Acceptance Criteria Panel of the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management. This scheme, reported by Hartwick (Reference 38)* includes categories in three areas: specifications, code, and data. It proved the most useful as a basis for developing anomaly categories.

B4. OBTAINING PROJECT RECORDS

Three types of records were needed for the study:

- IV&V technical results
- IV&V project data
- Development project data

IV&V technical results were obtained from anomaly reports and anomaly resolution records for each project. Figures B-4 and B-5 provide an example of each of these forms. IV&V project data were obtained from accounting records, project reports, interviews with project participants, and information provided by Air Force project officers. Development project data were obtained by channeling requests through RADC to the appropriate Air Force project officers. Table B-4 identifies the project records that were obtained.

B5. RECORDING THE DATA

The data-recording activity involved completion of a development and IV&V project questionnaire for each IV&V effort and an anomaly questionnaire for each anomaly reported. The development questionnaires were, for the most part, filled out by the Air Force project officers contacted by RADC. The

* Hartwick, R. D., Software Acceptance Criteria Panel Report, Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management, Software Workshop, April 1979.

ANOMALY QUESTIONNAIRE

Respondent _____

1. Program _____ Version _____ Report No. _____ Part _____

2. Report Date _____ Analysts _____

3. Anomaly Description

a. Anomaly location:

- | | |
|--|--|
| ☐ System/subsystem specification | ☐ Code |
| ☐ Interface specification | ☐ After-code design specification |
| ☐ Software requirements specification | ☐ User documentation |
| ☐ Before-code design specification | ☐ Other _____ |

b. Brief description of anomaly _____

c. Anomaly category:

Requirement specification:

- ☐ Incorrect requirements
- ☐ Inconsistent requirements
- ☐ Incomplete requirements
- ☐ Other requirement problems:
 - ☐ Unclear, untestable
 - ☐ Unfeasible, questionable
 - ☐ Extraneous, inappropriate
 - ☐ Other
- ☐ Presentation problems
 - ☐ Standards, development practices
 - ☐ Configuration management
 - ☐ Other

After-code design specification or user documentation:

- ☐ Incorrect documentation
- ☐ Inconsistent documentation
- ☐ Incomplete documentation
- ☐ Other content problems
- ☐ Presentation problems
 - ☐ Standards, development practices
 - ☐ Configuration management
 - ☐ Other

Other development problems

- ☐ Hardware system, other programs
- ☐ Other documents
- ☐ Unknown origin
- ☐ Development process
- ☐ Other

Before-code design specification or code:

- ☐ Requirement/design compliance
- ☐ Choice of algorithm, mathematics
- ☐ Sequence of operations
- ☐ Data definition
- ☐ Data handling:
 - ☐ Initialization
 - ☐ Addressing, indexing
 - ☐ Misuse of flags
 - ☐ Misuse of counters
 - ☐ Shared memory locations
 - ☐ Other
- ☐ Timing or interruptibility
- ☐ Interfaces or I/O:
 - ☐ Input handling
 - ☐ Output
 - ☐ Hardware interfaces
 - ☐ External software interfaces
 - ☐ Routine interfaces
- ☐ Other design/code problems
 - ☐ Extraneous/inefficient
 - ☐ Program error handling
 - ☐ Other
- ☐ Design/code presentation
 - ☐ Standards, development practices
 - ☐ Configuration management
 - ☐ Comments, annotations
 - ☐ Other

d. Special circumstances:

- ☐ An error made while correcting a previously-reported anomaly
- ☐ A disagreement among development materials with none clearly wrong
- ☐ A non-optimal, rather than incorrect, development decision
- ☐ A latent error--not wrong now but could cause maintenance problems
- ☐ A hold-over from a previous version of the program
- ☐ A "clone" of a previously-reported anomaly
- ☐ Other _____

