NUREG/CR-6316
SAIC-95/1028
Vol. 1

Guidelines for the Verification and Validation of Expert System Software and Conventional Software

RECEIVED
APR 21 1995
OSTI

Project Summary

Prepared by
L. A. Miller, J. E. Hayes, S. M. Mirsky

Science Applications International Corporation

Prepared for
U.S. Nuclear Regulatory Commission

and

Electric Power Research Institute

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

AVAILABILITY NOTICE

Availability of Reference Materials Cited in NRC Publications

Most documents cited in NRC publications will be available from one of the following sources:

1. The NRC Public Document Room, 2120 L Street, NW., Lower Level, Washington, DC 20555-0001
2. The Superintendent of Documents, U.S. Government Printing Office, P. O. Box 37082, Washington, DC 20402-9328
3. The National Technical Information Service, Springfield, VA 22161-0002

Although the listing that follows represents the majority of documents cited in NRC publications, it is not intended to be exhaustive.

Referenced documents available for inspection and copying for a fee from the NRC Public Document Room include NRC correspondence and internal NRC memoranda; NRC bulletins, circulars, information notices, inspection and investigation notices; licensee event reports; vendor reports and correspondence; Commission papers; and applicant and licensee documents and correspondence.

The following documents in the NUREG series are available for purchase from the Government Printing Office: formal NRC staff and contractor reports, NRC-sponsored conference proceedings, international agreement reports, grantee reports, and NRC booklets and brochures. Also available are regulatory guides, NRC regulations in the *Code of Federal Regulations*, and *Nuclear Regulatory Commission Issuances*.

Documents available from the National Technical Information Service include NUREG-series reports and technical reports prepared by other Federal agencies and reports prepared by the Atomic Energy Commission, forerunner agency to the Nuclear Regulatory Commission.

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions. *Federal Register* notices, Federal and State legislation, and congressional reports can usually be obtained from these libraries.

Documents such as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings are available for purchase from the organization sponsoring the publication cited.

Single copies of NRC draft reports are available free, to the extent of supply, upon written request to the Office of Administration, Distribution and Mail Services Section, U.S. Nuclear Regulatory Commission, Washington, DC 20555-0001.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at the NRC Library, Two White Flint North, 11545 Rockville Pike, Rockville, MD 20852-2738, for use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from the American National Standards Institute, 1430 Broadway, New York, NY 10018-3308.

DISCLAIMER NOTICE

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party's use, or the results of such use, of any information, apparatus, product, or process disclosed in this report, or represents that its use by such third party would not infringe privately owned rights.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Guidelines for the Verification and Validation of Expert System Software and Conventional Software

Project Summary

Manuscript Completed: February 1995
Date Published: March 1995

Prepared by
L. A. Miller, J. E. Hayes, S. M. Mirsky

Science Applications International Corporation
1710 Goodridge Drive
McLean, VA 22102

Prepared for
Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code L1530

and

Nuclear Power Division
Electric Power Research Institute
3412 Hillview Avenue
Palo Alto, CA 94303

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
ra

MASTER

ABSTRACT

This eight-volume report presents guidelines for performing verification and validation (V&V) on Artificial Intelligence (AI) systems with nuclear applications. The guidelines have much broader application than just expert systems; they are also applicable to object-oriented programming systems, rule-based systems, frame-based systems, model-based systems, neural nets, genetic algorithms, and conventional software systems. This is because many of the components of AI systems are implemented in conventional procedural programming languages, so there is no real distinction. The report examines the state of the art in verifying and validating expert systems. V&V methods traditionally applied to conventional software systems are evaluated for their applicability to expert systems. One hundred fifty-three conventional techniques are identified and evaluated (Section 2.2). These methods are found to be useful for at least some of the components of expert systems, frame-based systems, and object-oriented systems. A taxonomy of 52 defect types and their detectability by the 153 methods is presented. With specific regard to expert systems, conventional V&V methods were found to apply well to all the components of the expert system with the exception of the knowledge base. The knowledge base requires extension of the existing methods. Several innovative static verification and validation methods for expert systems have been identified and are described here, including a method for checking the knowledge base "semantics" and a method for generating validation scenarios. Evaluation of some of these methods was performed both analytically and experimentally. A V&V methodology for expert systems is presented based on three factors: (1) a system's judged need for V&V (based in turn on its complexity and degree of required integrity); (2) the life-cycle phase; and (3) the system component being tested. 16 guideline packages and eleven procedures which describe the recommended V&V methods to apply for a given system are presented for the various combinations of the three factors. The knowledge base verification methodology was utilized for two sample nuclear expert systems, and was found to be highly effective.

TABLE OF CONTENTS

ABSTRACT	iii
EXECUTIVE SUMMARY	xi
1. INTRODUCTION	1
1.1 Background	1
1.2 Technical Approach	1
1.3 Report Organization	2
2. SURVEY OF V&V METHODS	5
2.1 Definition of Systems in Terms of V&V Classes	5
2.2 Classification of Conventional and Expert System V&V Methods	7
2.2.1 Requirements/Design Techniques	7
2.2.2 Static Testing Methods	13
2.2.3 Dynamic Testing Methods	13
2.3 Characterization/Ranking of V&V Methods	19
2.3.1 Characterization in Terms of Faults Detected	19
2.3.2 Characterization in Terms of Eight Factors: Cost-Benefit Effectiveness and Measures	19
2.4 Assessment of Applicability of Methods to Expert System Components	41
2.4.1 Components of Expert Systems	44
2.4.2 Key V&V Characteristics of Expert System Components	44
2.4.3 Applicability of V&V Methods	44
3. KNOWLEDGE BASE CERTIFICATION	59
3.1 Methods Chosen For Evaluation	59
3.2 Experimental Design	
3.2.1 The four methods	62
3.2.1.1 Requirements Tracing Method	62
3.2.1.2 Syntax-Checking Method	64
3.2.1.3 Knowledge-Checking Method	65
3.2.1.4 Desk-Checking	66
3.2.2 Faults	66
3.2.3 Training	66
3.2.4 Hypotheses Under Test	67
3.3 Experimental Results	67
4. VERIFICATION AND VALIDATION GUIDELINES FOR ARTIFICIAL INTELLIGENCE SYSTEMS	73
4.1 Factors for Selecting Verification and Methods	73
4.1.1 Applications and Systems Covered	73
4.1.2 V&V Methods and the Developmental Life Cycle	74
4.1.3 Management Aspects of V&V	74
4.1.4 Risk Management Approach	76
4.1.4.1 Complexity, Faults, Failures, Risk, and Required Integrity	76

4.1.4.2	System Failures, Function of Life Cycle Phase	76
4.1.4.3	Three Classes of V&V	77
4.1.4.4	Quality Assurances for Each V&V Class	77
4.2	Guidelines	77
4.2.1	Guideline Packages	77
4.2.2	Guideline Procedures	78
5.	EVALUATION OF V&V METHODS AS APPLIED TO NUCLEAR EXPERT SYSTEMS	111
5.1	Evaluation of V&V Guidelines for Application to Systems A and B	111
5.2	Methods as Applied to System A	112
5.3	Methods as Applied to System B	119
5.4	Method Ranking by System A and B Developers/Users	122
5.5	Conclusions	122
6.	VALIDATION SCENARIO GENERATION	125
6.1	Industry Practices	125
6.2	Classification and Characterization of Scenarios	126
6.3	Methodology For Selecting Scenarios	128
6.3.1	Validation Scenarios for New Software	129
6.3.2	Validation Scenarios for Modified Software	129
6.3.3	Selection Philosophy	130
6.4	Methodology Applied To Systems A and B	130
6.5	Novel Method for Knowledge-Based Scenario-Generation	134
7.	OTHER V&V ACTIVITIES	137
7.1	V&V of Frame-Based Systems	137
7.1.1	Description of Frames	137
7.1.2	Errors and Anomalies in Frames	137
7.1.3	Detecting Frame Anomalies	141
7.2	V&V of Object-Oriented Systems	142
7.2.1	Approach to the Problem of V&V of Class/Object Hierarchies	142
7.2.2	Failure of Object-Oriented Class/Object Hierarchies	144
7.2.3	Detecting Object-Oriented Failure	144
7.3	V&V Activities in Professional Organizations	144
7.3.1	IEEE Activities	149
7.3.2	AIAA Activities	150
7.3.3	AAAI Activities	150
7.3.4	International Conferences	150

8. SUMMARY AND CONCLUSIONS	151
9. OBJECT-ORIENTED REFERENCES	155
10. GLOSSARY	157

LIST OF FIGURES

Figure ES-1.	Activity Dependencies and Interrelationships	xiii
Figure 2.1-1.	Three Classes of Stringency of Recommended V&V with Examples of Expert Systems for the Nuclear Power Industry	8
Figure 2.2-1.	Classes of Conventional V&V Methods Organized by Life-Cycle Phase	9
Figure 3.3-1.	Representative Significant Results for System and Condition	70
Figure 4.2-1.	Questionnaire To Determine the Recommended Class of V&V	102
Figure 4.2.2-1.	Sample of V&V Guideline Procedure	109
Figure 7.1.2-1	Example of Person-Concept Frames Based on Age	139

LIST OF TABLES

Table 2.1-1.	Six Factors of Software System Complexity	6
Table 2.2.1-1.	Description of the Conventional Requirements/Design V&V Methods	10
Table 2.2.2-1.	Description of the Conventional Static Testing V&V Methods	14
Table 2.2.3-1.	Description of the Conventional Dynamic Testing V&V Methods	21
Table 2.3.1-1.	Types of Software Faults	28
Table 2.3.1-2.	Capability of Testing Techniques to Detects	32
Table 2.3.1-3.	Applicability of Conventional Techniques to Defects Found in Conventional Software	39
Table 2.3.2-1	Interpretation of the 1-5 Rating Scale Value for Each of the Eight Cost Benefit Factors	42
Table 2.3.2-2.	Conventional V&V Techniques, Their Power and Ease-of-Use Factor Ratings, and the Cost-Benefit and Effectiveness Measures	45
Table 2.3.2-3.	Overall Highest Ranked Conventional V&V Techniques for all V&V Classes	49
Table 2.4.2-1.	Components and Typical Testing-Related Features of Knowledge-Based Systems, with Testing Recommendations	49
Table 2.4.3-1.	Rating of the Applicability of Conventional Techniques to Expert Systems and Their Components	51
Table 3.1-1.	Rank-Ordering of the Top 10 Rated Static-Testing Techniques for each of the Cost-Benefit Metric and the Extreme Effectiveness Metrics	52
Table 3.2-1.	Experiment Participant Key Qualifications and Group Assignment	61
Table 3.31	Representative significant Results for System and Condition	63
Table 4.1.1-1.	Three Types of AI and Conventional System Components and the Recommended V&V Philosophy for Each	75
Table 4.1.4.2-1.	Types of Failure that can Occur During the Requirements Stage	80

Table 4.1.4.2-2.	Decomposition of Identified Failure for the Requirements Stage	81
Table 4.1.4.2-3.	Types of Faults that can Occur During the Design Stage	83
Table 4.1.4.2-4.	Decomposition of Identified Faults for the Design Stage	84
Table 4.1.4.2-5.	Types of Failure that can Occur During the Implementation Stage	87
Table 4.1.4.2-6.	Decomposition of Identified Faults for the Implementation Stage	88
Table 4.1.4.4-1.	Types of Assurances Provided by Each V&V Class and Appropriate V&V Methods: at the Requirements Stage	94
Table 4.1.4.4-2.	Types of Assurances Provided by Each V&V Class and Appropriate V&V Methods: at the Design Stage	96
Table 4.1.4.4-3.	Types of Assurances Provided by Each V&V Class and Appropriate V&V Methods: at the Implementation Stage	99
Table 4.2.1-1.	Identification of the 15 Guideline Packages	104
Table 4.2.1-2.	Sample Listing of Recommended V&V Methods	107
Table 4.2.1-3	Sample Guideline Plan	108
Table 5.1-1.	Selection of V&V Techniques Appropriate for System A and System B	113
Table 5.1-2.	Suggested Methods for V&V of Systems A and B	116
Table 5.2-1.	Summary of RPV Control Text Document Labeling	117
Table 5.2-2.	Summary of Systems A and B Rule Labeling	118
Table 5.4-1.	Evaluation of V&V Methods	124
Table 7.1.2-1	Summary of Frame-Based Anomalies	143
Table 7.2.2-1	Name and Description of 32 Object-Oriented Class/Object Hierarchy Faults	145

EXECUTIVE SUMMARY

This report presents the results of the Expert System Verification and Validation (V&V) project which is jointly funded by the United States Nuclear Regulatory Commission (USNRC) and the Electric Power Research Institute (EPRI). The ultimate objective of this project is the formulation and documentation of a draft set of guidelines for V&V of Artificial Intelligence (AI) systems in the nuclear power industry. Most of the techniques used for AI systems are equally suited to all software.

The motivating concern behind this project involves the realization that artificial intelligence systems are more and more being introduced into use for nuclear power plants. EPRI itself has been a major developer of AI systems for nuclear applications. The movement toward digital instrumentation and controls will also likely increase the use of AI systems. There has been little effort made toward developing means to assure their reliability, particularly when integrated into existing conventional plant software. For software, the reliability assurance responsibility is known as software verification and validation (V&V) or Independent (IV&V), to emphasize the fact that it is accomplished by an organization outside of the development group. Thus, the focus goal of this project is the development of adequate V&V procedures for AI systems, primarily for use in the nuclear utility industry. These procedures are also beneficial for use in other industries.

The joint sponsorship of the work by both the USNRC and EPRI reflects the fact that in the absence of AI system V&V guidelines and recommended practices, neither could the utilities have guidance on how to evaluate the integrity of systems they developed nor could the USNRC have a technical basis for the safety evaluation of these systems. Clearly, some initial guidance for both was needed. There are three levels of direction which could be provided, arranged in decreasing order of formality and stringency: Standards, Recommended Practices, and Guidelines. While Standards and Recommended Practices are judged to be inappropriate for AI systems at this stage, the development of Guidelines, the third and lowest level, is a reasonable objective. Accordingly, the two sponsors agreed to jointly fund a project to develop guidelines for the V&V of AI systems and ensured, through involved technical management, that the interests of both parties were well reflected not only in the initial project goals but also in the details of the performance of the several project activities. The final product, a set of V&V guidelines and detailed procedures, is intended not as a stopping point but rather as a firm beginning on the road to developing and formalizing a consensus concerning procedures for assuring that AI systems introduced into nuclear utility environments will be safe and reliable. Feedback from industry and academia will play an important role in enhancing and improving these guidelines. These guidelines should truly be considered as guidance based on technical judgment, not as requirements.

The technical approach for this project included several facets. All possible sources of Verification and Validation methods were examined: those for AI systems, those for conventional software systems, and those that were innovative. Once the methods were identified or innovative methods were developed, they were "instantiated" on real expert systems. Four primary components of expert systems were identified as the inference engine, external interfaces, tools and utilities, and knowledge base. Each of the components were evaluated in terms of the degree to which conventional techniques were applicable for V&V of that component. In general, the conventional V&V methods were found to be applicable to the first three components of AI systems. The knowledge base was the one component which required new techniques or extension of existing conventional techniques.

A key aspect of the technical approach was the emphasis on quantification. An attempt was made to quantify evaluations of methods since this is currently a highly subjective area. For example, an experiment using behavioral methods was conducted to determine the effectiveness of various V&V methods in detecting errors in knowledge bases. A second aspect concerns consensus. To ensure that the research being performed was on target and would be of use to

parties performing V&V on expert systems, feedback was actively sought (and obtained) from users, sponsors, and industry (both software and nuclear industries).

Another key facet of the technical approach for this project was the emphasis on practicality and cost-effectiveness. Whenever appropriate, emphasis was placed on existing experience. For example, brand new approaches suggested by universities were de-emphasized in favor of "tried and true" approaches. Since the V&V methodology developed in this project will have to coexist within the nexus of existing standards, over 100 such standards were reviewed as part of this work.

The technical approach focused on specific software faults and methods that could detect those faults as opposed to undirected general testing. An attempt was also made to identify automated support tools for V&V to ease implementation of the methods. In general, an emphasis was placed on increasing the usability of the V&V methodology and guidelines, with the realization that unusable guidelines would realize that fate.

A further key aspect of the technical approach was assuring that it catered to the differing needs of different systems. The very wide range of systems to be subjected to V&V was taken into consideration and led to the identification of three classes of V&V based on system complexity and degree of required integrity. Finally, the team composition for this project was also an important facet of the technical approach. Individuals with backgrounds in AI systems, V&V, and the nuclear industry were brought together to ensure broad applicability.

It is important to note that various assessments and recommendations in the report are based, in part, on subjective evaluations both of the frequency and types of failure that can occur and of the effectiveness of various methods to detect these failures. There is a great need to capture and analyze the failures of expert systems (in particular) that do occur and to obtain empirical data on the effectiveness of various methods to identify these failures. Similarly, there is the important step of validating the effectiveness of the recommended V&V guideline packages. Only by these means can the basis for these findings be transitioned into sound engineering practice.

The AI Systems V&V project consisted of ten separate but interrelated activities. The activities were "re-organized" so that cornerstone activities would be performed first, and other activities could build upon that work. Figure ES-1 depicts the activities flow for this project. For example, keystone Activities 1 and 2 which identified the conventional and expert system V&V methods currently in use had to be completed in order for later activities to commence.

Figure ES-1 TASK DEPENDENCIES AND INTERRELATIONSHIPS

The major accomplishment of this project was its central goal -- the development of a set of V&V guidelines to be used to assure the quality of expert systems, and other AI systems, developed for the nuclear utility industry. These guidelines provide recommended information about the following:

- What issues are important in deciding how much V&V to perform;
- When V&V may be performed;
- What components (as well as the overall system) it may be performed on;
- For specific combinations of the above, what set of V&V methods are suggested;
- In what order the methods should be applied;
- Who should perform the investigation, with what qualifications;
- What metrics should be gathered as a result of applying the V&V methods;
- What decisions should be made, and what should the decision thresholds be;
- And finally, for 11 of the more important techniques, specific procedures laying out step by step how to apply the method.

This final report describing the activities and results of this project is organized into several volumes. This first volume represents the cumulative technical report which summarizes all the activities completed for the project. In addition, Volume 1 covers some limited work in standards and object-oriented programming systems. The second volume presents the survey of conventional V&V methods (Activity 1), and the third volume describes the survey of expert system V&V methods currently in use (Activity 2). Volume 4 presents the results of the knowledge-base certification experiment performed as part of Activity 6. Volume 5 describes the V&V guidelines developed as part of Activity 7, and Volume 6 presents an examination of validation scenarios from Activity 8. All of these volumes are described below. Volume 7 presents the User's Manual for the V&V guidelines. This manual assists a software development project manager or V&V manager in performing V&V on expert systems. Finally, Volume Eight presents the very extensive bibliography for the entire project.

Activity 1 - Survey and Assessment of Conventional Software Verification and Validation Techniques (Volume 2). The first activity consisted of a review of software engineering techniques for conventional software systems and an assessment of their applicability for expert systems. A total of 153 different conventional software V&V techniques were identified. A taxonomy of 52 different types of conventional software faults was developed. The V&V methods were judged as to which of the faults they might detect. Each of the techniques covered anywhere from 2 to 52 of these faults. Each fault was covered by anywhere from 21 to 50 V&V techniques. Therefore, the conventional V&V techniques taken together cover the total space of identified conventional software system faults. The reader may note overlap between Volumes 1, 5, and 7. This parallels the structure of the project.

Volume 1 presents the cumulative technical report for this project. Volume 5 describes all the considerations that went into developing the V&V guidelines, and Volume 7 presents the guidelines themselves.

A classification scheme to assign various combinations of complexity and required integrity levels into three classes of recommended software V&V was developed. The scheme placed high complexity software systems that require high integrity in V&V Class 1, and systems requiring the least stringent V&V levels in Class 3. The conventional V&V methods were classified by a sequential Life-cycle model, characterized by different factors of power and ease-of-use. The factors used to characterize the power of the methods were: broad power, hard power, formalizability, and human-computer interface testability. The ease-of-use factors were: ease of mastery, ease of setup, ease of running/interpretation, and ease of usage. These factors are defined in Section 2.3.2. A cost/benefit and an effectiveness measure based on these eight factors were developed. This permitted identification of the top-rated V&V techniques for all phases of the life-cycle. The four primary components of expert systems identified were inference engine, knowledge base, external interfaces, and tools and utilities. Each of the components were evaluated in terms of the degree to which conventional techniques were applicable for V&V of that component.