Figure B-1. The Anomaly Questionnaire

ANOMALY QUESTIONNAIRE--page 2

4. Consequences of Anomaly:

a. The anomaly could affect the program's

☐ Development ☐ Operation ☐ Usability
☐ Verifiability ☐ Maintainability ☐ Other _____

b. If it could affect operation, would it specifically affect:

☐ Correctness ☐ Security ☐ Other _____
☐ Accuracy precision ☐ Efficiency

c. Severity of consequences:

☐ High ☐ Medium ☐ Low ☐ Unknown

5. Detection Information

a. IV&V activity at time of detection:

☐ Requirements verification ☐ Validation/testing
☐ Design verification ☐ Documentation verification
☐ Code verification ☐ Other _____

b. Development phase at time of detection:

☐ Requirements definition ☐ Testing
☐ Design ☐ Post-testing
☐ Coding and checkout ☐ Other _____

c. Tools or methods that resulted in detection:

☐ Manual analysis, specifically _____
☐ Program execution, specifically _____
☐ Tool use, specifically _____

6. Anomaly Resolution:

a. Anomaly acceptance:

☐ Accepted as written ☐ Withdrawn/superseded
☐ Accepted with changes ☐ Unknown
☐ Rejected ☐ Other _____

b. Action taken:

☐ Fixed and: ☐ Fix/workaround deferred and:
☐ Fix was verified ☐ Taken care of later
☐ Fix found wrong ☐ Not taken care of later
☐ Fix not verified ☐ Outcome unknown or pending
☐ Workaround adopted ☐ Action unknown
☐ Negated by another change ☐ Other _____
☐ No action to be taken

c. Materials changed in response to report:

☐ System/subsystem specification ☐ Code
☐ Interface specification ☐ User documentation
☐ Software requirements specification ☐ Unknown
☐ Design specification ☐ Other _____

d. Resolution date _____

Figure B-1. The Anomaly Questionnaire (continued)

IV&V PROJECT QUESTIONNAIRE

Respondent _____

1. Program _____ Version _____

2. Cost of IV&V project:

- a. Total _____
- b. Labor _____
- c. Computer _____
- d. Documentation _____
- e. Other support _____

3. Duration of project (give start and stop dates):

- a. Total _____
- b. Requirements verification _____
- c. Design verification _____
- d. Code verification _____
- e. Testing _____
- f. Documentation verification _____

4. Man-months expended:

- a. Total _____
- b. Requirements verification _____
- c. Design verification _____
- d. Code verification _____
- e. Testing _____
- f. Documentation verification _____

5. Relationship with developer - check one:

- a. Good
- b. Fair -- some hostility
- c. Poor -- very poor cooperation; considerable hostility

6. Tools used on project:

7. Project participants:

Figure 9-2. The IV&V Project Questionnaire

DEVELOPMENT PROJECT QUESTIONNAIRE Respondent _____

1. Program _____ Version _____

2. Development cost:

- a. Total _____
- b. Labor _____
- c. Computer _____
- d. Documentation _____
- e. Other support _____

3. Development duration (give start and stop dates):

- a. Total _____
- b. Requirements definition phase _____
- c. Design phase _____
- d. Code and checkout phase _____
- e. Testing phase _____

4. Man-months expended:

- a. Total _____
- b. Requirements definition _____
- c. Design _____
- d. Code and checkout _____
- e. Testing _____

5. If programmer productivity figures were kept, please provide them.

6. Tools used on project:

- a. Were problems/errors recorded during program testing? _____
- b. If so:
 - How many? _____
 - Please provide copies of problem/error reports if available.
 - Please provide problem/error resolution information if available.

7. Operational performance:

- a. Were problems/errors detected during operational use? _____
- b. If so:
 - How many? _____
 - Please provide copies of problem/error reports if available.
 - Please provide problem/error resolution information if available.