Conventional V&V techniques were collectively and individually judged to be highly applicable to the expert systems as a whole and to the inference engine, external interfaces, and the tools and utilities, in particular. However, the methods were judged to be inadequate (in present form) to sufficiently evaluate the knowledge base component. This is particularly true for the more stringent V&V classes. However, the conventional V&V methods should be extendable for this purpose.

Activity 2 - Survey and Document Expert System V&V Methodologies (Volume 3). The second activity involved contacting a number of AI system development organizations to determine what methods they used to V&V AI systems. The survey included over 130 telephone interviews, site visits to nine institutions conducting research in or applying expert system V&V, and collection of an extensive bibliography with well over 300 references.

Many conventional V&V techniques are being researched for or applied to expert systems, particularly in the areas of Static Testing and Dynamic Testing. Formal techniques are infrequently applied during the Requirements and Design phases. In the survey, Requirements and Design documents for expert systems were reported often to be non-existent and therefore, unfortunately, cannot be used for V&V activities. (Even the two expert systems studied in this project lacked these documents; requirements specifications had to be developed after the fact to support various analysis activities.)

Most of the work in Static Testing of expert systems has focused on development of automated tools which perform syntax checking of rule bases. Some rule base checkers will perform semantic checks of the rule base using knowledge constraints defined by the programmer, while others perform checking "on the fly" during knowledge acquisition and/or rule base refinement.

Concerning Dynamic Testing, Ad hoc Testing (judged a poor method in Activity 1) still represents the single most used V&V technique for the operational AI system world.

Classification of the methods used in terms of the types identified in Activity 1 revealed only 4 new techniques, and these were easily accommodated in the Activity 1 taxonomy. There appears to be sufficient coverage by the conventional and the new V&V techniques across all the components and all the error types. The challenge is to select the appropriate combination of techniques that is cost efficient and effective for performing V&V on an expert system.

Activity 3 - Test Applicability and Limitations of Conventional Verification Methods (no separate volume, see Section 5 of Volume 1). Activity 3 examined, tested, and documented the applicability and limitations of conventional verification methods for expert systems using two sample systems, System A and System B. System A assists boiling water reactor (BWR) operators in monitoring and carrying out emergency operating procedures (EOPs) during

abnormal reactor events and accidents. System B is intended to aid the Reactor Safety Team at the Emergency Operations Center to monitor and project core status during an emergency at a licensed nuclear power plant. The applicability to expert systems of most of the conventional software V&V methods could easily be determined from the Activity 1 and 2 survey results. This was not the case for Requirements Tracing. Therefore, Requirements Tracing was applied directly to both System A and System B.

Application of Requirements Tracing led to several conclusions. First, a Requirements Document is essential to good V&V. Second, V&V techniques are most cost effective when performed early in the Life-cycle. Third, the V&V program should be Life-cycle-based as opposed to post-implementation testing. It was shown that ad hoc post-implementation testing can leave many undiscovered errors as well as being less cost effective. Finally, Requirements Tracing was shown to be highly effective and cost efficient, even when done post hoc.

Activity 4 - Guidelines for the Verification and Validation of Artificial Intelligence Software Systems (no separate volume, see Section 4 of Volume 1). The fourth activity of the contract followed Activity 6 and presented a draft of the guidelines for verification and validation of artificial intelligence expert systems. The types of applications and systems covered directly by the draft guidelines includes expert systems, rule-based systems, frame-based systems, model-based systems, object-oriented systems (by extension, neural networks and genetic algorithms are also covered) and conventional software systems. This is because many of the components of AI systems are implemented in conventional programming languages. It should be noted that the guidelines are subjective, and that extensive industry use will be required before they can become an accepted practice.

A brief overview of the V&V techniques identified in Activities 1 and 2 was presented. The developmental Life-cycle was described to assist the users of the guidelines in determining what their current Life-cycle phase is. Though no methods were found specifically designed for the Maintenance phase, all methods presented will be useful for that phase. Maintenance involves the re-visiting of all the phases of the Life-cycle, so therefore these guidelines apply to the V&V of already developed as well as evolving systems. Various management aspects of V&V such as Quality Assurance, Configuration Management, and software engineering practices were presented. A discussion of the importance of "front-loading" the V&V activity (for cost savings) was presented. Risk management, as implemented by the V&V guidelines, was discussed at length.

The V&V guidelines consisted of a section describing how to determine what phase of the Life-cycle a software system is in, what component of the system is subject to V&V, how to determine the class of V&V required for a system, and how to determine which guideline package/procedure to use. The guideline packages describe what methods to perform and the metrics which should be used to determine the success of the method. Detailed step-by-step guideline procedures were provided for a subset of the V&V techniques, providing step-by-step instructions on how to implement a given method. The final version of the guidelines can be found in Volume 5.

Activity 5 - Test Verification Methodology (no separate volume, see Section 5 of Volume 1). This activity involved the application of the draft guidelines from Activity 4. The determination of the level of required V&V for a system was examined. The system component categories were also examined, as were the life-cycle phase determination instructions. Site visits were made to observe both Systems A and B and to gather information on the V&V methods already applied to these systems. An examination was provided of the V&V methods already applied to each of the two expert systems and those planned for future application.

Application of the V&V methodology from Activity 4 to Systems A and B led to several conclusions. First, the V&V methodology postulated in Activity 4 was further validated by the System A and B development and test teams. Second, almost half of the V&V methods recommended for use on Systems A and B had already been applied and were

evaluated and found to be useful. Third, the System A and B teams assisted in determining scores for Power, Ease-of-Use, Relative Cost-Benefit, and Effectiveness for these techniques. Finally, some of the V&V methods applied to Systems A and B were found to be possibly even more cost-beneficial and effective than originally supposed.

Activity 6 - Knowledge Base Certification (Volume 4). Activity 1 identified the knowledge base as the one expert system component which could not be adequately certified by current conventional software V&V methods. Activity 6 presented the first known behavioral evaluation of the V&V of any type of Artificial Intelligence software system. It provided empirical evidence that knowledge base certification methods are effective in helping people find problems with knowledge bases.

Two real nuclear expert systems, denoted as System A and System B, were used in the knowledge base certification method experiment. The three-day experiment involved twenty participants from nuclear power plant operating utilities, the USNRC Technical Training Center, University of Maryland Graduate Center, EG&G, and SAIC. Ten participants were blindly assigned to each of the two expert systems. These groups were further divided into groups of five, with one being a control group and the other the experimental group using new methods and tools. The groups were carefully composed to ensure that the same mix of utility and non-utility participants with a range of applicable nuclear and testing experience were included in each group. Designed errors of varying types and difficulty of detection were seeded into the two expert systems for evaluating the various knowledge base certification methods.

Four separate tasks were completed by each of the groups: requirements grouping, requirements tracing, syntax checking, and knowledge checking. All of these are static analysis techniques. The selection criteria and methodology defined in Activity 1 was used to select these methods as being the best and most appropriate to study from over a hundred possible V&V techniques. A set of new automatable tools was used by the experimental group for each of these tasks. Three of the four tools were designed as part of the project for the express purpose of certifying the knowledge base component. These three tools were not implemented, but their output for the error-seeded knowledge bases was simulated for use by the participants. The first two were detailed printouts from a requirements tracing tool called SuperTrace. The third was a simulated tool for examining the knowledge in the knowledge base using meta-knowledge, called MetaCheck. The fourth tool was an automated tool for syntax checking that is still experimental. The tools were used by the experimental group while the control group used traditional inspection (desk-checking) to perform the same four tasks.

The experimental results led to several conclusions. First, the use of tools or aids results in a significant improvement in detecting all types of faults, avoiding false alarms, and completing the effort in less time. Second, the knowledge checking tool showed the greatest improvement in performance with the requirements tracing tool being second. Third, testing and extensive subject matter experience (7 to 22 years) did not improve the performance of the participants as compared with less testing and subject matter experience (2 to 4 years). Finally, there were still seeded errors that went undetected by all participants with and without tools; percentage of seeded errors found ranged from 45%-73%. These were the most complex and were not expected to be picked up by the tools. Other methods or tools may therefore be warranted to augment the knowledge base certification methods.

Activity 7 - Guidelines for the Verification and Validation of Artificial Intelligence Software Systems (Volume 5). Activity 7 represented the final version of the V&V guidelines. Several updates from Activity 4 were made to this report. The guideline section was updated to provide a "roadmap" of how to use the guidelines. Standard decision thresholds used in the guideline packages and procedures were defined. The cost-benefit tradeoffs inherent in performing V&V on an expert system was discussed. Measures used in the guideline packages (majors, show-stoppers, minors) were defined. Finally, several new guideline packages and procedures were provided with Activity 7.

Activity 8 - Validation Scenarios (Volume 6). Activity 8 included a review of the current state-of-the-art in developing validation scenarios for nuclear power industry-related computer software. The review involved telephone discussions and a literature assessment which included nuclear industry, consultant, architect-engineer, nuclear steam supply systems, and national laboratory organizations and resulted in validation scenario data for 39 specific nuclear software systems. Validation scenarios are defined as realistic dynamic tests of a software (or software and hardware) system which covers only the intended range of application of the software and are designed to sample important subsets of function, usually familiar and important operating situations, to provide assurance that the system performs the tested functions with the required accuracy and performance. Potential problems outside of the intended range of applications should be determined as part of the basic V&V process.

The review resulted in the categorization of validation scenarios into six types: PLANT, TEST, CODE, LICENSING, REGRESSION, and BASICS. The review showed that there are no standards, regulations, or guidelines for the selection of number and type of validation scenarios in the nuclear industry. Some rules of thumb for the selection and number of validation scenarios were developed which relate to class of V&V, availability and pedigree of nuclear power plant or experimental test facility data, and range of application. Characteristics and good practices were established so that the features of a system can be examined and a ballpark estimate of the number of scenarios can be generated. These rules of thumb are felt to apply both to AI systems and conventional software since they test the functional capability of the system without regard for its structure or method of development. Finally, a novel methodology for generating scenarios directly from the knowledge base component was suggested.

Conclusions:

Several major accomplishments and conclusions have resulted from this project:

- A set of recommended V&V guidelines to be used to assure the quality of expert systems and other AI systems, as well as conventional systems, were developed for the nuclear power industry.
- These AI systems include knowledge-based systems, frame-based systems and object-oriented systems. Since neural networks are implemented in either conventional or object-oriented languages, the guidelines apply to these systems as well.
- These guidelines have applicability to other systems as well, such as rule-based, frame-based, model-based, object-oriented, and conventional software systems.
- A methodology of characterizing IV&V techniques based on power and ease-of-use features and facilitating the selection of different methods (based on differing needs) was presented.
- Fault-specific verification, a methodology which identifies the specific faults which need to be avoided for a system and designs a set of test procedures to detect such faults, was introduced.
- A behavioral methodology for empirically determining the usability and effectiveness of various V&V methods was developed.
- A general methodology for generating validation scenarios of nuclear utility systems was developed.

1 INTRODUCTION

This report presents the results of the Expert System Verification and Validation (V&V) project which is jointly funded by the United States Nuclear Regulatory Commission (USNRC) and the Electric Power Research Institute (EPRI). The ultimate objective of this project is the formulation and documentation of guidelines for V&V of Artificial Intelligence (AI) systems in the nuclear power industry. The results of this project can also be used to benefit V&V of conventional software systems.

1.1 Background

The motivating concern behind this project involves the realization that artificial intelligence systems are more and more being introduced into use for nuclear power plants. EPRI itself has been a major developer of AI systems for nuclear applications. The movement toward digital instrumentation and controls will also likely increase the use of AI systems. There has been little effort made toward developing means to assure their reliability, particularly when integrated into existing conventional plant software. For software, the reliability assurance responsibility is known as software V&V or Independent (IV&V), to emphasize the fact that it is accomplished by an organization outside of the development group. Thus, the goal of this project is the development of adequate V&V procedures for AI systems, primarily for use in the nuclear utility industry. The phrase "expert systems" has been used frequently throughout this project. However, results apply to other AI systems and to systems used in other industries.

The absence of AI system V&V guidelines and recommended practices meant that neither could the utilities have guidance on how to evaluate the integrity of systems they developed nor could the USNRC have a technical basis for the safety evaluation of expert systems. Clearly, some initial guidance for both was needed. There are three levels of direction which could be provided, arranged in decreasing order of formality and stringency: Standards, Recommended Practices, and Guidelines. While Standards and Recommended Practices are judged to be inappropriate for AI systems at this stage, the development of Guidelines, the third and lowest level, is a reasonable objective. Accordingly, the two sponsors agreed to jointly fund a project to develop guidelines for the V&V of AI systems and ensured, through involved technical management, that the interests of both parties were well reflected not only in the initial project goals but also in the details of the performance of the several project activities. The final product, a set of V&V guidelines and detailed procedures, is intended not as a stopping point but rather as a firm beginning on the road to developing and formalizing a consensus concerning procedures for assuring that AI systems introduced into nuclear utility environments will be safe and reliable. Feedback from industry and academia will play an important role in enhancing and improving these guidelines. These guidelines should be considered as guidance based on technical judgment, not as requirements.

1.2 Technical Approach

The technical approach for this project included several facets. All possible sources of Verification and Validation methods were examined: those for AI systems, those for conventional software systems, and innovative V&V methods. Once the methods were identified or innovative methods were developed, they were "instantiated" on real expert systems. Four primary components of expert systems were identified as the inference engine, knowledge base, external interfaces, and tools and utilities. Each of the components were evaluated in terms of the degree to which conventional techniques were applicable for V&V of that component. In general, expert systems were found to be the same as conventional programming language systems, and the conventional V&V methods can generally be applied to expert systems. The knowledge base was the one component which required new techniques or extension of existing conventional techniques (see section 7.4.1 of Volume 2).

A key aspect of the technical approach was the emphasis on quantification. An attempt was made to quantify evaluations of methods since this is currently a highly subjective area. For example, an experiment using behavioral methods was conducted to determine the effectiveness of various V&V methods in detecting errors in knowledge bases. A second aspect concerns consensus. To ensure that the research being performed was on target and would be of use to parties performing V&V on expert systems, feedback was actively sought (and obtained) from users, sponsors, and industry (both software and nuclear industries).

Another key facet of the technical approach for this project was the emphasis on practicality and cost-effectiveness. Whenever appropriate, emphasis was placed on existing experience. For example, new approaches suggested by universities were de-emphasized in favor of "tried and true" approaches. Since the V&V methodology developed in this project will have to coexist within the nexus of existing standards, over 100 such standards were reviewed as part of this work.

The technical approach focused on specific software faults and methods that could detect those faults as opposed to undirected general testing. An attempt was also made to identify automated support tools for V&V to ease implementation of the methods. In general, an emphasis was placed on increasing the usability of the V&V methodology and guidelines, with the realization that unusable guidelines would realize that fate.

A further key aspect of the technical approach was assuring that it catered to the differing needs of different systems. The very wide range of systems to be subjected to V&V was taken into consideration and led to the identification of three classes of V&V based on system complexity and degree of required integrity. Throughout this report, the term "system" is used (e.g., system complexity, AI system) to refer to a software system. Finally, the team composition for this project was also an important facet of the technical approach. Individuals with backgrounds in AI systems, V&V, and the nuclear industry were brought together to ensure project success.

1.3 Report Organization

The AI systems V&V project consisted of ten separate but interrelated activities. The activities were re-organized so that cornerstone activities would be performed first, and other activities could build upon that work. Figure ES-1, in the executive summary, depicts the activity flow for this project. For example, keystone Activities 1 and 2, which identified the conventional and expert system V&V methods currently in use, had to be completed in order for later activities to commence.

The final report describing the activities and results of the project is organized into several volumes. This first volume represents the cumulative technical report, which summarizes all the activities completed for the project. The applicability and limitations of conventional V&V methods (Activity 3), the guidelines for the V&V of AI software systems (Activity 4), and the test verification methodology (Activity 5) are discussed in Volume 1. In addition, Volume 1 covers some limited work in standards and object-oriented programming systems. The second volume presents the survey of conventional V&V methods (Activity 1), and the third volume describes the survey of expert system V&V methods currently in use (Activity 2). Volume 4 presents the results of the knowledge-base certification experiment performed as part of Activity 6, volume 5 describes the V&V guidelines developed as part of Activity 7, and Volume 6 presents an examination of validation scenarios from Activity 8. all of these volumes are described below in Sections 2-6. Volume 7 presents the User's Manual for the V&V guidelines. This manual assists a software development project

manager of V&V manager in performing V&V on expert systems. the manual includes 16 guideline packages (set of recommended V&V methods) for a comprehensive coverage of development situations plus 11 detailed step-by-step explanations of selected V&V techniques. The guidelines are subjective and extensive industry use will be required before they can become an accepted practice. Finally, Volume 8 presents the extensive bibliography for the entire project.

2 SURVEY OF V&V METHODS

The overall objective of the survey is to determine how much of the extensive context and accomplishment of conventional V&V activities can be employed directly for expert systems. Expert systems V&V lacks the history of conventional software V&V, though it is further along than V&V of other non-conventional software technologies such as neural network systems, object-oriented systems, and specialized software for parallel processors. First, the amount of V&V that is desirable for a system will be addressed, in terms of degree of required integrity and system complexity. Next, a taxonomy of conventional V&V methods will be presented. The methods will then be characterized according to the faults they detect, and will be ranked by their cost-benefit. Finally, the applicability of these methods to expert systems will be assessed.

In the context of this report, the term "system" refers primarily to the software system. Where a complete system must include hardware devices for sensing and control, the focus is always on the software. Thus, the term "system" should not be interpreted to mean physical system.

2.1 Definition of Systems in Terms of V&V Classes

All conclusions and findings on definition of systems in terms of V&V classes are based on Section 2.4.3 of Volume 2. Two major factors are proposed as determinants of the amount of V&V that is desirable for a software system: complexity of the system and degree of required integrity. As complexity increases, potential system faults can increase. The six contributors to system complexity are described in Table 2.1-1. The degree of required integrity determines how strongly the customer/user is interested in avoiding having such faults occur. This is an individual decision to be based on a wide variety of factors. Of primary importance, though, are negative consequences that can occur with system faults, such as:

- loss of human lives
- human injuries
- long-term human health problems
- injuries or fatalities to plant or animal life
- destruction or pollution of the environment
- destruction of plant system elements
- discomfort to people or animals
- interruption of service and/or disruption of system mission
- inconvenience to people
- direct financial loss
- resource cost
- loss of business
- impact on an organization's capability to perform
- loss of information
- loss of opportunity
- loss of privilege, authority, or rights
- loss of good will or reputation
- impact on the availability or operation of other systems

Table 2.1-1 Six factors of software system complexity¹

COMPLEXITY FACTOR	COMPLEXITY LEVEL		
	LOW	MEDIUM	HIGH
Physical Control Capability	<ul style="list-style-type: none"> • None • Advisory function only 	<ul style="list-style-type: none"> • No direct, but can provide decision data into control modules 	<ul style="list-style-type: none"> • Can directly manipulate and control system elements
Processing	<ul style="list-style-type: none"> • Not real-time • Sequential • Single Processor • Synchronous • Centralized • Batch/Interactive 	<ul style="list-style-type: none"> • Near or full real-time • Multiple Processors • Central/ Distributed • Interactive 	<ul style="list-style-type: none"> • Real-time • Concurrent/Multiple, Heterogeneous Processors • Highly Distributed • Cooperating • Asynchronous • Interactive
Interactivity with Other Systems	<ul style="list-style-type: none"> • Stand-alone • Single user-interface • No data-interfaces • User-driven • No interrupt handling 	<ul style="list-style-type: none"> • Embedded/Attached • Continuous/Intermittent Data-Input • Usually Data-driven • Possible Interrupt-handling 	<ul style="list-style-type: none"> • Embedded • Continuous Data-Input, Multiple channels • Usually Data Driven • Possible Interrupt-handling
Knowledge/Data Structures and Storage	<ul style="list-style-type: none"> • Homogeneous • Centralized • Derived from codified sources 	<ul style="list-style-type: none"> • Homogeneous/Heterogenous • Centralized/Distributed • Derived from codified sources and experts 	<ul style="list-style-type: none"> • Heterogeneous • Centralized/Distributed • Derived from codified sources, experts, or invented
Decision Procedure	<ul style="list-style-type: none"> • Backward (top-down) or forward (bottom-up) Chaining • Breadth first or depth first • Monotonic Reasoning 	<ul style="list-style-type: none"> • Backward, forward, and mixed chaining • Breadth first or depth first • Monotonic or non-monotonic reasoning • Heuristic Reasoning • Constraint-based reasoning • Belief-revision, truth maintenance 	<ul style="list-style-type: none"> • All types of chaining • Breadth first or depth first • Monotonic or non-monotonic reasoning • Model-based inferencing, plus all other types
Uncertainty Handling	<ul style="list-style-type: none"> • None 	<ul style="list-style-type: none"> • Fuzzy Reasoning • Reasoning under uncertainty 	<ul style="list-style-type: none"> • Complex Fuzzy and uncertainty reasoning • Multiple-Hypothesis evaluation (e.g., Bayesian)

¹This table is replicated from Table 6.3-1 in Volume 5.

High values of both of the factors of complexity and required integrity would produce a high likelihood of faults at a high level of desired avoidance, and a very substantial amount of V&V would be warranted to achieve this integrity goal. Similarly, lesser amounts of V&V would be indicated for lower levels of these factors.

Figure 2.1-1 depicts the determination of the level of required V&V as a function of these two factors. The three levels of complexity were chosen to roughly cover equal intervals of the expected number of contained system faults. The three levels are low complexity, moderately high complexity, and quite high complexity. Note that direct control functions exist only in the quite high case. Since required integrity is a subjective factor, equally-spaced levels of Low, Medium, and High were chosen to cover the range. The intersection of the highest complexity and integrity levels (cell 3) defines the situation requiring the most aggressive V&V approach and is called Class 1. Class 3 includes cells 7 and 8 and is the lowest level of required V&V. The remaining six cells (1,2, 4, 5, 6, and 9) define the intermediate V&V class, Class 2.

2.2 Classification of Conventional and Expert System V&V Methods

All conclusions and findings on classification of conventional and expert system V&V methods are based on the results detailed in Volume 2. After gathering and reviewing over 300 technical sources, 153 conventional V&V techniques were discovered. These techniques have been clustered into three major categories: one for the requirements/design phase of a Life-cycle (28 methods), and two for the implementation phase of a Life-cycle: static testing (58 methods) and dynamic testing (67 methods). Static testing involves analysis and inspection of the system's source code without actually executing the code. Dynamic testing involves the actual execution of the system's code. Figure 2.2-1 depicts the clustering of the 153 methods. A method was included in the survey as a distinct type if it was commonly distinguished or discussed in depth by individual authors. Methods are not evaluated at this stage. This is accomplished by means of defining metrics for all methods, as discussed in section 2.3.2.

2.2.1 Requirements/Design Techniques

The 28 techniques for the requirements/ensign phase are broken into four major classes and their subclasses: formal methods, semi-formal methods, reviews and analyses, and traceability assessments. **Formal methods** involve mathematical and logical calculations for expressing relationships among data and other objects and for expressing the processes which interact with them. **Semi-formal methods** are not as difficult to apply as formal methods and often use sequencing and selection of operators/objects to achieve their goal. **Reviews and Analyses** include well honed procedures which allow various parties interested in the final system to hear presentations on the work in progress and express their concerns. **Traceability Assessments** establishes the relations between the requirements specification and the design, matching elements of one to the other. This analysis identifies unmapped requirements and unmotivated (or superfluous) design elements. The descriptions of the 28 individual requirements/design techniques are provided in Table 2.2.1-1.

Figure 2.1-1 Three Classes of Stringency of Recommended V&V with Examples of Expert Systems for the Nuclear Power Industry¹

System Complexity	Degree of Required System Integrity		
	Low	Medium	High
INTEGRITY DIMENSION			
Quite High Embedded, Real-time, Continuous Data- Input channels, Direct Control Functions, May have Interrupt Processing	1 Steam Generator Blowdown Control system Radioactive waste management	2 Automatic control-rod manipulation Main Feedwater Control System	3 Reactor Protection System V&V CLASS 1
Moderately High Embedded or Attached No Direct Control functions, Control- Decision Support function, At least near real-time Continuous Data Input Channels	4 TPA (Thermal Plant Analyzer) Turbine Generator Diagnostic Monitoring	5 EOPTS (Emergency Operating Procedure Tracking System) RSAS (Reactor Safety Assessment System) REALM (Reactor Emergency Action Level Monitor)	6 ECCS (Emergency Core Cooling System) Real-time Monitoring and Diagnosis V&V CLASS 2
LOW Stand-Alone, User Driven, Non Real-time, Advisory Functions, No continuous Data Input	7 Fuel-Rod Reshuffling Planner Water Chemistry Advisor	8 SARA (Safety Review Advisor) Plant-Layout V&V CLASS 3	9 In-service ECCS (Emergency Core Cooling System) inspection Advisor ESAS (Emergency Safety Actuation System) Testing System

¹ Candidate systems for V&V are to be matched to the closest cell examples to determine suggested V&V class.

* Number in parentheses indicate the number of individual number of V&V Methods of that type.

Figure 2.2-1. Classes of Conventional V&V Methods Organized by Life-Cycle Phase

Table 2.2.1-1 Description of the conventional requirements/design V&V methods

V&V Classes/Subclasses	Description
1.1 FORMAL METHODS	
1.1.1 General Requirements Language Analysis/ Processing (Davis, 1990)	Expression of requirements specifications in a special requirements language and analysis of execution of that expression to assess the adequacy of the requirements.
1.1.2 Mathematical Verification of Requirements (Jones, 1986)	Translation of requirements into mathematical form for proving various properties (security, ultra-high reliability).
1.1.3 EHDM (Rushby, 1991)	A specification (and verification) language based on a strongly typed higher-order logic, incorporating elements of the Hoare relational calculus, with complete formal semantic characterization.
1.1.4 Z (Chisholm, 1990)	A typed set-theoretic language employing mathematical expressions, schema, to describe aspects of a system; the schema consist of declarations grouped with property predicates about the declarations.
1.1.5 Vienna Definition Method (Jones, 1986)	A discrete-mathematical formalism for rigorously defining the semantics specification processes.
1.1.6 Refine Specification Language (Ng, 1990)	A knowledge-based commercial specification language and environment based on transformational programming concepts.
1.1.7 Higher Order Logic (HOL) (Gordon, 1985)	An implemented logic notation that allows specifications to be written in terms of hierarchically structured collection of logical theories which contain axiomatic properties of the operations that are introduced.
1.1.8 Concurrent System Calculus	Provides a calculus for the description and specification of concurrent systems. Similar to Milner's (1986) calculus, and basis for LOTOS (ISO, 1987) Language for Temporal Ordering Specification.
1.2 SEMI-FORMAL METHODS	
1.2.1 Ward-Mellor Method (Ward, 1986)	An extension of Structured Analysis system specification techniques (e.g., Ross, 1977; DeMarco, 1978) developed at Yourdon, Inc. for real-time systems, emphasizing data flow diagrams with control-flow annotations.

Table 2.2.1-1 (Continued).

V&V Classes/Subclasses	Description
1.2.2 Hatley-Pirbhai Method (Hatley & Pirbhai, 1987)	Like Ward-Mellor, except that the techniques were developed at the Boeing and Lear companies; emphasizes control flow diagrams; considered to have superior architectural modeling capability.
1.2.3 Harel Method (Harel, 1987)	Like Ward-Mellor, but using unique Statechart notations to accomplish similar modeling as the above, but generally considered richer and more elegant. Implemented in a set of tools called STATEMATE (cf. Harel et al., 1990).
1.2.4 Extended Systems Modeling Language (ESML; Bruyn et. al., 1988)	A modeling language with elements of the Ward-Mellor and Hatley-Pirbhai methods, currently under development.
1.2.5 Systems Engineering Methodology (SEM; Wallace, 1987)	A method combining structured analysis techniques and software cost reduction methods (Heninger, 1980), similar to ESML.
1.2.6 System Requirements Engineering Methodology (SREM; Alford, 1977)	A hardware/software specification language for describing both data- and control-flow of systems, used extensively in US DoD weapons systems development (also known as the DCDS method; later implemented in a commercial system called TAGS). Now fully implemented, with complete system support, as the RDD-100 tool (supports design animation; Ascent, 1990).
1.2.7 FAM (Chisholm, 1990)	Representation of systems in terms of graphical annotated flow-nets , based on extensions to Petri-net theory, permitting symbolic execution with an automated theorem prover.
1.2.8 Critical Timing/Flow Analysis (Wallace, 1989)	Modeling and (usually) simulation of process and control timing aspects of the design to determine if such requirements are satisfied (e.g., with Petri Nets).
1.2.9 Simulation-Language Analysis (Hartway, 1990)	Representation of a system design in a general purpose simulation language (e.g., SLAM II), and analysis of the execution results.
1.2.10 Petri-Net Safety Analysis (Leveson & Stolzy, 1987)	Systems modeling with untimed (and timed) Petri nets to assure design adequacy for catastrophic-failure and other safety problems.
1.2.11 PSL/PSA (Teichroew, 1977)	Constrained natural language-like representation of requirements and specifications with automated support.

Table 2.2.1-1 (Continued).

V&V Classes/Subclasses	Description
1.3 REVIEWS AND ANALYSES	
1.3.1 Formal Requirements Review (NBS500-93, 1982)	Review by special personnel of the adequacy of the requirements specification according to detailed pre-established set of criteria and procedures.
1.3.2 Formal Design Review (NBS500-93, 1982)	Review by special personnel of the adequacy of the design according to detailed pre-established set of criteria and procedures.
1.3.3 System Engineering Analysis (DSMC, 1990)	A variety of activities associated with developing a complete operational system which satisfies certain requirements; these activities include creation of a functional architecture, allocating function to hardware and software, accomplishing trade-off and make/buy studies, and development of a work-breakdown structure.
1.3.4 Requirements Analysis (Davis, 1993)	Analysis of requirements to ensure completeness, consistency, clarity, explicitness, etc.
1.3.5 Prototyping (Schulmeyer, 1992)	Building a model of a design to evaluate one's approach or to better define the requirements; prototypes may range from mock-ups to initial versions which are retained and built on.
1.3.6 Database Design Analysis (Nijssen, 1989)	Checking the design of the structure, normal form, declarations, and values of a database.
1.3.7 Operational Concept Design Review (Rasmussen, 1987)	Review of the design of the concept of operations for the system, especially the interaction with human operators.
1.4 TRACEABILITY ASSESSMENTS	
1.4.1 Requirements Tracing Analysis (NBS500-93, 1982)	Identification of individual requirement aspects and tracing of these to design aspects, and from the design to aspects of the implemented program.
1.4.2 Design Compliance Analysis (Wallace, 1989)	Verification process that design is compliant with--realizes--all aspects of requirements.

2.2.2 Static Testing Methods

Static testing involves analysis and inspection of the system's source code without actually executing the code. Static testing is broken into five major classes: algorithm analysis, control and performance analysis, data analysis, fault/fault analysis, and inspection. **Algorithm analysis** methods involve analysis of the algorithms of a program and may involve the translation of the algorithms into some kind of language or structured format. **Control and Performance Analysis** technique: detect and characterize program control-flow with the sequential and hierarchical aspects, detect timing aspects, and/or focus on the concept of operations.

Data analysis methods deal with the relationship of named variables in one module to those in other module, an analysis of the database or data model, an analysis of the flow of data from input to output or one process to another, sequential and concurrent processes, and/or qualitative relationships among system states as determined by data flow. **Fault/Fault Analysis** methods use a general fault-analysis strategy (carried over from hardware testing) to examine programs for faults. High-level functional and operational descriptions are used to identify how the system might logically fail. The program is then examined to determine if any of those fault-mode possibilities could logically occur and in what context and under what conditions.

Inspections involve groups which serve as a critical review audience for a discussion of program code. Specific fault-exposure techniques (Clean-room techniques) also fall into this class. The descriptions of the 58 Static testing techniques are provided in Table 2.2.2-1.

2.2.3 Dynamic Testing Methods

Dynamic testing involves the actual execution of the system's code. Dynamic testing has been divided into 11 major classes: general testing, special input testing, functional testing, realistic testing, stress testing, performance testing, execution testing, competency testing, active interface testing, structural testing, and error-introduction testing. **General testing** is a catch-all class, including unit/module testing, system testing, compilation testing, post-compiler checkers, reliability testing, regression testing, metric-based testing, and ad hoc testing.

Special input testing is broken into two classes: random testing and domain testing. Random testing involves a variety of means for randomly selecting input test cases. Domain testing involves performing a detailed analysis of the input space to design a set of test cases which sample its features.

Functional testing assesses whether the system exhibits specific functionality as described in the requirements. Sometimes called "black box testing", this method focuses on the input/output functionality of the system without taking into account the internal structure of the program. **Realistic testing** assesses the system under realistic conditions, often in the field, frequently with realistic scenarios, and sometimes with special data-stream simulators. **Stress testing** places the system under heavy load conditions, involves unusual inputs to the system under degraded conditions, tests the boundaries of variables, and/or systematically violates design parameters.

Performance testing focuses on assessing system performance measures such as CPU usage, memory usage, timing characteristics, delays, and queues and paging. **Execution testing** assesses aspects of the program's execution such as following the path of execution of a specific functional capability within and across program modules. **Competency testing** uses test cases to compare software system output to an external standard.

Table 2.2.2-1 Description of the conventional static testing V&V methods

Static Testing V&V Methods	Description
2.1 Algorithm Analysis	
2.1.1 Analytic Modeling (Jones, 1986)	Representing the program logic and processing in some kind of model and analyzing it for sufficiency.
2.1.2 Cause-effect Analysis (Davis, 1990)	Identifying the triggers of processes, their effect during activation in states of variables, and the final terminating conditions.
2.1.3 Symbolic Execution (King, 1976)	Representing the data computations as algebraic equations and solving these algebraically through the whole program.
2.1.4 Decision Tables (Omar, 1991)	Tables which represent different logical combinations of events of conditions that might occur. Used as a static method to identify functionality and to provide the basis for selection of dynamic testing test-class.
2.1.5 Trace-assertion Method (Parnas, 1988)	An algorithm specification (or representation) method involving description of the sequence of invocations of system modules, including I/O values, in terms of axiomatic assertions about the traces, in a "black-box" fashion. Used in conjunctions with "A-7 Table Format" representation (similar to decision tables).
2.1.6 Functional Abstraction (Mills, 1987)	Representing design or program as a series of mathematical functional based on a small set of primitive programming functions (e.g., iteration, sequence, select, etc.) then, recursively, dividing parent functional specifications into sub-specifications and mathematically verify equivalence.
2.1.7 L-D Relation Methods (Parnas, 1988)	An alternative to functional specifications for non-deterministic programs using relation and the competence set of states in which termination is guaranteed.
2.1.8 Program Proving (Mills, 1987)	For each code segment, developing formal specifications of functional intent and specific I/O characteristic; for actual or symbolic input then proving via some proof procedure that the segment performed as intended.
2.1.9 Metric Analyses (Jensen, 1985)	Computation of various complexity metrics for the program.
2.1.10 Algebraic Specification (United Kingdom 0055, 1989)	Specification of program procedures in terms of algebraic expressions.

Table 2.2.2-1 (Continued)

2.1.11	Induction-Assertion Method (Hoare, 1985)	Use of abstract data types to present a specification so that proofs of correctness can be implemented using these.
2.1.12	Confidence Weights Sensitivity Analysis (O'Leary, 1990)	Using statistical analyses to measure the sensitivity, accuracy or bias in the confidence factor/weights placed on rules' conclusions.
2.1.13	Model Evaluation (Bellman, 1990)	Evaluation of models in the system by modeling experts and by subject matter experts.
2.2 Control and Performance Analyses		
2.21	Control Flow Analysis (Ward, 1985)	Analyzing the program into a series of decision and process actions and representing all of the possible alternative process sequences in the program. Often used to assess whether program is well-structured and has no unreachable code.
2.22	State Transition Diagram Analysis (NBS 500-93, 1982)	Determining the condition (variable states, etc.) that trigger the onset and cessation of program processes.
2.23	Program Control Analysis (NUREG/CR-4640, 1987)	Related to 2.2.1 and 2.2.2 but concerned more with the sequential aspects leading to a particular execution path in a program.
2.24	Operational Concept Analysis (Rasmussen, 1987)	Analysis of the manner in which the software system interacts with and is dependent upon states of the environment and external decisions, especially of human operators.
2.25	Calling Structure Analysis (NBS 500-93, 1982)	Module by module analysis of hierarchically structured programs involving procedure calls to determine what sequence of high level module calls led to the invocation of that particular module and what modules in turn are called by it.
2.26	Processing Trigger/Timing Analysis (Hatley, 1987)	Analysis of the conditions which activate a process (similar to 2.2.2) with special concern for the timing of activation relative to other processes.
2.2.7	Worst-case Timing Analysis (Wallace, 1989)	Analysis to determine the longest execution-time path through a program often comparing this to a reference safety limit.
2.2.8	Concurrent Process Analysis (Rattray, 1990)	Analysis of the overlap or Concurrency of different processes in multi-tasking, parallel processing, concurrent processing programs.

Table 2.2.2-1 (Continued)

2.3.1 Data Analysis	
2.3.1 Data Flow Analysis (Deutsch., 1982)	Analysis of the data inputs, outputs, and controls to all program processes.
2.3.2 Signed Directed Graphs (Sudduth, 1991)	Graphical qualitative representative of the directional effects of system states on other states of certain other system components (also called influence Diagrams).
2.3.3 Dependency Analysis (Dunn, 1984)	Determining what variables depend on what other variables, similar to "influence diagrams".
2.3.4 Qualitative Causal Models (Oyeleye, 1990)	Development of models of the process and event causes of qualitative stated and changes of a system.
2.3.5 Look-up table Generator (NBS 500-93, 1982)	Generating the location within various modules of data and control variables.
2.3.6 Data Dictionary generator (Ng, 1990)	Generating a defined/used table of locations of all program variables.
2.3.7 Cross-reference List Generator (NBS 500-93, 1982)	Generating the location of all data variables, in form of cross-reference table of modules.
2.3.8 Aliasing Analysis (NBS 500-93, 1982)	Analysis of the aliases of variables used in the main procedure and passed as parameters/arguments to its called procedures (and theirs).
2.3.9 Concurrency Analysis (Rattray, 1990)	Analyzing programs for existing or potential concurrent data-paths and processing.
2.3.10 Database Analysis (Nijssen, 1989)	Checking the implementation of structure, normal forms, declarations, and values of a database.
2.3.11 Database Interface Analyzer (NBS 500-93, 1982)	Checking the interfaces(s) of a program with primary data input for error-detection and handling, consistency-checking, etc.
2.3.12 Data-Model Evaluation (Davis, 1990)	Evaluating the adequacy and features of the data schema or meta-schema used to organize the DB and data structures.
2.4 Fault Failure Analysis	
2.4.1 Failure Mode, Effects, Causality Analysis (FMECA, MIL-STD-1629A, 1984)	Identifications of the failure modes of each system component and analyzing the consequences of each failure type. Information gathered includes failure description, cause(s), defect(s), detection means, resultant safety consequences or other hazards, and recovery methods and conditions.

2.4.2	Criticality Analysis (Wallace, 1989)	Identification of the critical points of failure in a program and the development of test cases to verify their accuracy and robustness.
2.4.3	Hazards/safety Analysis (Rushby, 1988)	Analysis of failure data to develop metrics and casual hypotheses about system components' failure rates, fault-sources, and future behavior.
2.4.4	Anomaly Testing (Ng, 1990)	Checking the program for irregularities of style, syntax, or practice, or for signs of potential defects.
2.4.5	Fault-tree Analysis (Event-tree Analysis, Leveson, 1983)	Beginning with a system hazard or failure, the analysis identifies or hypothesizes immediate and proximal causes, and describes the combination of environment and events that preceded, usually in the form of a directed graph or "and-or tree". Often accompanied by an "event tree analysis" showing relevant event-propagation information.
2.4.6	Failure Modeling (Davis, 1990)	Analysis of the system (and requirements) to identify potential hazards or safety events, or to consider such events determined by dynamic testing. Full analysis involves 2.4.2 in addition.
2.4.7	Common-cause Failure Analysis (United Kingdom, 1989)	Identifying failures that affect apparently independent modules.
2.4.8	KB Syntax Checking (Preece, 1991)	Examination of a knowledge base for syntactic errors or anomalies in the composition of rules, frames, and other knowledge elements; no additional external information in required (see below).
2.4.9	KB Semantic Checking (also "Knowledge-Checking", Stachowitz, 1987)	Use of external meta-rules, constraints, or engineering knowledge to check the internal semantic consistency of a knowledge base.
2.4.10	Knowledge Acquisition/Refinement Aid (Desimone, 1990)	Use of an automated tool during knowledge acquisition refinement (maintenance) to prevent some types of errors from refinement created in the first place or to assure complete coverage of possible input values, possibly through machine leaning techniques.
2.4.11	Knowledge Engineering Analysis (Hart, 1986)	Similar to Knowledge Acquisition/Refinement Aid, but a manual process.
2.5 Inspections		
2.5.1	Informed Panel Inspection (Culbert, 1987)	Convocating a qualified group to review the equality of a program.