Figure B-3. The Development Project Questionnaire

DEVELOPMENT PROJECT QUESTIONNAIRE -- page 2

8. Maintenance:

- a. When did the program go operational? _____
- b. Has the program been modified since going operational? _____
- c. If so, was modification due to:
 - _____ User requirements change
 - _____ Problems/errors detected in operational use
 - _____ Other: _____
- d. If modification was due to problems/errors, describe or provide documentation describing the needed changes.

9. Which of these programming practices were used? (See RADC Computer Software Development Specification No. CP 0787796100E)

	Yes	No	If yes, to what extent? If no, what was used instead?
a. Top down design	_____	_____	_____
• At program level	_____	_____	_____
• At system level	_____	_____	_____
b. Program support library			
• Manual PSL	_____	_____	_____
• Basic PSL	_____	_____	_____
• Full PSL with management data collection and reporting	_____	_____	_____
c. Language standards			
• Structured code accomplished with simulated constructs	_____	_____	_____
• Structured code accomplished with preprocessor	_____	_____	_____
• Structured code directly compliant	_____	_____	_____
d. Coding conventions/procedures			
• RADC-required coding conventions	_____	_____	_____
• RADC conventions with code reading	_____	_____	_____
• RADC conventions with design code reviews	_____	_____	_____
e. Personnel organization			
• Modified programmer team	_____	_____	_____
• Full programmer team	_____	_____	_____

Figure 8-3. The Development Project Questionnaire (continued)

Table B-3. Error Categories Found in the Literature

Authors	Major Error Categories
Amory, Clapp (Reference 39)*	Input data Internal data Computation procedures Control procedures Interface procedures
Rubey (Reference 2)	Incomplete or erroneous specification Intentional deviation from specification Violation of programming standard Erroneous data accessing Erroneous decision logic or sequencing Erroneous arithmetic computations Invalid timing Improper handling of interrupts Wrong constants and data values Inaccurate documentation
Dana, Blizzard (Reference 7)	Incomplete or erroneous specification Specification violation due to incorrect implementation Violation of programming practices Incorrect data/instruction access and storing Incorrect logic and sequencing Incorrect branching and jumping Incorrect equation computation and arithmetic Incorrect timing and process allocation Problems with interruptibility and data coherency Incorrect constant value and data formats Incorrect documentation Erroneous use of system hardware/software
Thayer, et al. (Reference 5)	Computation Logic Data input Data handling Data output Interface Data definition Data base Operation Other Documentation

* Amory, W., and Clapp, J.A., Engineering of Quality Software Systems (A Software Error Classification Methodology), RADC-TR-74-324, Vol. VII, Jan. 1975

Table B-3. Error Categories Found in the Literature (continued)

Authors	Major Error Categories
Hartwick (Reference 38)	Software specification - Unnecessary functions - Incomplete requirements or design - Inconsistent requirements or design - Untestable requirements or design - Requirements not traceable to higher specification - Incorrect algorithm - Incomplete or inaccurate interface specifications Code - Syntax errors - Noncompliance with specifications - Interface errors - Exception handling errors - Shared variable accessing errors - Software support environment errors - Violation of programming standards - Operational support environment errors Data - Accuracy - Precision - Consistency
Endres (Reference 40)*	Understanding the problem/choice of algorithm - Machine configuration or architecture - Dynamic behavior and communication between processes - Functions offered - Output listings and formats - Diagnostics - Performance Implementation - Initialization of fields and areas - Addressability - Reference to names - Counting and calculating - Masks and comparisons

* Endres, A., "An Analysis of Errors and Their Causes," Proceedings of the International Conference on Reliable Software, April 1975, pp. 327-336.