Table 2.2.2-1 (Continued)

2.5.2	Structured Walkthroughs (Fagan, 1986)	An analysis of a program module, usually by programmer, with audience of programming team members.
2.5.3	Formal Customer Review (NUREG/CR-4640, 1987))	A formal evaluation by the customer representatives of the adequacy of a program.
2.5.4	Clean-room Techniques (Mills, 1987)	A number of analytic (and procedural) practices for extremely high reliability code production.
2.5.5	Peer Code-checking (Mills, 1987)	Fellow programmers checking each other's production code.
2.5.6	Desk Checking (Dunn, 1984)	Inspecting program source code without benefit of automated tools.
2.5.7	Data Interface Inspection (Ng, 1990)	Inspection of all data interfaces of a program for adherence to specification, error-handling, consistency-checking, and other features.
2.5.8	User Interface Inspection (NUREG/CR-4227, 1985)	Inspection of all aspects of the interface to the user and operator for adequacy and other criteria.
2.5.9	Standards Audit (Dunn, 1984)	Evaluation of the program to determine its compliance with the set of governing standards and guidelines.
2.5.10	Requirements Tracing (NBS 500-93, 1982)	Tracing forward from each unique requirement element to specify code modules which are intended to implement that requirement. Requirements which cannot be so mapped are flagged as "unfilled requirements." Program elements which are not traceable back to any requirement are flagged as "unintended function."
2.5.11	Software Practices Review (Humphrey, 1990)	A review of a software organization to advise its management and professionals on how they can improve their operation.
2.5.12	Process Oriented Audits (Humphrey, 1990)	An examination of products of the software development effort(such as unit development folders) with the emphasis on improving the software development process.
2.5.13	Standards compliance (Bryan, 1988)	Comparison of system to imposed standards (e.g., documentation content, coding).
2.5.14	System Engineering Review (DMSC 1990)	A variety of checks and analyses to determine that good system engineering principles have been followed in the implementation (see 1.3.3).

Active interface testing examines data interfaces, user interfaces, display characteristics, and the concept of operations of the system. **Structural testing**, sometimes called white box testing, uses knowledge about the actual program composition and structure to generate test cases. This encompasses statement testing, branch testing, path testing, test-coverage analysis testing, conditional testing, and data-flow testing. **Error-introduction testing** covers error seeding and fault insertion. These methods can be used to assess the sensitivity of the test case suite.

The 67 Dynamic testing techniques are described in Table 2.2.3-1.

2.3 Characterization/Ranking of V&V Methods

All conclusions and findings on characterization/ranking of V&V methods are based on Section 6.0 of Volume 2. V&V techniques are characterized, first, in terms of the types of faults they are judged to detect, and, second, in terms of eight operational features which are described in Volume 2. These eight features are then combined in various ways to produce a "cost/benefit" measure and an "effectiveness" measure. These measures finally permit overall ranking of methods within the three classes of required V&V. There is little empirical data dealing with the effectiveness of specific techniques, therefore these ratings on each V&V technique are subjective. However, it should be noted that anyone who believes the ratings should be different can change them appropriately and still use this approach.

2.3.1 Characterization in Terms of Faults Detected

The taxonomy presented in Table 2.3.1-1 has face validity (in that it appears reasonable on the face of things) and construct validity (in that it is based upon reasonable underlying constructs), but it lacks empirical validity. This is due to the fact that there is little data on the effectiveness of fault detecting techniques. Nonetheless, it reflects a consensus of views on types of faults and is sufficiently detailed for this activity. Table 2.3.1-1 shows the taxonomy of 52 faults according to the three main Life-cycle phases: requirements, design, and coding. The coding types of faults were further divided into 3 sub-categories of logic and control, data operations and computations, and other. Faults were identified using two main criteria: 1) they did not overlap or subsume each other, and 2) they were not specific to a particular language or environment. ("Faults" are used synonymously with "defects" and "flaws.")

The 153 V&V methods were evaluated against this set of 52 types of faults. The evaluation was based on whether individual techniques could be expected to detect a specific type of bug. This was an informed but subjective assessment, and did not consider how well or how easily a fault could be found by a method. The results of these subjective judgments are shown in the second column of Table 2.3.1-2. The total number of detectable fault types by a particular method is shown in the same column in parentheses. The total number of detected fault types is considered to be an index of what is called **Broad Power** (discussed below). Table 2.3.1-3 shows how many techniques can detect each type of fault. Each of the techniques covered anywhere from 2 to 52 of these faults. Each fault was covered by anywhere from 21 to 50 V&V techniques.

2.3.2 Characterization in Terms of Eight Factors: Cost-Benefit and Effectiveness Measures

In addition to Broad Power, seven additional factors were introduced to characterize the V&V techniques. These are discussed in more detail in Section 6.0 of Volume 2. Four factors measure the **power** aspects of each technique.

- 1) **Broad Power** is a function of the number of different faults detectable by the technique.
- 2) **Hard Power** is a judgment about the capability of the technique to detect hard problems, problems which are not at all obvious on inspection and are difficult to find with dynamic techniques. These problems may also

be intermittent because they depend on non-obvious aspects of the running context, or they are enabled by various obscure means.

3) **Formalizability** assesses the extent to which a technique lends itself to formal calculus or algebraic representations of the specification, design, or implemented system. This would allow automated theorem-provers (if developed) to detect anomalies, contradictions, inconsistencies, etc.

4) **HCI Testability** characterizes whether the Human-Computer Interface is directly testable using the technique. Since the applications of strongest interest are decision-support type systems to help users process and interpret information as well as to advise them on alternative actions, the HCI is an important aspect to be tested. Techniques that test the HCI are seen as having higher power.

Four factors measure the ease-of-use aspects of each technique.

1) **Ease of Mastery** is the ease with which a technique can be taught, understood, and applied. It is measured in terms of the educational or professional level required to deal with the technique concepts and the amount of training time needed to teach the concepts.

2) **Ease of Setup** refers to the labor, time, and resources required to have the V&V technique ready to be applied to the program.

3) **Ease of Running/Interpretation** measures the ease (or difficulty) of actually applying the technique and interpreting the findings.

4) **Usage** is the extent to which the technique is generally and commonly used. The inference is that the higher the usage, the greater the general ease-of-use of the technique, or the easier it might be to get approval to use it.

Table 2.2.3-1 Description of the conventional dynamic testing V&V methods

Dynamic Testing Methods	Description
3.1 GENERAL TESTING	
3.1.1 Unit/Module Testing (Ng, 1990)	General testing of single program modules.
3.1.2 System Testing (Dunn, 1984)	Testing of the overall completed software system with test-cases representative of general program characteristics including its logic and computation, and its timing.
3.1.3 Compilation Testing (Davis, 1990)	Using compiler diagnostics and problem-reports to test system.
3.1.4 Reliability Testing (Boehm, 1981)	Selecting test-cases to exercise particular aspects of the system believed to be unreliable; also extensive testing to assess component failure rates.
3.1.5 Statistical Record-Keeping (Boehm, 1981)	Collecting data on errors discovered for particular system modules to suggest which modules should be tested more thoroughly or even redesigned; especially important in the maintenance phase of the life cycle.
3.1.6 Software Reliability Estimation (BSI, 1989)	Related to 3.1.4 but applying sophisticated statistical estimation techniques to fault and error data, to guide continued data collections and to predict system and sub-system failures.
3.1.7 Regression Testing (NBS500-93, 1982)	Repetition of a test suite after program modification to assess effects of changes.
3.1.8 Metric-Based Testing (Jones, 1986)	Selection of some aspect of the program for testing on the basis of the value of some metrics computed for it (usually "complexity").
3.1.9 Ad Hoc Testing (Dunn, 1984)	Test-cases selected arbitrarily by programmer without careful planning.
3.1.10 Beta Testing (Dunn, 1984)	Early release of the system to one or more volunteer user sites for final testing under realistic field conditions.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.2 SPECIAL INPUT TESTING	
3.2.1 <u>Random Testing</u> ¹ (Barnes, 1987)	Selecting test-cases according to some random statistical procedure.
3.2.1.1 Uniform whole program testing	Selecting test-cases such that each input variable is assigned any value inside its range with equal probability, over the whole program domain. (Best of the 4 techniques).
3.2.1.2 Uniform boundary testing	Selecting test-cases, with equal probability, around the boundaries of the ranges of input variables (within the range, at range limit, and outside range).
3.2.1.3 Gaussian whole program testing	Selecting test-cases for input variables according to Gaussian distribution (usually with a mean in the middle of the variable range, and with a standard variation of 1/12 its range).
3.2.1.4 Gaussian boundary testing	Selecting test-cases for input variables drawn from a Gaussian distribution across the boundaries of their valid ranges.
3.2.2 <u>Domain Testing</u> ¹ (Beizer, 1990)	Analysis of the boundaries and partitions of the input space and selection of interior, boundary, extreme, and external test-cases as a function of the orthogonality, closedness, symmetry, linearity, and convexity of the boundaries.
3.2.2.1 Equivalence Partitioning (Myers, 1979)	A type of domain testing which partitions the input domain into equivalence classes such that a test of a representative value from a class is assumed to be a test of all the class values.
3.2.2.2 Boundary-value Testing (Myers, 1979)	A type of domain testing which selects test values at and around (just inside, just outside) input boundaries.
3.2.2.3 Category-Partition Method (Ostrand, 1988)	Similar to equivalence partitioning, but the equivalence classes are more rigorously derived from a functional decomposition, and there is much more attention in selecting representative value to co-occurrence constraints among classes.
3.2.2.4 Revealing Subdomains Method (Weyuker, 1980)	A type of domain testing which takes into account both the partitioning of the overall input space by functional decomposition and also the internal program path structure.

¹ Both of the sub-categories of 3.2 (3.2.1 and 3.2.2) are included as distinct methods as well as their (non-exhaustive) sub-sub-categories.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.3 FUNCTIONAL TESTING	
3.3.1 Specific Functional Requirement Testing (Howden, 1980)	Selecting test-cases to assess the implementation of specific required functions.
3.3.2 Simulation Testing (Pritsker, 1986)	Generating special code to emulate various aspects of code to-be-implemented system.
3.3.3 Model-Based Testing (Davis, 1990)	Use of an analytic or process model of desired function to assess the implemented function.
3.3.4 Assertion Checking (NUREG/CR4640, 1987)	Bracketing code segments with assertions which can be compiled into executable code to verify assertions during code operation. Similar to the static technique of program proving but involves actual execution of code and assertions.
3.3.5 Heuristic Testing (Miller, L., 1990)	Emphasizes the importance of prior fault prioritization and analysis to determine fault-enabling conditions and appropriate test cases.
3.4 REALISTIC TESTING	
3.4.1 Field Testing (Rushby, 1988)	Testing of the program under actual installed conditions.
3.4.2 Scenario Testing (Ng, 1990)	Lab or Field testing with highly realistic cases or situations.
3.4.3 Qualification/Certification Testing (Jensen, 1979)	Extensive testing to meet some set of high standards of quality or performance.
3.4.4 Simulator-Based Testing (Ng, 1990)	Use of a simulator to generate realistic input data streams to the system to be tested.
3.4.5 Benchmarking (Mayrhauser, 1990)	Use of standard widely supported tests to exercise a number of aspects of system performance.
3.4.6 Human Factors Experimentation (CHI, 1988)	Evaluation of the human-user performance with the real or simulated system to determine adequacy of the human-computer interface.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.4.7 Validation Scenario Testing (ASME, 1990)	A realistic dynamic system test which samples important subsets of functional capability. Is usually the last type of testing done, and is intended to provide assurance to the end-user/customer (or regulator). More restrictive in extent and focus than 3.4.2.
3.4.8 Knowledge Base Scenario Generation (Vol. 6 of this report)	A proposed automated method for automatically generating validation scenarios from knowledge bases.
3.5 STRESS TESTING	
3.5.1 Stress/Accelerated Life Testing (NBS500-75, 1981)	Exercising the program as rapidly, with as much data input, CPU tasking, and memory load, as possible.
3.5.2 Stability Analysis (Dunn, 1984)	Choosing test-cases to exercise and stress the stability of the system.
3.5.3 Robustness Testing (Miller, L., 1990)	Testing the program with bizarre inputs under variously degraded conditions.
3.5.4 Limit/Range Testing (Ng, 1990)	Selecting test-cases to test (and exceed) the extreme ranges of allowable limits on variables/ parameters. (Also called Boundary Testing; strategy for automating test-case generation for rule basis given in Miller, 1990).
3.5.5 Parameter Violation (NBS 500-93, 1982)	Determining the various design parameters which led to the present implementation and systematically generating test-cases which violate these parameters.
3.6 PERFORMANCE TESTING	
3.6.1 Sizing/Memory Testing (Wallace, 1989)	Assessing the CPU and memory requirements of the program under various conditions.
3.6.2 Timing/Flow Testing (Dunn, 1984)	Assessing the rate of operation (and concurrency) of various program components and the rate of flow of information.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.6.3 Bottleneck Testing (Ng, 1990)	Determining the location of undesired delays and processing queries in the program's operation.
3.6.4 Queue size, register allocations, paging, etc. (Beizer, 1990)	Assessing any other performance aspect of the program.
3.7 EXECUTION TESTING	
3.7.1 Activity Tracing (Dunn, 1984)	Monitoring and evaluating the results of a particular program function or activity.
3.7.2 Incremental Execution (Ng, 1990)	Halting program execution at multiple points to assess performance variable values and data storage characteristics.
3.7.3 Results Monitoring (NBS 500-93, 1982)	Similar to 3.7.1 but more focused on a particular outcome regardless of the activity that generated it.
3.7.4 Thread Testing (Jensen, 1979)	Following control and data for a single function through multiple modules.
3.7.5 Using Generated Explanations (Miller, 1989)	Examining the explanations or rule traces produced by the expert system to evaluate if the reasoning process is correct.
3.8 COMPETENCY TESTING	
3.8.1 Gold Standard (Rushby, 1988)	Measuring program results against widely accepted standards.
3.8.2 Effectiveness Procedures (Llinas, 1987)	Assessing the sequential effectiveness of the program against some external standard.
3.8.3 Workplace Averages (Rushby, 1988)	Measuring program results against averages established in some workplace.
3.9 ACTIVE INTERFACE TESTING	
3.9.1 Data Interface Testing (Ng, 1990)	Testing the data interfaces to insure that all aspects of data I/O are correct, including buffering, change detection, checking, etc.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.9.2 User Interface Testing (NUREG/CR-4227, 1985)	Evaluation of the user interface from low level ergonomic aspects to instrumentation and controls human factors to global consideration of ease-of-use and appropriateness, taking into account CONOPS and information analyses (below).
3.9.3 Information System Analysis (NUREG/CR-4227, 1985)	Determining that operator-needed information is well organized and is available directly, and quickly, neither too much nor too little, neither inaccurate nor contradictory.
3.9.4 Operational Concept Testing (CONOPS Testing, Miller, L., 1990)	Testing that the concept of operations --the partitioning of decisions and initiative between the computer and the user-- is adequate, appropriate, and sufficiently flexible.
3.9.5 Organizational Impact Analysis/Testing (Booher, 1990)	Testing or analyzing the effect of the system on the user organization/corporate structure and/or methods after installation.
3.9.6 Transaction-flow testing (Beizer, 1990)	Identifying the flow of information between people and computers, for user-driven systems, as well as the internal computer processing transformation of that information, and developing a suite of tests to exercise each of the processing steps.
3.10 STRUCTURAL TESTING	
3.10.1 Statement Testing (Beizer, 1983)	Generating test-cases to exercise specific (or all) program statements in the source code.
3.10.2 Branch Testing (Miller, E., 1990)	Generating test-cases to exercise branches from conditional or case control structures.
3.10.3 Path Testing (Tung, 1990)	Augmenting branch testing to test various repetitions of program flow through interactive or loop structures of the program.
3.10.4 Call-Pair Testing (Miller, E., 1990)	Developing cases to test the argument and parameter interfaces among programs.
3.10.5 Linear Code Sequence and Jump (LCSAJ) (Miller, E., 1990)	Selecting of test-cases based on control-flow analysis, similar to branch testing, but often used in "lower" level languages such as assembler.

Table 2.2.3-1 (Continued).

Dynamic Testing Methods	Description
3.10.6 Test-Coverage Analysis Testing (Beizer, 1983)	Determining what statements, paths, branches, etc. are exercised by a set of test-cases.
3.10.7 Conditional Testing (Beizer, 1990)	Testing of statements involving Boolean or Relational (e.g., "A<B") tests. Selecting test-cases corresponding to values equal to, less than, and greater than the values in the conditions.
3.10.8 Data-flow Testing (Beizer, 1990)	Selection of test-cases to explore data anomalies discovered by examination of the program's control flow graph.
3.11 Error-Introduction Testing	
3.11.1 Error Seeding (Dunn, 1984)	Introducing errors of arbitrary kinds in a software system to assess the effects of such errors on system performance.
3.11.2 Fault Insertion (Rushby, 1988)	Introducing modifications to a program which will induce a failure or fault of a particular kind.
3.11.3 Mutation Testing (Ng, 1990)	Introducing errors of various kinds in a program and determining whether a given suite of test-cases detect the errors. Used to assess power of one's testing techniques for discovering problems.

Table 2.3.1-1 Types of software Faults

Type	Description	Occurs
1.0 Requirements	Originate in Requirements Phase; found in the Requirements Specification	
.1 Incomplete Decomposition	Failure to adequately decompose a more abstract specification.	System, Sub, Mod
.2 Omitted Requirement	Failure to specify one or more of the next lower levels of abstraction of a higher level specified.	System, Sub, Mod
.3 Improper Translation	Failure to carry detailed requirement through decomposition process, resulting in ambiguity in the specification.	System, Sub, Mod
.4 Operational Environment Incompatibility	Specification which does not accommodate the operational environment, such as data rates, data formats, etc.	System, Sub, Mod
.5 Incomplete Requirement Description	Failure to fully describe all requirements of a function.	Mod
.6 Infeasible Requirement	Requirement which is unfeasible or impossible to achieve given other system factors, e.g., process speed, memory available.	Mod
.7 Conflicting Requirement	Requirements which are pairwise incompatible.	System, Sub, Mod
.8 Incorrect Assignment of Resources	Over or Under stating the computing resources assigned to a specification.	Mod
.9 Conflicting Inter-system Specification	Requirements of cooperating systems, or parent/embedded systems, which taken pairwise are incompatible.	System
.10 Incorrect or missing external constants	Specification of an incorrect value or variable, or a missing value or variable in a requirement	Mod
.11 Incorrect or missing description of initial system state	Failure to specify the initial system state, when that state is not equal to 0.	Mod
.12 Overspecification of Requirements	Requirements or specification limits that are excessive for the operational need, causing additional system cost.	System, Sub
.13 Incorrect input or output descriptions	Failure to fully describe system input or output.	Mod

Table 2.3.1-1 (Continued).

Type	Description	Occurs
2.0 Design	Generated in design and appear in design documentation	
.1 Omitted requirement	Failure to address a requirement or specification in design.	System, Sub, Mod
.2 Misinterpreted requirement	Failure to accurately represent a requirement or specification in design.	System, Sub, Mod
.3 Data limitation	Failure to accommodate the full range of possible data.	Mod
.4 Unintended Design Element	Inclusion of design elements that cannot be traced to a requirement or specification.	Sub, Mod
.5 Hardware incompatibility	Non-existent or more capable hardware resources prescribed beyond those available, e.g. process cycles or memory.	Mod
.6 Software incompatibility	Assumes commercial package, utilities or operating system capabilities which are not available or function differently than assumed.	Mod
.7 Poor man-machine interface design	Man-machine interface is clumsy, hard to learn/use, hard to see/read, etc.	Sub, Mod
.8 Incorrect analyses of computational error	Design fails to adequately address factors of computation error, such as round-off, truncation, numerical approximation.	Mod
.9 Non-compliance	Design fails to conform to standards.	System, Sub, Mod
.10 Lack of adequate error traps	Failure to provide error traps that occur adequately frequently, are sufficient in scope, or supply a recovery mechanism.	Mod
.11 Failure to handle exceptions	Failure to handle unique conditions, or boundary conditions.	Mod
.12 Weak modularity	Design inadequately groups functions or requirements into modules.	Mod
.13 Rigid control structure	Control structure is designed with in-line logic or in other ways which preclude ease of expansion or modification.	Mod
.14 Missing or incorrect processing priorities	Control structure design does not allow processing priorities to be established or modified to satisfy requirements or implements them incorrectly.	System, Sub, Mod

Table 2.3.1-1 (Continued).

Type	Description	Occurs
.15 Breakdown between top-level & detail design	Design modifications at one level are not reflected at the other.	Sub, Mod
3.0 Code	Originate in code; exclusive of defects regularly detected by an assembler or compiler.	
3.1 Logic and Control		
.1 Unreachable Code	Code which fails to be accessed due to redundant, contradictory branching conditions.	Mod
.2 Improperly used flow control constructs	Improperly formed or used looping or branching constructs undetected by the compiler.	Mod
.3 Inverted predicates	A predicate (e.g. "if" statement) which is incomplete, transposed or incorrect.	Mod
.4 Improper process sequencing	Process steps are in wrong order (e.g., attempt to read a file before it is opened) and timing/synchronization errors in concurrent processing.	Sub, Mod
.5 Halting problem	A loop or recursive or non-deterministic machine without a meetable exit condition or halting condition.	Mod
.6 Instruction modification	Dynamic instruction modification.	Mod
.7 Failure to save or restore process communication	Failure to save the contents of registers to be used later or restore them upon exit, or to correctly handle interprocess communication mechanisms, e.g., semaphores, file/record locks.	Mod
.8 Unauthorized or incorrect recursion	Code developed to be recursive with a language which doesn't support recursion or improper use of recursion.	Mod
.9 Incorrect labels or control flags	A referenced but uncoded statement label or control flag, or a missing statement label or control flag or unreferenced labels, flags which remain in the code.	Mod
3.2 Data Operations and Computations		
.1 Missing validity test	Failure to test data imported by a procedure.	Mod
.2 Incorrect data referencing	Conditions which potentially allow a subscript, pointer or index to exceed the boundaries of a declared array or other data structure.	Mod

Table 2.3.1-1 (Continued).

Type	Description	Occurs
.3 Mismatched parameter list	Procedure calls where the parameter or argument list of the calling program differs in number or type from that of the called program unit.	Mod
.4 Definition or initialization fault	Failure to initialize or incorrect initialization, or variables used before they are defined.	Mod
.5 Anachronistic data	A mix of data pertinent to the current iteration and data erroneously included from previous iterations.	System, Sub, Mod
.6 Improperly used data handling construct	Errors in use of data handling constructs such as type mismatches, improper transformations, moves or subsetting.	Sub, Mod
.7 Variable misuse	Any misuse of a variable, either locally or globally.	Sub, Mod
.8 Incompatible data representation	Inconsistency in units of data, e.g., pounds, kilograms.	Mod
.9 Insufficient data transport	Poor handling of input and output operations which have an effect on throughput, i.e., input-output statements, library routines or database, indexing.	System, Sub, Mod
.10 Input-Output faults	Incorrect communication protocols and external data mismatches.	System, Sub, Mod
3.3 Other		
.1 Calls to non-existent subprograms	A call to a subprogram that is not yet in the system.	Mod
.2 Improper program linkages	Involving a variable of one data type in a program being declared as another data type in the calling program, or a mismatch in control information between a called and calling program.	System, Sub, Mod
.3 Failure to implement design element	A design element is missing from the code.	System, Sub, Mod
.4 Improperly implemented design element	Code that does not conform to the definition of its corresponding design element.	System, Sub, Mod
.5 Unintended function	"Extra" code that cannot be mapped to any design element.	System, Sub, Mod

Table 2.3.1-2 Capability of testing techniques to detect defects

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)		Types of Defects Detected (from Table 2.3.1-1)
Requirements/Design Methods		
1.1.1	General Requirements Language Analysis/Processing	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.13 (11) ¹
1.1.2	Mathematical Verification of Requirements	1.1, 1.2, 1.3, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13 (11)
1.1.3	EHDM	1.1, 1.2, 1.3, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8-2.11, 2.14, 2.15 (19)
1.1.4	Z	1.1, 1.2, 1.3, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8-2.11, 2.14, 2.15 (19)
1.1.5	Vienna Definition Method	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8-2.11, 2.14, 2.15 (19)
1.1.6	Refine Specification Language	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8-2.11, 2.14, 2.15 (19)
1.1.7	Higher Order Logic (HOL)	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15 (19)
1.1.8	Concurrent System Calculus	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15 (19)
1.2.1	Ward-Mellor Method	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15 (19)
1.2.2	Hatley-Pirbhai Method	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15 (19)
1.2.3	Harel Method	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15 (19)
1.2.4	Extended Systems Modeling Language	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 (23)
1.2.5	Systems Engineering Methodology	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 (23)
1.2.6	System Requirements Engineering Methodology	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 (23)
1.2.7	FAM	1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.2, 2.4, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 (23)

¹ Numbers in parenthesis are sums

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
1.2.8 Critical Timing/Flow Analysis	1.4, 1.6, 1.8, 2.4, 2.6, 2.7, 3.1.2, 3.1.6, 3.1.9, 3.3.1 (10)
1.2.9 Simulation-Language Analysis	1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.12, 1.13, 2.10-2.12, 2.14 (14)
1.2.10 Petri-Net Safety Analysis	1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.12, 1.13, 2.1, 2.2 (12)
1.2.11 PSL/PSA	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.12, 2.15 (24)
1.3.1 Formalized Requirements Review	All of 1.0 (13)
1.3.2 Formal Design Review	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.12 (23)
1.3.3 System Engineering Analysis	1.1-1.10, 1.12, 1.13, 2.1, 2.2, 2.7, 2.9, 2.11-2.15 (21)
1.3.4 Requirements Analysis	All of 1.0 (13)
1.3.5 Prototyping	1.4, 1.5, 1.6, 1.8-1.13, 2.2, 2.3, 2.5-2.7, 2.10-2.15 (20)
1.3.6 Database Design Analysis	1.2, 2.3 (2)
1.3.7 Operational Concept Design Review	1.13, 2.1, 2.2, 2.7 (4)
1.4.1 Requirements Tracing Analysis	1.1, 1.10, 1.11, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.12 (13)
1.4.2 Design Compliance Analysis	2.4, 2.9, 2.15 (3)
Static Methods	
2.1.1 Analytic Modeling	1.1 thru 1.11, 1.13, 2.1 thru 2.3, 2.5, 2.6, 2.8, 2.9, 2.11, 2.14, 2.15, 3.1.1, 3.2.6, 3.3.1 thru 3.3.4 (28)
2.1.2 Cause-Effect Analysis	1.1, 1.9, 2.6, 2.8, 3.1.2 (5)
2.1.3 Symbolic Execution	1.13, 2.3, 2.9, 3.1.1-3.1.5, 3.1.9, 3.3.1, 3.3.2 (11)
2.1.4 Decision Tables	1.2, 1.5, 1.13, 2.1, 2.2, 2.4, 2.7, 2.11, 2.13, 2.14, 3.1.2, 3.1.3, 3.1.8, 3.2.6, 3.3.3, 3.3.4, 3.3.5 (17)
2.1.5 Trace-Assertion Method	1.1, 1.2, 1.5, 1.11, 1.13, 2.1, 2.2, 2.11, 2.14, 2.15, 3.1.4, 3.1.5, 3.2.6, 3.3.1, 3.3.2, 3.3.3 (16)
2.1.6 Functional Abstraction	1.1, 1.2, 1.5, 1.11, 1.13, 2.14, 2.15, 3.1.5, 3.1.9, 3.2.6, 3.3.1 thru 3.3.4 (14)
2.1.7 L-D Relation Methods	1.1, 1.2, 1.5, 1.11, 1.13, 2.15, 3.1.3 thru 3.1.6, 3.2.6, 3.3.1 thru 3.3.4 (15)
2.1.8 Program Proving	1.7 thru 1.10, 2.1, 2.6, 2.8, 2.9, 2.11, 2.14, 3.1.1-3.1.6, 3.1.8, 3.2.6 (18)
2.1.9 Metric Analysis	3.1.1 thru 3.1.6, (6)

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
2.1.10 Algebraic Specification	1.9, 1.11, 1.12, 1.13, 2.1, 2.4, 2.8, 2.15, 3.1.2, 3.1.3, 3.1.4, 3.2.3, 3.2.4, 3.2.6-3.2.9, all of 3.3 (22)
2.1.11 Induction-Assertion Method	1.9, 1.11, 1.12, 1.13, 2.1, 2.4, 2.8, 2.15, 3.1.2, 3.1.3, 3.1.4, 3.2.3, 3.2.4, 3.2.6-3.2.9, all of 3.3 (22)
2.1.12 Confidence Weights Sensitivity Analysis	2.11, 3.2.6 (2)
2.1.13 Model Evaluation	1.1-1.5, 1.7, 1.9-1.11, 1.13, 2.1-2.3, 2.7, 2.11, 2.14, 3.1.4, 3.2.6-3.2.8, 3.2.10, 3.3.4 (22)
2.2.1 Control Flow Analysis	1.11, 2.4, 2.10, 2.13, All of 3.1 (13)
2.2.2 State Transition Diagram Analysis	1.11, 2.4, 2.10, 2.13, All of 3.1 (13)
2.2.3 Program Control Analysis	All of 3.1, 3.3.1, 3.3.2 (11)
2.2.4 Operational Concept Analysis	1.1 thru 1.6, 2.1, 2.2, 2.11, 2.14 (10)
2.2.5 Calling Structure Analysis	1.11, 2.4, 2.10, 2.11, 2.12, 2.13, All of 3.1 (15)
2.2.6 Process Trigger/Timing Analysis	1.6, 1.7, 1.8, 1.9, 1.12, 2.4, 2.7, 2.10, 2.11 (9)
2.2.7 Worst-Case Timing Analysis	1.8, 1.10, 2.3, 2.5, 2.6, 2.7, 2.8, 2.14 (8)
2.2.8 Concurrent Process Analysis	1.6, 1.7, 1.8, 1.9, 2.5, 2.6, 2.7, 2.14, 3.1.7 (9)
2.3.1 Data Flow Analysis	1.10, 1.11, 1.13, 3.1.4, 3.1.8, All of 3.2 (15)
2.3.2 Signed Directed Graphs	1.6, 1.7, 1.9, 2.5, 2.6, 3.2.2, 3.2.3 (7)
2.3.3 Dependency Analysis	1.6, 1.7, 1.9, 2.5, 2.6, 3.2.2, 3.2.3 (7)
2.3.4 Qualitative Causal Reasoning Analysis	1.6, 1.7, 1.9, 2.5, 2.6, 3.2.2, 3.2.3 (7)
2.3.5 Look-up Table Generator	2.9, All of 3.2 (11)
2.3.6 Data Dictionary Generator	2.9, All of 3.2 (11)
2.3.7 Cross-Reference List Generator	2.9, 3.1.7, 3.1.9, All of 3.2 (13)
2.3.8 Aliasing Analysis	3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.7 (5)
2.3.9 Concurrency Analysis	All of 3.1 (9)
2.3.10 Database Analyzer	All of 3.2 (10)
2.3.11 Database Interface Analyzer	All of 3.2 (10)
2.3.12 Data-Model Evaluation	2.3, 2.4, 2.7, 2.8, 2.9, 2.10, 2.11, 2.14 (8)
2.4.1 Failure Mode, Effects, Causality Analysis	1.2, 1.7, 1.8, 2.1, 2.8, 2.10, 2.11, 3.1.2, 3.1.4, 3.1.7, 3.1.8, 3.1.9, all of 3.2 (22)
2.4.2 Criticality Analysis	1.2, 1.9, 2.8, 3.1.4, 3.1.7, 3.1.9, all of 3.2 (16)
2.4.3 Hazards/Safety Analysis	1.1, 1.2, 1.3, 2.1, 2.2, 2.4, 2.7, 2.11, 2.14, 3.1.4, 3.1.9, all of 3.2, 3.3 (26)
2.4.4 Anomaly Testing	2.3, 2.10, 2.11, 3.2.1, 3.3.3, 3.3.4 (6)

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
2.4.5 Fault-Tree Analysis	1.7, 1.9, 1.12, 2.10, 3.1.2, 3.1.3, 3.1.4, 3.2.1-3.2.4, 3.2.6, 3.2.10 (16)
2.4.6 Failure Modeling	1.2, 1.3, 1.4, 1.6, 1.7, 2.1, 2.3, 2.4, 2.7, 2.11, 2.14, 3.1.4, 3.1.9, all of 3.2, 3.3 (28)
2.4.7 Common-Cause Failure	3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.1.6, 3.1.7, 3.1.8, (7)
2.4.8 Knowledgebase Syntax Checking	1.3, 1.10, 1.11, 1.13, 2.3, 2.11, 2.14, 3.1.1, 3.1.2, 3.1.3, 3.1.5, 3.1.9, 3.2.1-3.2.4, 3.2.6, 3.2.7, 3.2.10, 3.3.3, 3.3.4 (22)
2.4.9 Knowledgebase Semantic Checking	1.3, 1.10, 1.11, 1.13, 2.3, 2.11, 2.14, 3.1.1, 3.1.2, 3.1.3, 3.1.5, 3.1.9, 3.2.1-3.2.4, 3.2.6, 3.2.7, 3.2.8, 3.2.10, 3.3.2, 3.3.3, 3.3.4 (24)
2.4.10 Knowledge Acquisition/Refinement Aid	1.3, 1.7, 1.10, 1.11, 1.13, 2.3, 2.11, 3.2.6, 3.2.8 (9)
2.4.11 Knowledge Engineering Analysis	1.1-1.3, 1.5, 1.7, 1.10, 1.11, 1.13, 2.1-2.4, 2.7, 2.11, 2.14, 3.2.2, 3.2.4, 3.2.6, 3.2.10, 3.3.4 (20)
2.5.1 Informed Panel Inspection	All of 1.0, 2.0 (28)
2.5.2 Structured Walkthroughs	All of 3.1, 3.2.3, 3.2.6, 3.2.7, 3.3.1, 3.3.2 (14)
2.5.3 Formal Customer Review	All of 1.0, all of 2.0 (28)
2.5.4 Clean-room Techniques	All of 1.0, 2.0, all of 3.0 (52)
2.5.5 Peer Code-Checking	2.7, 2.9, 2.11-2.15, 3.1.1, 3.1.2, 3.1.3, 3.1.9, 3.2.3, 3.2.6, 3.2.7, 3.3.1, 3.3.2 (16)
2.5.6 Desk Checking	2.7, 2.9, 2.11-2.15, 3.1.1, 3.1.2, 3.1.3, 3.1.9, 3.2.3, 3.2.6, 3.2.7, 3.3.1, 3.3.2 (16)
2.5.7 Data Interface Inspection	1.4, 1.7, 1.9, 1.10, 1.11, 2.3, 2.9, 2.11, 3.2.1-3.2.4, 3.2.6, 3.2.8, 3.2.10, 3.3.2 (16)
2.5.8 User Interface Inspection	1.4, 1.8, 2.15 (3)
2.5.9 Standards Audit	2.5, 2.6, 2.8, 2.12, 3.3.4 (5)
2.5.10 Requirements Tracing	1.1, 1.2, 1.3, 1.7, 1.9, 1.10, 1.11, 1.12, 1.13, 2.1, 2.2, 2.4, 2.8, 2.9, 2.15 (15)
2.5.11 Software Practices Review	2.9, 3.3.4 (2)
2.5.12 Process Oriented Audits	2.9, 3.3.3 (2)
2.5.13 Standards Compliance	2.9, 3.2.10 (2)
2.5.14 System Engineering Review	1.1-1.10, 1.12, 1.13, 2.1, 2.2, 2.7, 2.9, 2.11-2.15, 3.1.7, 3.2.1, 3.2.8, 3.2.9, 3.2.10, 3.3.3, 3.3.4 (27)
Dynamic Methods	
3.1.1 Unit/Module Testing	1.2, 1.4, 1.9, 1.10, 2.3, 2.5, 2.6, 2.7, 2.8, 2.10, 2.11, 2.12, 2.13, 2.14, 3.1.2-3.1.9, all of 3.2, 3.3 (37)

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
3.1.2 System Testing	1.2, 1.4, 1.9, 1.10, 2.3, 2.5, 2.6, 2.7, 2.8, 2.10, 2.11, 2.12, 2.13, 2.14, 3.1.2-3.1.9, all of 3.2, 3.3 (37)
3.1.3 Compilation Testing	All of 3.1, 3.2.1-3.2.4, 3.2.6, 3.2.10, 3.3.1, 3.3.2 (20)
3.1.4 Reliability Testing	3.1.3, 3.1.4, 3.1.7, all of 3.2, 3.3.2 (14)
3.1.5 Statistical Record-Keeping	3.1.1, 3.1.2, 3.1.5, 3.1.6, 3.1.7, 3.1.9, (6)
3.1.6 Software Reliability Estimation	3.1.1, 3.1.2, 3.1.5, 3.3.1, 3.3.2 (5)
3.1.7 Regression Testing	1.2, 1.4, 1.9, 1.10, 2.3, 2.5, 2.6, 2.7, 2.8, 2.10, 2.11, 2.12, 2.13, 2.14, 3.1.2-3.1.9, all of 3.2, 3.3 (37)
3.1.8 Metric-Based Testing	3.1.2, 3.1.4, 3.1.8, 3.1.9, 3.2.2, 3.2.10, 3.3.2 (7)
3.1.9 Ad Hoc Testing	3.1.2-3.1.4, 3.1.9, 3.2.1-3.2.4, 3.2.6-3.2.9, 3.3.1 (14)
3.1.10 Beta Testing	1.2, 1.4, 1.8, 1.13, 2.1-2.4, 2.7, 2.8, 2.10, 2.11, 2.14, 3.1.4, 3.1.5, 3.2.5-3.2.10 (21)
3.2.1 Random Input Testing	All of 2.0, all of 3.0 (39)
3.2.2 Domain Testing	All of 2.0, all of 3.0 (39)
3.3.1 Specific Functional Requirement Testing	All of 2.0, all of 3.0 (39)
3.3.2 Simulation Testing	1.2, 1.4, 1.6, 1.7, 1.9, 1.10, 1.11, 1.12, 2.3, 2.6, 2.7, 2.10, 2.14, 3.1.2, 3.1.4, 3.1.5, all of 3.2, 3.3, 3.3.3 (33)
3.3.3 Model-Based Testing	2.1-2.8, 2.10, 2.11, 2.14, 3.1.2, 3.1.4, 3.1.7, 3.2.1, 3.2.4, 3.2.7, 3.2.8, 3.2.10 (19)
3.3.4 Assertion Checking	3.1.3, 3.1.4, 3.1.7, 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.6, 3.2.7, 3.2.10, 3.3.2 (11)
3.3.5 Heuristic Testing	All of 2.0, all of 3.0 (39)
3.4.1 Field Testing	All of 2.0, 3.0 (39)
3.4.2 Scenario Testing	All of 2.0, 3.0 (39)
3.4.3 Qualification/Certification Testing	All of 1.0, 2.0, 3.0 (52)
3.4.4 Simulator-Based Testing	All of 1.0, 2.0, 3.0 (52)
3.4.5 Benchmarking	All of 3.1, 3.2 (19)
3.4.6 Human Factors Experimentation	2.7, 2.13, 2.14, 3.2.10 (4)
3.4.7 Validation Scenario Testing	1.2, 1.4, 1.8, 1.13, 2.1-2.4, 2.7, 2.8, 2.10, 2.11, 2.14, 3.1.4, 3.1.5, 3.2.5-3.2.10 (21)
3.4.8 Knowledgebase Scenario Generation	3.1.2, 3.1.4, 3.1.5, 3.1.7, 3.1.9, all of 3.2, 3.3.2-3.3.5 (19)
3.5.1 Stress/Accelerated Life Testing	All of 3.0 (24)

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
3.5.2 Stability Analysis	All of 3.0 (24)
3.5.3 Robustness Testing	All of 3.0 (24)
3.5.4 Limit/Range Testing	2.3, 2.8, 2.10, 2.11, 3.1.3, 3.1.4, 3.1.7, 3.2.1, 3.2.2, 3.2.8 (10)
3.5.5 Parameter Violation	2.3, 2.8, 2.10, 2.11, 3.1.3, 3.1.4, 3.1.7, 3.2.1, 3.2.2, 3.2.3, 3.2.6, 3.2.8, 3.3.2 (13)
3.6.1 Sizing/Memory Testing	1.6, 1.8, 2.5, 2.7 (4)
3.6.2 Timing/Flow Testing	2.4, 2.7, 3.1.2, 3.1.6, 3.1.9, 3.3.1 (6)
3.6.3 Bottleneck Testing	2.7, 2.14, 3.2.9 (3)
3.6.4 Queue Size, Register Allocations, Paging, Etc.	1.4, 1.5, 1.7, 1.8, 2.7, 2.14, 3.2.9, 3.3.4 (8)
3.7.1 Activity Tracing	3.1.1 thru 3.1.9, 3.2.4 (10)
3.7.2 Incremental Execution	All of 3.1, 3.2, 3.3.1, 3.3.2 (21)
3.7.3 Results Monitoring	2.3, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.14, 3.2, 3.3 (24)
3.7.4 Thread Testing	All of 2.0, 3.1.1-3.1.5, 3.2.1-3.2.4, 3.2.6-3.2.10, 3.3.1-3.3.5 (34)
3.7.5 Using Generated Explanations	3.1.4, 3.1.7, 3.2.1, 3.2.2, 3.2.4, 3.2.6-3.2.8, 3.2.10, 3.3.3-3.3.5 (12)
3.8.1 Gold Standard	2.1-2.9, 2.11, 2.14, 3.1.5, all of 3.2, 3.3.3-3.3.5 (25)
3.8.2 Effectiveness Procedures	2.1-2.9, 2.11, 2.14, 3.1.5, all of 3.2, 3.3.3-3.3.5 (25)
3.8.3 Workplace Averages	2.1-2.9, 2.11, 2.14, 3.1.5, all of 3.2, 3.3.3-3.3.5 (25)
3.9.1 Data Interface Testing	1.4, 1.13, 2.1, 2.2, 2.3, 2.9, 2.10, 2.11, 2.14, 3.1.5, 3.1.7, 3.2.1, 3.2.8, 3.2.9, 3.3.3 (15)
3.9.2 User Interface Testing	1.1-1.7, 2.1-2.4, 2.7, 2.10, 2.11, 2.14, 3.1.5, 3.1.7, 3.2.8 (18)
3.9.3 Information System Analysis	1.1-1.7, 2.1-2.4, 2.7, 2.10, 2.11, 2.14, 3.1.5, 3.1.7, 3.2.8 (18)
3.9.4 Operational Concept Testing	1.4, 2.1, 2.2, 2.7, 3.2.10 (5)
3.9.5 Organizational Impact Analysis/Testing	1.1 thru 1.4, 1.10, 1.12, 1.13, 2.1, 2.2, 2.3, 2.8, 2.9, 2.11, 2.14, 2.15 (15)
3.9.6 Transaction-Flow Testing	All of 2.0, 3.0 (39)
3.10.1 Statement Testing	All of 3.0 (24)
3.10.2 Branch Testing	All of 3.0 (24)
3.10.3 Path Testing	All of 3.0 (24)

Table 2.3.1-2 (Continued).