Table B-3. Error Categories Found in the Literature (continued)

Authors	Major Error Categories
	- Estimation of range limits - Placing of instructions within a module, bad fixes
	Nonprogramming errors
	- Spelling errors in messages and commentaries - Missing commentaries or flowcharts - Incompatible status of macros or modules - Other
Bowen (Reference 41)*	Design Interface Data definition Logic Data handling Computational Other
AN/SLQ-32(V) V&V SOW (Reference 41)	Requirements Processing design Data base design Interface design Processing construction Data base construction Interface construction Verification Specification/documentation
Bowen (Reference 41)	Expanded, reduced, or erroneous requirements Nonresponsive program design Incomplete or erroneous program design specifications Erroneous decision logic or sequencing Improper program storage or response time Improper handling of interrupts Incorrect module or routine linkages Erroneous arithmetic computations Insufficient accuracy in implementation of algorithm Inaccurate or incomplete comments in prologue Erroneous editing for new version update Incomplete or inconsistent data structure definition Wrong value for constant or preset data Improper scaling of constant or preset data

* Bowen, J. B., "Standard Error Classification to Support Software Reliability Assessment," Proceedings of the National Computer Conference, 1980, pp. 697-705.

Table B-3. Error Categories Found in the Literature (continued)

Authors	Major Error Categories
	Uncoordinated use of data by more than one user Erroneous access or transfer of data Erroneous reformatting or conversion of data Improper masking and shifting of data Failure to initialize flags, counters, data areas New error introduced during correction Noncompliance with programming standards or conventions
Baker (Reference 42)*	Computational errors Logic errors Input/output errors Data handling errors Operating system/system support software errors Configuration errors Routine/routine interface errors Routine/system software interface errors Tape processing interface errors User interface errors Data base interface errors User-requested changes Preset data base errors Global variable/COMPOOL definition errors Recurrent errors Documentation errors Requirement compliance errors Unidentified errors Operator errors Questions Hardware errors Design/requirement logic errors
Fries (Reference 43)†	Logic errors Data handling errors User-requested changes Operator errors Recurrent errors Requirements compliance errors Computational errors

* Baker, W. F., Software Data Collection and Analysis: A Real-Time System Project History, RADC-TR-77-192, June 1977.

† Fries, M. J., Software Error Data Acquisition, RADC-TR-77-130, April 1977.

Table B-4. Project Records Obtained for the IV&V Study

Needed Information	Project 1	Project 2	Project 3	Project 4	Project 5
Technical Results					
Anomaly Reports	x	x	x	x	x
Anomaly Resolution Data	x	x	x	x	x
IV&V Project Data					
Cost	x	x	x	x	-
Schedule	x	x	x	x	x
Man-loading	-	-	x	x	x
Objectives	x	x	x	x	x
Relationship With Developer	x	x	x	x	x
Tools Used	x	x	x	x	x
Participants	x	x	x	x	x
Development Project Data					
Cost	x	x	x	x	-
Schedule	x	x	x	x	x
Man-loading	x	x	x	x	x
Programmer Productivity	-	-	-	-	-
Test Results	x	x	x	x	x
Operational Performance	x	x	-	x	N/A
Maintenance Requirements	x	x	-	N/A	N/A
Programming Practices	x	x	x	x	x

ANOMALY RESOLUTION							Project _____
Report Number	Report Date	Anomaly Description	Anomaly Severity	Report Acceptance	Anomaly Resolution	Resolution Date	

Figure B-4. Typical Anomaly Resolution Form

IV&V ANOMALY REPORT				
Project _____		Analyst _____		
Report No. _____		Subject _____		
Date _____				
Anomaly Type: ☐ Requirements ☐ Design ☐ Code ☐ Documentation				
Anomaly Severity: ☐ High ☐ Medium ☐ Low				
Modules Affected: _____				
Documents Affected: _____				
References: _____				
Description:				
Effects:				
Recommendations:				
Resolution:				

Figure B-5. Typical Anomaly Report Form

IV&V project questionnaires were filled out using Logicon records and IV&V management data supplied by Air Force project officers.

Most of the data-recording activity was devoted to filling out the anomaly questionnaire. This process consisted of the following steps for each anomaly:

- Recording program name, version, anomaly report number, date, analyst, location, and severity
- Writing a short description of the anomaly
- Making judgments as to anomaly category and effects
- Correlating the anomaly report date with IV&V and development schedules to determine the phase in which the anomaly was detected
- Cross-referencing the report to anomaly resolution forms to determine its acceptance and resolution
- Determining, often by inference, the detection method and special circumstances associated with the anomaly

The following paragraphs discuss lessons learned in this process.

B5.1 Severity Ratings

The study originally proposed to classify anomalies into four severity categories--Critical, Serious, Moderate, and Trivial--as had been done in the IV&V study conducted by Dana and Blizzard (Reference 7). All five of the IV&V projects selected for the study, however, used only three levels--High, Medium, and Low--and it was decided to adopt these three levels rather than to try to map the three levels given on the anomaly reports to the four levels proposed for the study.

Upon closer inspection, it turned out that two of the projects--Projects 1 and 2--had actually used six severity ratings: High with nuclear safety implications, High without nuclear safety implications, Medium with nuclear safety implications, and so on. Discussions with project participants revealed that simply ignoring the nuclear safety implications and using the High, Medium, and Low designations would be inaccurate because anomalies with nuclear safety implications are inherently more serious than those without. The participants' recommendation was to use the nonnuclear safety ratings as they were, but to rate High and Medium nuclear safety anomalies as High, and Low nuclear safety anomalies as Medium. This is the approach that was used.

Another interesting discovery was that there were two distinct approaches to assigning severity ratings. One approach asked the question: "How severe would the consequences be if the problem made possible by this anomaly were to occur?" The other asked the dual question: "How likely is it that this anomaly will cause an operational problem, and how severe would the consequences be if the problem occurred?"

The two approaches can result in considerably different severity ratings. Many anomalies concern a remote, yet real, set of circumstances that could affect program operation. These anomalies would be rated higher using the first approach than the second. Anomalies concerned with incorrect documentation, code that is wrong but happens to work correctly in the current version, and other such problems would also be rated differently by the two approaches.

The question that arose for the study was how to resolve the potentially inconsistent ratings for the five projects. Discussions with project participants, however, led to the conclusion that the soundest approach was to use the ratings that were originally assigned, despite their different interpretations. Severity ratings are by nature subjective, and if the IV&V agency, DoD project officer, and developer all agreed to a given rating at the time of the project, that rating reflects project outcome more accurately than one that might be imposed later. Except for the nuclear safety anomalies, therefore, all severity ratings were accepted without change.

B5.2 When is a "Typo" not a "Typo"?

In categorizing documentation anomalies, an attempt was made to differentiate between "substantive" anomalies, such as incorrect, inconsistent, and incomplete facts, and "presentation" anomalies, such as format errors and typographical errors. The distinction turned out to be unclear in the case of typographical errors.

In English text, most typographical errors are easily recognizable as such. A sentence that says "The program shall accept 400 inputs per sacond" may give the reader pause, but is, after a moment's thought, understandable. On the other hand, if the sentence says "The program shall accept 40 inputs per second," where "40" should be "400," a typographical error has turned into a potential development disaster if not caught and corrected.

Because of the wide disparity in the potential effects of typographical errors, each such anomaly was judged on its own merits. General guidelines that emerged were that errors in the typing of numbers, mathematical symbols, variable names, and set/use table entries were regarded as substantive anomalies; those for which interpretation of the intended meaning was relatively clear were classed as presentation problems.

B5.3 Anomaly Effects

For purposes of the study, the effect of each anomaly was at least as important as its cause. During the course of the data collection activity, certain guidelines emerged for determining anomaly effects. The following paragraphs describe these guidelines.

95.3.1 Requirement Specification Anomalies

The impact of a requirement specification anomaly depends upon both the type of problem and the development stage in which it is detected. Anomalies detected before the appearance of design or code were generally considered to

affect program development, verifiability, operation, and maintainability. For anomalies detected after the appearance of design or code, the following guidelines applied:

- If a faulty requirement resulted in faulty design or code, it was considered to have an impact on program development, verifiability, operation, and maintainability.
- If the design and code were correct despite the faulty requirement, the requirement anomaly was considered to have an impact on verifiability and maintainability or on maintainability alone, depending on the anomaly.