Conventional V&V Testing Technique (from Tables 2.2.1-1, 2.2.2-1, and 2.2.3-1)	Types of Defects Detected (from Table 2.3.1-1)
3.10.4 Call-Pair Testing	3.2.3, 3.2.8, 3.3.2 (3)
3.10.5 Linear Code Sequence and Jump	3.1.5, 3.1.8, 3.2.10, 3.3.2 (4)
3.10.6 Test-Coverage Analyzer	All of 3.0 (24)
3.10.7 Conditional Testing	3.1.3, 3.1.4, 3.2 (12)
3.10.8 Data-Flow Testing	All of 2.0, 3.0 (39)
3.11.1 Error Seeding	3.1.3, 3.1.4, 3.1.7, 3.1.9 (4)
3.11.2 Fault Insertion	All of 3.1, 3.2 (19)
3.11.3 Mutation Testing	All of 3.1, 3.2 (19)

**Table 2.3.1-3. Applicability of conventional techniques
to defects in conventional software**

Software Defects Type	Number of Test Techniques Applicable	Ranking of Defect Coverage
1.0 REQUIREMENTS		
.1 Incomplete decomposition	15	39
.2 Omitted requirement	23	18
.3 Improper translation	19	30
.4 Operational environment incompatibility	17	35.5
.5 Incomplete requirement description	13	47
.6 Infeasible requirement	14	43.5
.7 Conflicting requirement	17	35.5
.8 Incorrect assignment of resources	13	47
.9 Conflicting inter-system specification	15	39
.10 Incorrect or missing external constants	19	30
.11 Incorrect or missing external constants	18	33.5
.12 Overspecification of requirements	12	49.5
.13 Incorrect input or output description	18	33.5
2.0 DESIGN		
.1 Omitted requirement	23	18
.2 Misinterpreted requirement	27	10.5
.3 Data limitation	28	8.5
.4 Unintended design element	21	22
.5 Hardware incompatibility	14	43.5
.6 Software incompatibility	15	39
.7 Poor man-machine	29	6.5
.8 Incorrect analyses of computational error	20	25.5
.9 Noncompliance	21	22
.10 Lack of adequate error traps	23	18
.11 Failure to handle exceptions	35	1
.12 Weak modularity	13	47
.13 Rigid control structure	14	43.5
.14 Missing or incorrect processing priorities	34	2
.15 Breakdown between top-level & detail design	14	43.5

Table 2.3.1-3. 15 (Continued).

Software Defects Type	Number of Test Techniques Applicable	Ranking of Defect Coverage
3.0 CODE		
3.1 Logic and Control		
.1 Unreachable code	12	49.5
.2 Improperly used flow control constructs	20	25.5
.3 Improper predicates	19	30
.4 Improper process sequencing	25	14
.5 Halting problem	24	15
.6 Instruction modification	10	52
.7 Failure to save or restore process communication data	20	25.5
.8 Unauthorized or incorrect recursion	11	51
.9 Labels or control flags	19	30
3.2 Data Operations and Computations		
.1 Missing validity test	28	8.5
.2 Incorrect data referencing	27	10.5
.3 Mismatched parameter list	26	12.5
.4 Definition or initialization fault	26	12.5
.5 Anachronistic data	19	30
.6 Improperly used data handling construct	33	3
.7 Variable misuse	29	6.5
.8 Incompatible data representation	31	5
.9 Insufficient data transport	23	18
.10 Input-output faults	32	4
3.3 Other		
.1 Calls to non-existent subprograms	15	39
.2 Improper program linkages	20	25.5
.3 Failure to implement design element	21	22
.4 Improperly implemented design element	23	18
.5 Unintended function	15	39

1 = defect is covered by the most number of techniques

52 = defect is covered by the least number of techniques

An inverse relationship is assumed between the ease-of-use factors and cost: the greater the ease-of-use, the lower the costs associated with the technique.

All eight factors are associated with a five-point rating scale (1-5), with 1 representing the least value (power or ease-of-use), and 5 representing the greatest value. Table 2.3.2-1 presents the interpretation of each of the five values for each of the eight factors. The assumption is that all eight factors are based on continuous underlying distributions of values, and that these distributions are of the same kind (e.g., Normal). It is also assumed that the definitions of the five points of the eight scales are partitioned into equal parts. It should be noted that there is no evidence that the defects on the list will be encountered with equal frequency. Each of the V&V techniques were rated on these eight scales by two of the authors. Since no objective data concerning the method's characteristics was available, the ratings are based on the authors' judgments, personal software experiences, and extensive review of software literature.

These eight factors were used to rate the 153 V&V techniques. The ratings are shown in columns E-H and K-N of Table 2.3.2-2. These eight rating factors have been combined to develop reasonable Cost-Benefit and Effectiveness measures for each technique, where a higher score indicates a higher cost-benefit or more effective technique. These measures (columns Q and S) are described in detail in Volume 2. The three classes of V&V have been taken into account in computing the effectiveness measures for the various techniques also. For example, the V&V Class 1 systems require the greatest capability for familiarizability and human-computer interaction quality. The weighing of these three factors would thus be increased relative to the other five factors. More detailed information on these procedures is presented in Volume 2.

The methods were rank-ordered according to the V&V class effectiveness measures. The top ten methods for Requirements/Design, Static testing, and Dynamic testing are shown in Table 2.3.2-3. For Requirements/Design, Requirements Analysis was rated the best method, followed by Systems Requirements Engineering and Formal Requirements review.

For Static testing, the top three methods were Knowledge base Semantic Checking, User Interface Inspection, and Formal Customer Review. The top three methods for Dynamic testing were System Testing, Random Testing, and Field Testing. Many of the high ranked methods listed in Table 2.3.2-3 are widely used and accepted as conventional software V&V.

It is important to emphasize that the results of the rankings are determined by the effectiveness metric defined for the three classes of V&V. While there is subjectivity in the ratings of the methods on the eight factors that make up these metrics, the computation of the metric and the resulting selection of top methods is completely determined by the mathematic assumption of the metric -- which can be easily changed if there is some rationale.

2.4 Assessment of Applicability of Methods to Expert System Components

All conclusions and findings on assessment of applicability of methods to expert system components are based on Section 7.0 of Volume 2. The previous sections discussed the results of classifying and characterizing conventional V&V techniques. An examination of whether these V&V techniques are applicable to expert systems is in order. First, the components of expert systems are examined. Next, the V&V characteristics of these components are discussed. Finally, the applicability of the methods to the components are assessed.

Table 2.3.2-1 Interpretation of the 1-5 rating scale values for each of the eight cost/benefit factors

Factors	Rating Scale Values	
	Value	Explanation
Power Factors		
Broad Power	1	0-23% of Applicable defects detectable
	2	24-44% of Applicable defects detectable
	3	45-65% of Applicable defects detectable
	4	66-86% of Applicable defects detectable
	5	87-100% of Applicable defects detectable
Hard Power	1	Not good for finding hard defects
	2	Good at finding a few hard defects
	3	Good at finding several kinds of hard defects
	4	Very good at finding a number of hard defects
	5	Excellent at finding a wide number and variety of difficult defects
Formalizability	1	Not really possible
	2	Partially feasible, but requires extensive effort
	3	Feasible to formalize simply in small software systems
	4	Very feasible
	5	Been done at least once or else designed completely
Human-Computer Interaction Tested	1	Not really at all
	2	Somewhat, as a side effect
	3	Tests some aspects OK
	4	Quite thorough in testing HCI
	5	A primary focus of the technique

Table 2.3.2-1 (Continued).

FACTORS	Rating Scale Values	
	Value	Explanation
Ease-of-Use Factors		
Ease of Mastery	1	Very difficult, requires specialized mathematical or programming skills and then specialized training
	2	Quite difficult, but required skill level and training is somewhat less
	3	Requires concentrated training, but most people can acquire the method without difficulty
	4	Requires only a little training, almost everyone can acquire the skill
	5	Requires virtually no training, anybody can pick it up while using it after a few minutes
Ease of Setup	1	Have to do a considerable amount of programming or specification in some language, days to weeks
	2	Takes a significant amount of programming or specification to set up, a day or so
	3	Takes a moderate amount of time and thought, several hours
	4	Sets up rapidly in an hour or so
	5	Set-up is almost immediate
Ease of Run/ Interpretation	1	Very complicated to run, and the findings take quite a while to interpret, several hours to days
	2	Difficult to run, interpretation requires detailed analysis over a number of hours
	3	Requires some time and care to run; interpretation requires several hours
	4	Quite easy to run, interpretation is accomplished within a few minutes
	5	Completely easy to run, interpretation is almost immediately made
Usage	1	Almost nobody uses it or is even familiar with it
	2	Used by a few, some people have heard of it
	3	Used by quite a few, most people have heard of it
	4	Used by a majority, almost everyone has heard of it
	5	Highly familiar to all and almost always used, no matter what other techniques are employed

2.4.1 Components of Expert Systems

The first component of an expert system is the **Inference Engine**. It is that part of the expert system which determines what gets done next. It controls the interpretations and decisions, and manages the results. The inference engine works on declarative knowledge which is most frequently in the form of IF-THEN rules. The second major component is the **Knowledge Base**. It contains all the specific domain knowledge for a particular application and is the heart of the expert system. There are many possible types of knowledge representation such as rules, frames, objects, facts, external databases, and in-line demons. A detailed discussion of these representations can be found in Section 7.1 of Volume Two. The most common representation is that of IF-THEN Rules, rules which specify a set of conditions to be satisfied before a set of actions can be taken.

The third component of expert systems consists of **Interfaces**. With the exception of the user interface, these are seldom considered in discussions of expert system components since most expert systems have historically been stand-alone systems. However, today expert systems are often data driven embedded systems and have a number of interfaces. The fourth component of expert systems, **Tools and Utilities**, is also seldom considered. They comprise all of the general applications, tools, and programs that were used to develop the implemented expert systems. This includes compilers, linker/loaders, and debugger/code checkers. A very detailed examination of these four components and their sub-components can be found in Section 7.1 of Volume Two.

2.4.2 Key V&V Characteristics of Expert System Components

Table 2.4.2-1 provides a look at the expert system components with respect to conventional Verification and Validation. The first feature examines whether conventional, procedural programming languages such as C, FORTRAN, or Ada are typically used to program the components. This feature is shown in the first column of the table, with a Y signifying Yes, N for No, and S for Sometimes. The second column highlights the reusability feature, whether or not the components can easily be reused over a wide variety of different applications. If the component has to be modified for each new application, it has low reusability. The final column discusses the problems or faults that could plague a component. If enough progress has been made for practitioners to agree that formal approaches are quite promising and should be further pursued, a Y will appear for that component.

2.4.3 Applicability of V&V Methods

Each of the 153 V&V techniques was rated as to its applicability to the four components of expert systems as well as to the system as a whole. A rating of 1 indicates that the technique can be applied to the component directly without modification, while the lowest rating of 4 indicates that the technique is not applicable to that component. Techniques with the highest applicability rating for the expert system components are marked by an asterisk in Table 2.4.3-1.

Table 2.3.2-2: Conventional V and V Techniques, Power and Ease-of-Use Factor Ratings, \Cost-Benefit and Effectiveness Measures

	B	C	E	F	G	H	I	K	L	M	N	O	Q	S	T	U	V
1		TECHNIQUE	P	O	W	E	R		E	A	S	E	O	F	U	S	E
2	Technique		P1	P2	P3	P4	TOTAL	E1	E2	E3	E4	TOTAL	of	BENEFIT	V&V	V&V	V&V
3	Number	NAME	Hard	Broad	Formal	HCI	POWER	Learn	Setup	Run/Interp	Usage	EASE	use	MEASURE	Class 3	Class 2	Class 1
4																	
5	1.0	REQS./DESIGN Methods															
6	1.1	FORMAL METHODS															
7	1.1.1	Gen'l Reqs. Lang. Analysis	2	4	3	2	11	2	4	2	2	10	14	-3	275	263	246
8	1.1.2	Mathematical Verification	3	4	4	1	12	1	2	1	2	6	18	-6	218	247	260
9	1.1.3	EHDM	3	4	4	1	12	1	1	2	1	5	19	-7	197	235	265
10	1.1.4	Z	3	4	4	1	12	1	1	1	1	4	20	-8	185	225	252
11	1.1.5	Vienna Definition Method	3	4	4	1	12	1	1	1	1	4	20	-8	185	225	252
12	1.1.6	Refine Specification Language	4	4	4	2	14	2	2	2	2	8	16	-2	260	300	324
13	1.1.7	Higher Order Logic	2	4	4	2	12	1	2	1	1	5	19	-7	203	220	236
14	1.1.8	Concurrent System Calculus	3	4	4	1	12	1	1	1	1	4	20	-8	185	225	252
15	1.2	SEMI-FORMAL METHODS															
16	1.2.1	Ward-Mellor Method	4	4	4	2	14	2	2	2	3	9	15	-1	280	312	327
17	1.2.2	Hatley-Pirbhai Method	3	4	4	2	13	2	2	1	2	7	17	-4	243	265	277
18	1.2.3	Harel Method	3	4	4	2	13	2	1	2	2	7	17	-4	242	265	285
19	1.2.4	Extended Sys. Model. Lang	3	4	4	2	13	1	2	2	1	6	18	-5	220	255	283
20	1.2.5	Sys. Eng. Methodology	3	4	4	2	13	1	1	2	1	5	19	-6	207	245	278
21	1.2.6	Sys. Req. Eng. Method.	4	4	4	2	14	3	2	2	3	10	14	0	295	320	331
22	1.2.7	FAM	3	4	4	1	12	1	2	2	2	7	17	-5	230	257	273
23	1.2.8	Critical Timing/Flow Analysis	4	2	2	2	10	2	1	3	2	8	16	-6	209	250	276
24	1.2.9	Simulation-Language Analysis	4	3	4	3	14	2	2	3	2	9	15	-1	258	302	342
25	1.2.10	Petri-Net Safety Analysis	3	2	4	1	10	1	2	2	2	7	17	-7	182	221	257
26	1.2.11	PSL/PSA	2	4	3	2	11	1	1	2	2	6	18	-7	221	225	227
27	1.3	REVIEWS AND ANALYSES															
28	1.3.1	Formal Requirements Review	2	5	1	3	11	5	4	4	4	17	7	4	416	345	271
29	1.3.2	Formal Design Review	2	4	1	3	10	5	4	4	4	17	7	3	392	327	263
30	1.3.3	System Engineering Analysis	3	4	1	3	11	1	3	3	2	9	15	-4	272	276	257
31	1.3.4	Requirements Analysis	3	5	2	3	13	4	4	4	4	16	8	5	407	369	321
32	1.3.5	Prototyping	2	4	2	4	12	3	2	4	3	12	12	0	327	296	275
33	1.3.6	Database Design Analysis	3	1	1	1	6	3	2	4	3	12	12	-6	229	230	226
34	1.3.7	Operational Concept Design Review	3	1	1	4	9	3	3	3	3	12	12	-3	260	260	257
35	1.4	TRACEABILITY ANALYSES															
36	1.4.1	Requirements Tracing	2	3	1	3	9	3	4	3	4	14	10	-1	326	283	234
37	1.4.2	Design Compliance Analysis	2	1	1	2	6	3	3	2	2	10	14	-8	203	193	181
38	2.0	STATIC TESTING METHODS															
39	2.1	ALGORITHM ANALYSIS															
40	2.1.1	Analytic Modeling	2	3	1	1	7	2	2	2	1	7	17	-10	193	189	172
41	2.1.2	Cause-effect Analysis	2	1	1	1	5	2	2	2	2	8	16	-11	165	165	159
42	2.1.3	Symbolic Execution	3	2	5	1	11	1	1	2	1	5	19	-8	150	206	269
43	2.1.4	Decision Tables	3	3	3	2	11	2	3	2	3	10	14	-3	263	272	270
44	2.1.5	Trace-assertion Method	4	2	4	2	12	2	1	1	1	5	19	-7	167	232	287
45	2.1.6	Functional Abstraction	2	2	4	1	9	2	1	1	1	5	19	-10	147	172	206
46	2.1.7	L-D Relation Method	3	2	4	2	11	2	1	1	1	5	19	-8	162	207	253
47	2.1.8	Program Proving	4	2	4	1	11	1	2	1	1	5	19	-8	155	224	275

Table 2.3.2-2: Conventional V and V Techniques, Power and Ease-of-Use Factor Ratings, \Cost-Benefit and Effectiveness Measures