B5.3.2 Design Specification Anomalies

The impact of an anomaly in the design specification was also determined by both the type of anomaly and the time at which it was detected. Anomalies detected in the before-code design specification were regarded as design anomalies and were usually considered to have an impact on program development, operation, and maintainability. Usability was sometimes affected; verifiability was usually not since software is tested against requirements rather than design. Anomalies detected in the after-code design specification were treated as follows:

- If the faulty design resulted in incorrect code, the design specification anomaly was considered to affect development, operation, and maintainability.
- If the code was correct, the design specification anomaly was considered to affect maintainability only.

B5.3.3 Code Anomalies

Most code anomalies affected program operation. Anomalies affecting usability included implementations that provided unclear output messages, or cases in which input formats were overly restrictive. Anomalies affecting maintainability included cases of extraneous or inefficient code, incorrect comments, inconsistent implementation, and code that was incorrect but happened to work correctly in the current version.

B5.3.4 User Documentation

Most user documentation anomalies affected program usability. A few were considered to affect maintainability as well.

B5.4 Detection Methods

Of considerable interest to the study were the tools and techniques that resulted in the detection of each anomaly. Unfortunately, this information was not generally contained in the anomaly reports and was therefore unavailable to the study. In many cases, the IV&V technique could be inferred from the IV&V activity in progress at the time of detection. Requirements verifi-

to involve program execution. For code verification, however, detection could have resulted from either manual analysis or use of static analysis tools, so no assumption could be made.

B.6. GENERATING THE IV&V DATA TAPE

The final task of the data collection activity was preparing a magnetic tape containing the data collected. This task consisted of:

- Determining the required tape characteristics from the Data and Analysis Center for Software (DACS), where the tape was to reside
- Designing the tape format
- Selecting an encoding scheme for the data
- Encoding the data
- Transferring the encoded data to magnetic tape
- Generating a listing of the tape contents

The tape and listing were used for subsequent data analysis. At the conclusion of the study they were delivered to RADC.

APPENDIX C

SOFTWARE FEATURES THAT CONTRIBUTE TO MAINTAINABILITY

Specific software features that contribute to maintainability are given below (References 17, 19, 44*).

Area	Feature
Requirements	Correct requirements
	Consistent requirements
	Complete requirements
	Testable requirements
	Inclusion of traceability information
Design	Allowance for excess computer capacity
	Top-down design
	Modular design
	Modules of limited size
	Single function for each module
	Separate modules for input, output, computation
	Single entry, single exit for all modules, except for certain computer interrupts and error-condition exits
	Initialization and housekeeping functions internal to the modules needing them
	Only control modules able to make abort decisions
	Communication between modules limited to defined interfaces
	All control data passed only through defined interfaces
	Coherent conceptual organization
	Consistent application of design principles

*Stanfield, J. R., and Skrukrud, A. M., Software Acquisition Management Guidebook: Software Maintenance, ESD-TR-77-327, Oct. 1977.

Area	Feature
Design (continued)	Centralized data base
	Controlled data base
	Limited access to data base by each module
	Procedures to define and control data base entries
	Module and data base interfaces not overly complex
	Data base designed for expansion and change
	Data base symbolically defined
	Limited equipment interfaces
	Machine dependencies isolated and encapsulated
	Allowance for future extensions
Code	Self-monitoring features
	Full implementation of design features that contribute to maintainability
	Maintaining a reasonable storage and time margin
	Use of a single higher order language if possible
	No assembly language embedded in other code unless explicitly called for
	Adherence to module size constraints
	Use of structured programming
	Use of blank cards to set off functional blocks visually
	Use of indentation to reflect block structure
	Use of general comments preceding each module
	Use of adequate comments to identify flow of control and purpose of each section
	Adequate commenting of time-sensitive areas to alert maintainers

Area	Feature
Code (continued)	Use of symbolic parameters for constants and basic data structure sizes
	Use of subroutine arguments rather than global common
	Use of named common
	No sharing of variables or temporary storage locations
	No self-modifying code
	No absolute addressing
	No relative addressing
	No embedded constants or literals
	Symbolic, meaningful data references
	No code that implicitly couples one module to another
	Avoidance of dynamic allocation of resources
	Avoidance of unnecessarily complex arithmetic and conditional statements
	Avoidance of recursive/reentrant coding
	Avoidance of unnecessarily complex logical structures
	Consistency in design implementation, I/O processing, error processing, module interfacing, module/variable naming
	Inclusion of test aids or implementation to support their use
Documentation	Complete, accurate description of the program as coded
	Incorporation of all design changes made during coding and testing
	Time-sensitive portions of code clearly identified and described
	Modular organization
	Consistency in detail and style
	Emphasis on ease of use

Area	Feature
Documentation (continued)	Inclusion of objectives and assumptions
	Avoidance of complexity
	Allowance for expandability and ease of change
Configuration Management	Accurate status records for software, documentation, and changes
	Accurate date and version indicators in source listings and documentation
	Adequate control and documentation of changes during development
	Consistent numbering schemes to relate corresponding source listings, documentation, status records
	Controlled use of program patches

REFERENCES

1. Nie, N. H., et al., Statistical Package for the Social Sciences, McGraw Hill, 1975.
2. Rubey, R. J., "Quantitative Aspects of Software Validation," Proceedings of the International Conference on Reliable Software, April 1975, pp. 246-251.
3. Boehm, B. W., et al., "Characteristics of Software Quality," TRW Software Series TRW-SS-73-09, Dec. 1973.
4. Proceedings of the TRW Symposium on Reliable, Cost-Effective, Secure Software, March 1974, pp. 5.13-5.17.
5. Thayer, T. A., et al., Software Reliability Study, RADC-TR-76-238, Feb. 1976, A030798.
6. Shooman, M. L., and Bolsky, M. I., "Types, Distribution, and Test and Correction Times for Programming Errors," Procedures of the International Conference on Reliable Software, April 1975, pp. 347-357.
7. Dana, J. A., and Blizzard, J. D., The Development of a Software Error Theory To Classify and Detect Software Errors, Logicon Report HR-74012, May 1974.
8. Finfer, M. C., Software Data Collection Study, Volume III: Data Requirements for Productivity and Reliability Studies, RADC-TR-76-329 - Vol. III, June 1976, A036064.
9. Radatz, J. W., Ramsey, O.C., and McKillop, T. L., NSCCA/PATE Guidebooks, Volume III, Logicon Report R:SED-80204-III, June 1980.
10. Miller, C. R., "Software Maintenance and Life Cycle Management," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 53-59.
11. Fife, D. W., "Software Management Standards," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 63-80.
12. Prokop, J., Computers in the Navy, Annapolis, MD, Naval Institute Press, 1976.
13. Robinson, D. G., "Beyond the Four Stages: What Next," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 187-201.
14. McGonagle, J. D., A Study of a Software Development Project, James P. Anderson and Co., Sept. 1971.

15. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp. 48-59.
16. Lehman, M. M., "Evolution Dynamics--A Phenomenology of Software Maintenance," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 313-323.
17. Peercy, D. E., "A Software Maintainability Evaluation Methodology," Proceedings of the AIAA 2nd Computers in Aerospace Conference, Oct. 1979, pp. 315-325.
18. Boehm, B. W., "Software Engineering," IEEE Transactions on Computers, Dec. 1976, pp. 1226-1241.
19. Neil, G., and Gold, H. I., Software Acquisition Management Guidebook: Software Quality Assurance, ESD-TR-77-255, Aug. 1977.
20. Georghiou, D. L., Guidelines for Programming in Portable Fortran, Logicon Report No. DS-R78069, Sept. 1978.
21. Donahoo, J. D., A Review of Software Maintenance Technology, RADC-TR-80-13, Feb. 1980, A082985.
22. Basili, V. R., and Zelkowitz, M. V., "Analyzing Medium-Scale Software Development," Proceedings of the Third International Conference on Software Engineering, May 1978, pp. 116-123.
23. Doty, D. L., Nelson, P. J., and Stewart, K. R., Software Cost Estimation Study, Volume II: Guidelines for Improved Software Cost Estimating, RADC-TR-77-220 - Vol. II, Feb. 1977, A044609.
24. Brooks, F. P., "The Mythical Man Month," Datamation, Dec. 1974, pp. 45-52.
25. Nanus, B., and Farr, L., "Some Cost Contributors to Large Scale Programs," Proceedings - Spring Joint Computer Conference, 1964, pp. 239-248.
26. Finfer, M. C., and Mish, R., Software Acquisition Management Guidebook: Cost Estimation and Measurement, ESD-TR-78-140, Mar. 1980.
27. Walston, C. E., and Felix, C. P., "A Method of Programming Measurement and Estimation," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 153-174.
28. Office of the Secretary of Defense, Embedded Computer Resources and the DSARC Process--A Guidebook; Part IV: Software Estimating Guidelines, 1977.
29. Love, T., "Software Psychology: Shrinking Life-Cycle Costs," Software Phenomenology--Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 605-625.

30. Wolverton, R. W., "Cost Estimating Algorithm," Proceedings of the IEEE Computer Software and Applications Conference, Nov. 1977, pp. 205-245.
31. Schwartz, J., "Resource Estimation," Software Phenomenology -- Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 117-130.
32. Norden, P. V., "Project Life Cycle Modelling: Background and Application of the Life Cycle Curves," Software Phenomenology -- Working Papers of the Software Life Cycle Management Workshop, Airlie House, Aug. 1977, pp. 217-227.
33. Black, R. K. E., "Effects of Modern Programming Practices on Software Development Costs," Proceedings of the 15th IEEE Computer Society International Conference, Sept. 1977, pp. 250-253.
34. Wolverton, R. W., "The Cost of Developing Large-Scale Software," IEEE Transactions on Computers, June 1974.
35. Herndon, M. A., and Keenan, A. P., "Analysis of Error Remediation Expenditures During Validation," Proceedings of the Third International Conference of Software Engineering, May 1978, pp. 202-205.
36. Miyamoto, I., "Software Reliability in On-Line Real Time Environment," Proceedings of the International Conference on Reliable Software, April 1975, pp. 194-197.
37. Weinberg, G. M., "The Psychology of Improved Programming Performance," Datamation, Nov. 1972.
38. Hartwick, R. D., Software Acceptance Criteria Panel Report, Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management, Software Workshop, April 1979.
39. Amory, W., and Clapp, J. A., Engineering of Quality Software Systems (A Software Error Classification Methodology), RADC-TR-74-324 - Vol. VII, Jan. 1975.
40. Endres, A., "An Analysis of Errors and Their Causes," Proceedings of the International Conference on Reliable Software, April 1975, pp. 327-336.
41. Bowen, J. B., "Standard Error Classification to Support Software Reliability Assessment," Proceedings of the National Computer Conference, 1980, pp. 697-705.
42. Baker, W. F., Software Data Collection and Analysis: A Real-Time System Project History, RADC-TR-77-192, June 1977, A041644.
43. Fries, M. J., Software Error Data Acquisition, RADC-TR-77-130, April 1977, A039916.
44. Stanfield, J. R., and Skrukrud, A. M., Software Acquisition Management Guidebook: Software Maintenance, ESD-TR-77-327, Oct. 1977.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.