	B	C	E	F	G	H	I	K	L	M	N	O	Q	S	T	U	V
1		TECHNIQUE	P	O	W	E	R		E	A	S	E	O	F	U	S	E
2	Technique		P1	P2	P3	P4	TOTAL	E1	E2	E3	E4	TOTAL	Difficulty	COST-BENEFIT	EFFECTIVENESS		
3	Number	NAME	Hard	Broad	Formal	HCI	POWER	Learn	Setup	Run/Interp	Usage	EASE	of use	MEASURE	Class 3	Class 2	Class 1
4																	
48	2.1.9	Metric Analyses	1	1	3	1	6	3	3	3	3	12	12	-6	222	194	190
49	2.1.10	Algebraic Specification	3	2	4	1	10	1	2	1	1	5	19	-9	150	199	241
60	2.1.11	Induction-Assertion Method	3	2	4	1	10	1	1	1	1	4	20	-10	137	169	236
51	2.1.12	Confidence Weights Sensitivity Analysis	2	1	3	1	7	3	2	1	2	8	16	-9	170	177	190
62	2.1.13	Model Evaluation	3	3	2	2	10	2	2	2	1	7	17	-7	209	231	239
63	2.2	CONTROL ANALYSIS															
64	2.2.1	Control Flow Analysis	2	2	2	2	8	3	4	3	5	15	9	-1	313	274	236
65	2.2.2	State Transition Diagram	3	2	3	3	11	2	3	3	3	11	13	-2	261	274	288
66	2.2.3	Program Control Analysis	3	1	2	2	8	3	3	3	5	14	10	-2	281	271	257
67	2.2.4	Operational Concept Analysis	3	1	1	5	10	4	3	3	3	13	11	-1	285	278	274
68	2.2.5	Call Structure Analysis	3	2	2	2	9	4	3	3	5	15	9	0	320	297	269
69	2.2.6	Process Trigger/Timing Analysis	4	1	2	2	9	2	2	3	3	10	14	-5	218	254	276
60	2.2.7	Worst-case Timing Analysis	2	1	3	1	7	2	2	3	2	9	15	-8	179	189	212
61	2.2.8	Concurrent Process Analysis	4	1	2	2	9	2	2	3	1	8	16	-7	178	230	270
62	2.3	DATA ANALYSIS															
63	2.3.1	Data Flow Analysis	3	2	3	2	10	2	1	3	4	10	14	-4	245	256	268
64	2.3.2	Signed Directed Graphs	3	1	2	2	8	2	3	3	2	10	14	-6	206	227	244
65	2.3.3	Dependency Analysis	2	1	3	2	8	2	3	4	3	12	12	-4	234	231	246
66	2.3.4	Qualitative Causal Analysis	3	1	3	2	9	2	3	3	2	10	14	-5	207	234	264
67	2.3.5	Look-up Table Generator	2	1	2	1	6	4	5	4	4	17	7	-1	299	262	234
68	2.3.6	Data Dictionary Generator	2	1	2	1	6	4	4	4	4	16	8	-2	286	252	229
69	2.3.7	Cross-reference List Generator	2	2	2	1	7	4	5	4	4	17	7	0	323	280	242
70	2.3.8	Aliasing Analysis	3	1	2	2	8	4	4	3	2	13	11	-3	249	253	257
71	2.3.9	Concurrency Analysis	2	1	2	1	6	2	2	2	1	7	17	-11	146	160	176
72	2.3.10	Database Analysis	2	1	2	1	6	2	3	3	3	11	13	-7	211	204	200
73	2.3.11	Database Interface Analysis	2	1	2	1	6	3	4	3	3	13	11	-5	239	222	209
74	2.3.12	Data-Model Evaluation	3	1	2	1	7	2	3	3	2	10	14	-7	196	217	231
75	2.4	FAULT/FAILURE ANALYSIS															
76	2.4.1	Failure-mode Effects Caus. Analysis	5	2	1	2	10	2	2	2	2	8	16	-6	214	268	282
77	2.4.2	Criticality Analysis	4	2	1	2	9	2	2	2	2	8	16	-7	209	243	248
78	2.4.3	Hazards/Safety Analysis	3	3	1	2	9	2	2	2	3	9	15	-6	248	248	225
79	2.4.4	Anomaly Testing	3	1	3	3	10	2	2	3	3	10	14	-4	224	246	275
80	2.4.5	Fault-Tree Analysis	4	2	1	2	9	2	2	2	2	8	16	-7	209	243	248
81	2.4.6	Failure Modeling	4	3	2	2	11	3	2	2	1	8	16	-5	229	264	277
82	2.4.7	Common-cause Failure Analysis	3	1	2	2	8	3	2	2	2	9	15	-7	196	215	230
83	2.4.8	Knowledgebase Syntax Checking	4	2	5	2	13	4	3	3	2	12	12	1	268	307	354
84	2.4.9	Knowledgebase Semantic Checking	5	3	5	3	16	4	2	4	1	11	13	3	286	348	414
85	2.4.10	Knowledge Acquisition/Refinement Aid	2	1	4	2	9	3	4	2	1	10	14	-5	199	212	243
86	2.4.11	Knowledge Engineering Analysis	3	2	3	2	10	2	2	2	2	8	16	-6	206	232	254
87	2.5	INSPECTIONS															
88	2.5.1	Informed Panel Inspection	3	3	1	2	9	5	4	4	3	16	8	1	343	312	273
89	2.5.2	Structured Walk-throughs	3	2	1	2	8	4	5	4	4	17	7	1	337	308	269
90	2.5.3	Formal Customer Review	3	3	1	2	9	5	4	4	4	17	7	2	363	324	276

Table 2.3.2-2: Conventional V and V Techniques, Power and Ease-of-Use Factor Ratings, \Cost-Benefit and Effectiveness Measures

	B	C	E	F	G	H	I	K	L	M	N	O	Q	S	T	U	V
1		TECHNIQUE	P	O	W	E	R										
2	Technique		P1	P2	P3	P4	TOTAL	E1	E2	E3	E4	TOTAL	Difficulty	COST-	EFFECTIVENES		
3	Number	NAME	Hard	Broad	Formal	HCI	POWER	Learn	Setup	Run/interp	Usage	EASE	of use	BENEFI	V&V	V&V	V&V
4														MEASURE	Class 3	Class 2	Class 1
91	2.5.4	Clean-room Techniques	5	5	2	2	14	2	3	2	2	9	15	-1	300	339	331
92	2.5.5	Peer Code Checking	3	2	1	2	8	4	5	4	4	17	7	1	337	308	269
93	2.5.6	Desk Checking	2	2	2	2	8	4	5	5	5	19	5	3	365	312	271
94	2.5.7	Data Interface Inspection	3	2	1	3	9	5	4	4	3	16	8	1	329	304	278
95	2.5.8	User Interface Inspection	3	1	1	5	10	4	4	4	4	16	8	2	330	310	295
96	2.5.9	Standards Audit	2	1	1	2	6	3	3	3	3	12	12	-6	235	215	197
97	2.5.10	Requirements Tracing	3	2	2	3	10	4	3	3	4	14	10	0	310	295	279
98	2.5.11	Software Practices Review	1	1	1	1	4	3	3	3	3	12	12	-8	220	180	150
99	2.5.12	Process Oriented Audits	1	1	1	1	4	4	3	3	3	13	11	-7	235	188	154
100	2.5.13	Standards Compliance	2	1	3	2	8	3	3	3	3	12	12	-4	237	229	237
101	2.5.14	System Engineering Review	3	3	1	3	10	1	3	3	2	9	15	-5	248	258	249
102																	
103																	
104	3.0	DYNAMIC TESTING METHODS															
105	3.1	GENERAL TESTING															
106	3.1.1	Unit/Module Testing	2	4	2	1	9	5	2	4	4	15	9	0	347	294	247
107	3.1.2	System Testing	3	4	2	3	12	5	5	5	5	20	4	8	443	391	338
108	3.1.3	Compilation Testing	2	2	2	2	8	5	5	5	3	18	6	2	340	296	269
109	3.1.4	Reliability Testing	2	1	2	2	7	4	4	5	3	16	8	-1	288	260	252
110	3.1.5	Statistical Record-Keeping	2	1	2	1	6	4	2	4	2	12	12	-6	220	208	213
111	3.1.6	Software Reliability Estimation	2	1	1	1	5	3	2	2	2	9	15	-10	180	173	163
112	3.1.7	Regression Testing	2	4	2	3	11	5	5	4	5	19	5	6	426	356	291
113	3.1.8	Metric-based Testing	1	1	1	1	4	3	2	5	2	12	12	-8	211	178	168
114	3.1.9	Ad-hoc Testing	1	1	1	2	5	5	5	5	3	18	6	-1	310	246	207
115	3.1.10	Beta Testing	2	2	1	3	8	4	4	3	2	13	11	-3	277	249	224
116	3.2	SPECIAL INPUT TESTING															
117	3.2.1	Random Testing	4	4	2	2	12	3	4	5	4	16	8	4	375	368	343
118	3.2.2	Domain Testing	2	4	2	2	10	3	3	4	3	13	11	-1	320	288	254
119	3.3	FUNCTIONAL TESTING															
120	3.3.1	Functional Reqs. Testing	3	4	1	3	11	4	3	4	3	14	10	1	349	322	285
121	3.3.2	Simulation Testing	4	3	3	2	12	3	1	4	3	11	13	-1	281	305	324
122	3.3.3	Model-based Testing	4	2	4	3	13	2	2	3	2	9	15	-2	234	284	334
123	3.3.4	Assertion Checking	3	1	4	1	9	1	2	1	2	6	18	-9	146	193	236
124	3.3.5	Heuristic Testing	4	4	2	3	13	3	3	3	3	12	12	1	328	336	322
125	3.4	REALISTIC TESTING															
126	3.4.1	Field Testing	3	4	1	3	11	5	4	5	5	19	5	6	429	374	313
127	3.4.2	Scenario Testing	3	4	2	3	12	4	3	5	3	15	9	3	362	339	318
128	3.4.3	Qualification/Certification	4	5	1	3	13	3	3	5	3	14	10	3	375	367	336
129	3.4.4	Simulator-based Testing	4	3	3	3	13	3	2	3	3	11	13	0	292	315	329
130	3.4.5	Benchmarking	2	2	1	1	6	3	2	3	3	11	13	-7	236	213	187
131	3.4.6	Human Factors Experimentation	3	1	1	5	10	3	3	3	3	12	12	-2	270	270	270
132	3.4.7	Validation Scenario Testing	2	2	1	3	8	4	3	3	3	13	11	-3	284	251	222
133	3.4.8	Knowledgebase Scenario Generation	3	2	4	3	12	4	4	4	2	14	10	2	297	305	331

Table 2.3.2-2: Conventional V and V Techniques, Power and Ease-of-Use Factor Ratings, Cost-Benefit and Effectiveness Measures

	B	C	E	F	G	H	I	K	L	M	N	O	Q	S	T	U	V
1		TECHNIQUE	P	O	W	E	R						Difficulty	COST-	EFFECTIVENESS		
2	Technique		P1	P2	P3	P4	TOTAL	E1	E2	E3	E4	TOTAL	of	BENEFIT	V&V	V&V	V&V
3	Number	NAME	Hard	Broad	Formal	HCI	POWER	Learn	Setup	Run/Interp	Usage	EASE	use	MEASURE	Class 3	Class 2	Class 1
4																	
134	3.6	STRESS TESTING															
135	3.5.1	Stress/Accelerated Life Testing	3	3	1	1	8	5	3	4	3	15	9	-1	320	292	255
136	3.5.2	Stability Analysis Testing	3	3	1	2	9	2	3	3	3	11	13	-4	273	268	243
137	3.5.3	Robustness Testing	4	3	1	1	9	5	3	5	2	15	9	0	317	315	299
138	3.5.4	Limit/Range Testing	3	1	2	1	7	5	3	5	3	16	8	-1	285	273	272
139	3.5.5	Parameter Violation	3	1	2	2	8	4	3	5	3	15	9	-1	280	275	281
140	3.6	PERFORMANCE TESTING															
141	3.6.1	Sizing/Memory Testing	2	1	1	1	5	4	2	4	3	13	11	-6	239	213	196
142	3.6.2	Timing/Flow Testing	4	1	1	2	8	4	1	3	3	11	13	-5	234	253	259
143	3.6.3	Bottleneck Testing	3	1	1	2	7	4	1	3	3	11	13	-6	229	228	225
144	3.6.4	Queue size, etc.	3	1	1	1	6	4	2	3	3	12	12	-6	232	228	217
145	3.7	EXECUTION TESTING															
146	3.7.1	Activity Tracing	3	1	1	1	6	4	2	5	4	15	9	-3	276	260	246
147	3.7.2	Incremental Execute	4	2	1	1	8	4	2	4	4	14	10	-2	293	293	275
148	3.7.3	Results Monitoring	3	3	1	3	10	4	2	4	4	14	10	0	332	306	275
149	3.7.4	Thread Testing	3	3	2	1	9	4	2	3	3	12	12	-3	281	271	253
150	3.7.5	Using Generated Explanations	3	1	2	2	8	3	2	3	3	11	13	-5	228	237	246
151	3.8	COMPETENCY TESTING															
152	3.8.1	Gold Standard Testing	2	3	1	4	10	4	3	3	3	13	11	-1	318	279	243
153	3.8.2	Effectiveness Procedures	2	3	1	3	9	4	3	4	1	12	12	-3	280	255	237
154	3.8.3	Workplace Averages	2	3	1	2	8	4	3	5	3	15	9	-1	322	279	243
155	3.9	ACTIVE INTERFACE TEST															
156	3.9.1	Data Interface Testing	3	2	2	3	10	4	2	5	4	15	9	1	321	305	300
157	3.9.2	User Interface Testing	3	2	1	5	11	4	3	4	5	16	8	3	361	330	301
158	3.9.3	Information System Analysis	2	2	1	4	9	3	3	3	3	12	12	-3	279	253	231
159	3.9.4	Operational Concept Testing	3	1	1	5	10	5	4	5	3	17	7	3	337	316	309
160	3.9.5	Organizational Impact Analysis	1	2	1	4	8	4	4	4	3	15	9	-1	314	256	219
161	3.9.6	Transaction-flow Test	3	4	2	4	13	3	3	3	2	11	13	0	313	309	298
162	3.10	STRUCTURAL TESTING															
163	3.10.1	Statement Testing	2	3	1	1	7	4	1	4	3	12	12	-5	274	239	207
164	3.10.2	Branch Testing	2	3	1	1	7	4	1	4	3	12	12	-5	274	239	207
165	3.10.3	Path Testing	2	3	1	1	7	4	1	4	2	11	13	-6	254	227	204
166	3.10.4	Call-pair Testing	2	1	1	1	5	4	1	4	3	12	12	-7	226	203	191
167	3.10.5	Linear Code Sequence	3	1	1	1	6	4	1	4	2	11	13	-7	211	216	222
168	3.10.6	Test Coverage Analysis	2	3	1	3	9	4	3	3	3	13	11	-2	308	269	230
169	3.10.7	Conditional Testing	2	1	1	1	5	4	1	4	3	12	12	-7	226	203	191
170	3.10.8	Data-Flow Testing	3	4	2	3	12	3	2	3	3	11	13	-1	310	301	283
171	3.11	ERROR INTRODUCTION															
172	3.11.1	Error Seeding	2	1	1	1	5	4	2	4	3	13	11	-6	239	213	196
173	3.11.2	Fault Insertion	2	1	1	1	5	4	2	4	2	12	12	-7	219	201	193
174	3.11.3	Mutation Testing	2	1	1	1	5	4	2	3	2	11	13	-8	207	191	180

**Table 2.3.2-3 Overall highest ranked conventional
V&V techniques for all V&V classes**

Conventional V&V Techniques	Cost Benefit Metric ¹ Ranking	Effectiveness Metric Ranking			Sum of Ranks	Ranking Over All 4 Metrics ²
		Class 1	Class 2	Class 3		
Requirement Analysis	1	5	1	2	9	1
Systems Requirements Engineering	4.5	2	4	6	16.5	2
Formal Requirements Review	2	13	2	1	18	3
Ward-Mellor Method	6	3	5	7	21	4
Formal Design Review	3	15	3	3	24	5
Simulation Language Analysis	6	1	6	12	25	6
Prototyping	4.5	11	8	4	27.5	7
Refine Specification Language	9	4	7	10.5	30.5	8
Harel Method	13	6	11.5	14	44.5	9
Requirements Tracing	6	25	9	5	45	10
Knowledgebase Semantic Checking	14.5	1	1	14.5	18	1
User Interface Inspection	6	4	6	6	19.5	2
Formal Customer Review	2	11.5	3	2	20	3
Desk Checking	1	17	4.5	1	24	4
Informed Panel Inspection	3	16	4.5	3	30.5	5
Clean-room Techniques	12	3	2	12	31.5	6
Data Interface Inspection	7	9	10	7	33	7
Knowledgebase Syntax Checking	18	2	9	18	36	8
Structured Walk-Throughs	4.5	21.5	7.5	4.5	40.5	9.5
Peer Code Checking	4.5	21.5	7.5	4.5	40.5	9.5

Table 2.3.2-3 (Continued).

Conventional V&V Techniques	Cost Benefit Metric ¹ Ranking	Effectiveness Metric Ranking			Sum of Ranks	Ranking Over All 4 Metrics ²
		Class 1	Class 2	Class 3		
System Testing	1	2	1	1	5	1
Random Testing	4	1	3	4. 5	1 2 5	2
Field Testing	2. 5	1 0	2	2	1 6 5	3
Qualification Certification	6. 5	3	4	4. 5	1 8	4
Regression Testing	2. 5	1 6	5	3	2 6 5	5
Scenario Testing	6. 5	9	6	6	2 7 5	6
User Interface Testing	6. 5	1 2	8	7	3 3 5	7
Operational Concept Testing	6. 5	1 1	1 0	1 1	3 8 5	8
Heuristic Testing	1 2	8	7	1 3	4 0	9
Functional Requirements Testing	1 2	1 7	9	8	4 6	10

¹ Based on the cost benefit measure (column 5) of Table 2.3.2-2, all the methods (for a phase) have been ranked. Note that only a subset are shown here.

² Based on the sums of rankings for the four metrics, ordered from lowest to highest.

Table 2.4.2-1 Components and typical testing-related features of knowledge-based systems, with testing recommendations

Components	Features ¹		
	Typically Written in or Involving Procedural Code	High Reusability Across Different Applications	Potential Defects Known, Fairly Delineated, & Amenable to Formal Test Methods
Inference Engine	Y²	Y	S
Pattern Matcher	Y	Y	S
Conflict-Set Handler	Y	Y	S
Proof Procedure	Y	Y	S
Proof Manager	S	S	N
Fuzzy Value/Uncertainty Handler	S	Y	S
Knowledge Processor	S	Y	S
Inheritance Processor	Y	Y	S
Agenda Handler	Y	Y	S
Knowledge Base	N	N	S
Rules	N	N	Y
Frames	N	N	Y
Objects	M	N	S
Facts	N	N	S
External Database	M	N	S
In-line Demons, Procedures, Functions	Y	N	N
Interfaces	Y	S	N
User Interface	Y	S	N
Database Interface	Y	S	N
Data I/O Channels	Y	S	S
Communication Ports	Y	N	N
Hyper-media	Y	N	N
Interfaces to Other Applications and the Operating System	Y	S	N
Tools and Utilities	Y	Y	N
Compilers	Y	Y	S
Linkers/Loaders	Y	Y	N
Debuggers/Code Checkers	Y	Y	N
Knowledge Engineering Tools	Y	Y	N
Graphics User Interface (GUI)	Y	Y	N
Capability	Y	Y	N
Expert System Shell	Y	N	N
Custom Code			

¹ Explanation of table entries: Y=Yes, N=No, S=Sometimes, M=Mixed procedural and non-procedural code

² The overall "average" of the cell

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
1.1.1 General Requirements Language Analysis/Processing	1	3	1	1	1
1.1.2 Mathematical Verification of Requirements	1	3	1	1	1
1.1.3 EHDM	1	2-3	1	1	1
1.1.4 Z	1	2-3	1	1	1
1.1.5 Vienna Definition Method	1	3	1	1	1
1.1.6 Refine Specification Language	1	3	1	1	1
1.1.7 Higher Order Logic (HOL)	1	3	1	1	1
1.1.8 Concurrent System Calculus	1	3	1	1	1
1.2.1 Ward-Mellor Method	1	3	1	1	1
1.2.2 Hatley-Pirbhai Method	1	3	1	1	1
1.2.3 Harel Method	1	3	1	1	1
1.2.4 Extended Systems Modeling Language	1	3	1	1	1
1.2.5 Systems Engineering Methodology	1	3	1	1	1
1.2.6 System Requirements Engineering Methodology	1	2-3	1	1	1
1.2.7 FAM	1	3	1	1	1
1.2.8 Critical Timing/Flow Analysis	1	3	1	1	1
1.2.9 Simulation-Language Analysis	1	3	1	1	1
1.2.10 Petri-Net Safety Analysis	1	3	1	1	1
1.2.11 PSL/PSA	1	3	1	1	1
1.3.1 Formalized Requirements Review	1	1	1	1	1*
1.3.2 Formal Design Review	1	1	1	1	1*

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors

* Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
1.3.3 System Engineering Analysis	1	3	1	1	1*
1.3.4 Requirements Analysis	1	1	1	1	1*
1.3.5 Prototyping	1	1	1	1	1
1.3.6 Database Design Analysis	4	2	3-4	3-4	3
1.3.7 Operational Concept Design Review	3	1-2	2	2	2
1.4.1 Requirements Tracing Analysis	1*	1*	1*	1*	1*
1.4.2 Design Compliance Analysis	1	1	1	1	1
2.1.1 Analytic Modeling	1	2	2	1	1
2.1.2 Cause-Effect Analysis	1	3	2	1	1
2.1.3 Symbolic Execution	1	4	2	1	1
2.1.4 Decision Tables	1	3	2	1	1
2.1.5 Trace-Assertion Method	1	4	2	1	1
2.1.6 Functional Abstraction	1	4	2	1	1
2.1.7 L-D Relation Methods	1	4	2	1	1
2.1.8 Program Proving	1	4	2	1	1
2.1.9 Metric Analysis	1	3	2	1	1
2.1.10 Algebraic Specification	1	4	2	1	1
2.1.11 Introduction-Assertion Method	1	4	2	1	1
2.1.12 Confidence Weights Sensitivity Analysis	3	2	3	3	2
2.1.13 Model Evaluation	4	1-2	3	4	2
2.2.1 Control Flow Analysis	1	3	2	1	1
2.2.2 State Transition Diagram Analysis	1	2* 0	1	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors

• Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
2.2.4 Operational Concept Analysis	1	3	3	1	1*
2.2.5 Calling Structure Analysis	1	4	3	1	1
2.2.6 Process Trigger/Timing Analysis	1	3	1	1	1
2.2.7 Worst-Case Timing Analysis	1	4	1	1	1
2.2.8 Concurrent Process Analysis	1	4	1	1	1
2.3.1 Data Flow Analysis	1*	2	1	1	1
2.3.2 Influence Diagrams	1	2	2	1	1
2.3.3 Dependency Analysis	1	2	1	1	1
2.3.4 Qualitative Causal Reasoning Analysis	1	2	1	1	1
2.3.5 Look-up Table Generator	1	4	3	1	1
2.3.6 Data Dictionary Generator	1	2	3	1	1
2.3.7 Cross-Reference List Generator	1	3	2	1	1
2.3.8 Aliasing Analysis	1	4	2	1	1
2.3.9 Concurrency Analysis	1	4	1	1	1
2.3.10 Database Analysis	1	3	1*	1	1
2.3.11 Database Interface Analyzer	1	4	1*	1	1
2.3.12 Data-Model Evaluation	1	4	1*	1	1
2.4.1 Failure Mode, Effects, Causality Analysis	1	3	2	1	1
2.4.2 Criticality Analysis	1	3	2	1	1
2.4.3 Hazards/Safety Analysis	1	3	2	1	1
2.4.4 Anomaly Testing	1	2*	1	1	1
2.4.5 Fault-Tree Analysis	1	3	2	1	1
2.4.6 Failure Modeling	1	3	1	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors
 • Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
2.4.7 Common-Cause Failure	1	3	2	1	1
2.4.8 Knowledgebase Syntax Checking	4	1	4	4	1*
2.4.9 Knowledgebase Semantic Checking		41	4	4	1*
2.4.10 Knowledge Acquisition/Refinement Aid	4	1	4	4	1
2.4.11 Knowledge Engineering Analysis	4	1	4	4	1
2.5.1 Informed Panel Inspection	1	2	1	1	1
2.5.2 Structured Walkthroughs	1	3	1	1	1
2.5.3 Formal Customer Review	1	4	1	1	1
2.5.4 Clean-room Techniques	1	4	1	1	1
2.5.5 Peer Code-Checking	1	3	1	1	1
2.5.6 Desk Checking	1	1	1	1	1
2.5.7 Data Interface Inspection	1	2	1*	1	1*
2.5.8 User Interface Inspection	1	2	1*	1	1*
2.5.9 Standards Audit	1	2	1*	1	1*
2.5.10 Requirements Tracing	1*	1*	1*	1*	1*
2.5.11 Software Practices Review	2	2	1	1	1
2.5.12 Process Oriented Audits	1	1	1	1	1
2.5.13 Standards Compliance	1	1	1	1	1
2.5.14 System Engineering Review	1	2	1	1	1
3.1.1 Unit/Module Testing	1	2	1	1	1
3.1.2 System Testing	1	3	1	1	1
3.1.3 Compilation Testing	1	4	2	1	1
3.1.4 Reliability Testing	1	4	2	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors

• Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
3.1.5 Statistical Record-Keeping	1	4	2	1	1
3.1.6 Software Reliability Estimation	1	4	2	1	1
3.1.7 Regression Testing	1*	1*	1*	1*	1*
3.1.8 Metric-Based Testing	2	2	3	1	1
3.1.9 Ad Hoc Testing	4	4	4	4	4
3.1.10 Beta Testing	1	1	1	1	1*
3.2.1 Random Input Testing	1	1	1	1	1
3.2.1.1 Uniform Whole Program Testing	1*	1*	1*	1*	1*
3.2.1.2 Uniform Boundary Testing	1	1	1	1	1
3.2.1.3 Gaussian Whole Program Testing	1	1	1	1	1
3.2.1.4 Gaussian Boundary Testing	1	1	1	1	1
3.2.2 Domain Testing	1	1	1	1	1
3.2.2.1 Equivalence Partitioning	1	1	1	1	1
3.2.2.2 Boundary-value Testing	1	1	1	1	1
3.2.2.3 Category-Partition Method	1	1	1	1	1
3.2.2.4 Revealing Subdomains Method	1	1	1	1	1
3.3.1 Specific Functional Requirement Testing	1*	1*	1*	1*	1*
3.3.2 Simulation Testing	1*	3	1	1	1
3.3.3 Model-Based Testing	1	2*	1	1	1
3.3.4 Assertion Checking	1	4	2	1	1
3.3.5 Heuristic Testing	1	3	1	1	1*
3.4.1 Field Testing	1	1	1	1	1
3.4.2 Scenario Testing	1	1	1	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors
 • Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
3.4.3 Qualification/Certification Testing	1*	1	1	1*	1
3.4.4 Simulator-Based Testing	1	3	1	1	1
3.4.5 Benchmarking	1*	3	1	1*	1
3.4.6 Human Factors Experimentation	1	1*	1*	1	1
3.4.7 Validation Scenario Testing	3	2	2	4	1
3.4.8 Knowledgebase Scenario Generation	4	1	4	4	4
3.5.1 Stress/Accelerated Life Testing	1	3	1	1	1
3.5.2 Stability Analysis	1	2	1	1	1
3.5.3 Robustness Testing	1	2	1	1	1
3.5.4 Limit/Range Testing	1	1*	1	1	1
3.5.5 Parameter Violation	1	2	1	1	1
3.6.1 Sizing/Memory Testing	1	4	1	1	1
3.6.2 Timing/Flow Testing	1	4	1	1	1
3.6.3 Bottleneck Testing	1	4	1	1	1
3.6.4 Queue Size, Register Allocations, Paging, Etc.	1	4	2	1	1
3.7.1 Activity Tracing	1	2*	1	1	1
3.7.2 Incremental Execution	1	2	1	1	1
3.7.3 Results Monitoring	1	2	1	1	1
3.7.4 Thread Testing	1	3	1	1	1
3.7.5 Using Generated Explanations	4	1	4	4	4
3.8.1 Gold Standard	1	1	1	1	1
3.8.2 Effectiveness Procedures	1	1	1	1	1
3.8.3 Workplace Averages	1	1	1	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors

* Technique with highest applicability rating

Table 2.4.3-1 Rating of the Applicability of Conventional Techniques to Expert Systems and their Components (Continued)^{1,2}

Conventional V&V Testing Technique	Expert System Components				
	Inference Engine	Knowledge Base	Inter-faces	Tools & Utilities	Total System
3.9.1 Data Interface Testing	1	2	1*	1	1
3.9.2 User Interface Testing	1	1	1*	1	1
3.9.3 Information System Analysis	1	4	1	1	1
3.9.4 Operational Concept Testing	1	2	1	1	1
3.9.5 Organizational Impact Analysis/Testing	1	4	2	1	1
3.9.6 Transaction-Flow Testing	1	2	1	1	1
3.10.1 Branch Testing	1	2	1	1	1
3.10.2 Path Testing	1	2	1	1	1
3.10.3 Statement Testing	1	3	1	1	1
3.10.4 Call-Pair Testing	1	4	2	1	1
3.10.5 Linear Code Sequence and Jump	1	4	2	1	1
3.10.6 Test-Coverage Analyzer	1	2	1	1	1
3.10.7 Conditional Testing	1	1*	1	1	1
3.10.8 Data-Flow Testing	1	2	1	1	1
3.11.1 Error Seeding	1	1	1	1	1
3.11.2 Fault Insertion	1	2	1	1	1
3.11.3 Mutation Testing	1	2	1	1	1

¹ Ratings are on a 1-4 scale: 1 = the method can be used directly without any modification, 2 = the method largely applies, but some modifications are necessary, 3 = the general concept of the method applies but extensive changes are needed, 4 = the method does not really apply at all

² Ratings are the subjective evaluation of the authors

* Technique with highest applicability rating

3 KNOWLEDGE BASE CERTIFICATION

Earlier activities on this project identified the knowledge base (KB) as the one component of artificial intelligence (AI) systems, particularly expert systems, which could not be adequately certified by conventional software verification and validation (V&V) techniques (see Section 7.4.1 of Volume 2.) The KB contains detailed subject matter information which can be in the form of "IF-THEN" rules, "frames" of associated attribute-value information and simple facts.

Because the KB contains most of the application-specific information ("knowledge") of the system, it would be very efficient if there were V&V techniques which could be applied directly to the KB and assess its quality without having to exercise the whole system dynamically. This is the sense of certification as used in this section: establish the quality level of the KB as it stands alone, separate from system testing.

Clearly, the only techniques which can be employed for such KB certification are static testing ones. The major concern of this activity was to go beyond the judgments of effectiveness discussed in Section 2.0 and actually assess how effective selected methods are in practice. That is, given a rationale for selecting promising KB analysis methods (as addressed in 3.1 below), how effective are these really when used by a representative group of users for actual non-trivial KBs from existing nuclear utility expert systems?

In order to answer this question, a three-day experiment was developed, conducted, and analyzed for this project. The experiment used a group of 20 participants from three operating nuclear power plant utilities, the USNRC Technical Training Center, a national laboratory, university graduate students, and engineers and scientists from SAIC. Along with these participants, sections from the KB of the two expert systems (ES), denoted System A and B, which were provided by the USNRC and EPRI, were used in this experiment.

Based on the knowledge base experiment results, the following conclusions were drawn:

- 1) The use of aids or tools for static knowledge base certification results in a significant improvement in detecting failure and completing the effort is less time.
- 2) The knowledge checking and requirements grouping tools showed the greatest improvement in performance
- 3) Performance was not improved by test experience and extensive subject matter experience
- 4) Even using tools and subject matter experts, some seeded faults were not found Section 3.3 shows the justification for these findings.

3.1 Methods Chosen For Evaluation

In deciding which methods should be used for the KB certification experiment, two major issues were first considered. The first issue was whether static testing could adequately certify the KB without dynamic testing. The second issue was to determine the method for selecting specific V&V techniques to be included in this experiment.

Earlier project activities indicated that static methods are less expensive, more practical to accomplish, and more amenable to automation than dynamic testing (see Section 7.4.2 of Volume 2). Dynamic testing would require usually unavailable or costly drivers or simulators. In addition, KB dynamic testing would require a perfect "oracle" to be present to review all test results and identify errors. Finally, dynamic KB certification would require that a large number of test cases be both generated and run to encompass all precursor conditions and states which is a very complex process. Those KB errors which can only be detected by using the inference engine can be found with static methods

that account for the inference engine's prioritization mechanism and the structure of the KB. This analysis substantiated the sole use of static methods for the experiment (see Section 2.2 of Volume 4).

To select specific V&V techniques for KB certification, information from the earlier completed survey and assessment of conventional and AI software V&V methods was utilized. The software V&V methods were rated on eight factors which encompassed power and ease-of-use as well as cost-benefit, which were then evaluated for three classes of V&V. Although both System A and B were classified as V&V Class 2, methods from V&V Class 1 and 3 were considered to ensure that a full range of potential techniques were included in this activity. Not considered in the selection of methods were the specific aspects of the two test systems, systems A and B, and whether they have specific features such as recovery blocks after failure (they don't, but that was irrelevant.)

The actual method used to identify the top V&V methods was to rank-order them on three separate measures -- their cost-benefit rating, their Class 1, and their Class 3 effectiveness (using Table 2.3.2-2 on page 47); the measure with the highest value was ranked "1", the next "2", and so forth. The three resulting rank measures were summed for each method, and then the methods were sorted in terms of this sum. The method with the largest sum was ranked "1", the next "2", and so forth. Table 3.1-1 shows the top 15 static-testing techniques resulting from this procedure, with the summed ranking shown in the last column (other columns show the individual ranks for the three measures).¹

Of the top ten static V&V methods identified in Table 3.1-1 there is one which can be immediately argued as having limited utility as a V&V procedure for expert system knowledge bases. This method is Data Interface Inspection. Knowledge bases typically contain very few aspects of direct interaction with data interfaces (these are typically accomplished via calls to special data-access routines programmed in conventional software languages). To a somewhat lesser extent, User Interface Inspection is similarly ill-satisfied by inspection of the knowledge base component alone, since all of the screen handling, most of the message presentation decisions, and much of the information context is accomplished by expert system components other than the knowledge base.

For the remaining eight top-rated methods, there is one final selection consideration: the ease with which a method could be behaviorally evaluated. There are four that present difficulties against this criterion. These involve groups of people having specialized knowledge of the system being evaluated and representing special interest groups. Structured Walkthroughs typically involve presentations by the developers of code elements to their peers. Formal Customer Reviews involve presentations by system developers to a group who have commissioned the system. Peer Code Checking essentially involves system developers swapping their work for examination. Finally, Informed Panel Inspection usually involves a senior group of quality-conscious developers with special knowledge of either the potential implementation problems with systems of this type or the difficulties of achieving the application functionality. The main problem with using such group approaches in an experiment is that the experienced people needed for such approaches simply would not be available, particularly to staff the multiple experimental and control conditions which are needed for good experimental design.

There remains four methods: knowledge (semantic) checking, syntax checking, desk checking, and requirements tracing. All could be evaluated relatively easily, given the availability of the automated tools for the first two. All four methods appear to be generally desirable for inclusion as part of a certification procedure.

¹ The set of V&V methods at the time of this ranking consisted of 136 techniques. Subsequent to this activity several new V&V techniques were identified, and the database of methods reported on in Section 2.0 of this report, and in Volume 2, now consists of 153 techniques. Thus, the ranking of methods in Table 3.1-1 and Table 2.3.2-3 are different. However, by applying the same selection criteria of this section the same set of four methods would still have been selected.

Table 3.1-1 Rank-ordering of the top 10 rated static-testing techniques for the cost-benefit metric and the extreme effectiveness metrics.¹

Static Testing Techniques	Cost-Benefit Metric Ranking	Effectiveness Metric Ranking		Ranking Overall 3 Metrics ²
		Class 3	Class 1	
Syntax Checking	2	2	2	1
Semantic Checking	1	10	1	2
Structured Walkthroughs	36	1	10	3
Requirements Tracing	6	5	6	4
User Interface Inspection	6	9	7.5	5
Formal Customer Review	6	3	16	6.5
Peer Code Checking	6	4	15	6.5
Data Interface Inspection	9.5	8	14	8
Informed Panel Inspection	9.5	6	18.5	9
Desk Checking	6	7	22	10
Clean Room	17	16	3	11
Process Trigger/Timing Analysis	21	30	4	12
Failure Mode Effects Causality Analysis	26	21	9	13
Program Proving	31	42	5	14
Trace Assertion	40	48	7.5	15

1. This was the ranking used to select the methods. However, a different method of ranking techniques was developed for the final report, and revised in the Volume 2 survey of V&V techniques.

2. Based on the sums of rankings for the three metrics, ordered from lowest to highest.

Both knowledge checking and syntax checking address the correctness and accuracy of the knowledge base and would also appear to be highly desirable for a certification procedure. Checking of constraints and "meanings" will reveal invalid knowledge. Checking for violations of rule-writing style within a rule or testing for anomalies between rules (such as circularity or contradiction) identifies accuracy problems involved with the composition and interaction of knowledge elements. Both methods would contribute greatly to the certification procedure.

Desk checking, the careful human examination of a program for problems, is also a powerful and desirable technique, particularly when guided by precepts of what to look for and how. In particular, the exercise of mentally executing the code in this method consistently has been shown to result in the detection of post-compilation faults (e.g., checking for structural anomalies, good programming violations, incomplete cases).

Requirements tracing constitutes tracing forward from each unique requirement element to, where possible, design elements and thence to the elements of the implementation. Requirements tracing is truly a mandatory procedure to investigate the completeness of the knowledge base with respect to the requirements that were established at the start of the project. It will help determine whether the individual requirements and design elements are satisfied (testing of other components and the system as a whole is also necessary for this determination). It will also expose "unintended functions", aspects of the system which may have been added in for various good (or not so good) reasons but which are nowhere motivated by a requirements or design element.

Based on the above arguments, these four methods were chosen for experimental test.

3.2 Experimental Design

All conclusions and findings on the KB Certification experiment are based on Section 4.0 of Volume 4. As discussed in Section 3.1, the KB certification experiment involved the application of the following four different methods for static testing of sections of System A and B: requirements tracing, desk checking, knowledge (semantic) checking, and syntax checking. The experiment, conducted over a three day period, used a group of 20 participants drawn from three nuclear power plant utilities, a university graduate school, the USNRC, a national laboratory, and SAIC. Table 3.2-1 lists the original 21 participants (one extra member in case of any "no shows") with their key qualifications and group assignment. This group of 20 was divided into four groups of five, each with an equivalent experience background. These four groups were necessary so that a different group worked on System A and System B and, within each system, one group was the control using desk checking, while the other used experimental new methods.

3.2.1 The Four Methods

3.2.1.1 Requirements Tracing Method

The requirements tracing method selected for this experiment is based on an existing tool called SuperTrace. SuperTrace is written in SuperCard and runs on an Apple Macintosh II. SuperTrace provides a number of automated and interactive utilities which facilitate requirements tracing. This tool was developed by SAIC for the U.S. Navy, and was used with permission of the Navy. To facilitate the entry of requirements into SuperTrace, the SuperTrace Front End Processor (SFEP) was utilized. SFEP is written in TURBO Pascal and C++ and runs on an IBM PC or compatible. SFEP marks and extracts requirements from a document by searching for key words and phrases such as "shall" and "must". It writes the extracted requirements to a file which is then imported into SuperTrace.

Table 3.2-1 Experiment participant key qualifications and group assignment

ID	Group	Organization	Highest Degree and Field	Yrs Exp.	Nuclear Experience	Test Exp.	ES Exp.
27	Copper	SAIC	B.S.; Nuclear Engineering	2	PWR, BWR, EOPs	No	No
73	Copper	EG&G Idaho	M.S.; Electrical Engineering	19	PWR, BWR, Design	Yes	Yes
44	Iron	Duke Power Co.	B.S.; Electrical Engineering	9	PWR, Maintenance	No	Yes
99	Silver	SAIC	B.S.; Psychology/Statistics	2	BWR, EOPs	No	No
97	Gold	SAIC	B.S.; Nuclear Engineering	3	BWR, EOPs, Design	No	Yes
39	Silver	University of Md.	M.S.; Electro Optics	2	PWR	Yes	Yes
82	Copper	SAIC	M.S.; Engineering Physics	2	PWR, BWR	Yes	Yes
23	Silver	USNRC	B.S.; Nuclear Technology	23	PWR, EOPs, Operations, Maintenance	No	No
56	Iron	SAIC	B.S.; Nuclear Engineering	2	PWR, BWR, EOPs, Design	Yes	Yes
46	Silver	SAIC	B.S.; Electrical Engineering	3	--	Yes	No
51	Copper	Baltimore Gas and Electric	B.S.; Nuclear Science	10	PWR, EOPs, Operations	No	No
35	Copper	Duke Power Co.	B.S.; Chemical Engineering	2	PWR, EOPs, Operations	No	No
38	Gold	SAIC	M.S.; Mech. Engineering	2	PWR, BWR, EOPs	No	No
43	Gold	Yankee Atomic	M.S.; Nuclear Engineering	12	PWR, EOPs	Yes	Yes
25	Iron	Baltimore Gas and Electric	M.S.; Nuclear Engineering	12	PWR, Operations, Maintenance	Yes	Yes
65	Gold	University of Md.	M.S.; Nuclear Engineering	7	PWR, EOPs, Operations, Maintenance	Yes	Yes
54	Gold	Baltimore Gas & Electric	Shift Sup; SRO	20	PWR, EOPs, Operations	No	No
62	Silver	Yankee Atomic	B.S.; Mech. Engineering	22	PWR, BWR, EOPs, Design	No	No
75	Silver	Baltimore Gas & Electric	M.S.; Computer Science	14	PWR, EOPs, Operations, Maintenance	Yes	Yes
67	Iron	USNRC	B.S.; Mech. Engineering	11	PWR, EOPs, Operations	No	No
88	Iron	SAIC	B.A. Mathematics	4	--	Yes	No

Shift Sup = Nuclear Power Plant Shift Supervisor

BWR = Boiling Water Reactor

Nucl = Nuclear

EOPs = Emergency Operating Procedures

PWR = Pressurized Water Reactor

SRO = Senior Reactor Operator

The requirements tracing portion of the experiment was broken into two portions: requirements grouping and requirements tracing. Requirements grouping refers to the identification of requirements which belong to a common group or category. An example would be identifying all the requirements pertaining to "software" (or hardware, or knowledge). SuperTrace was used to group requirements based on various topics (e.g., "logic", "Heat Sink Control" (System B), and "RPV Flooding" (System A)). In SFEP, the user can specify categories for requirements (such as software, hardware, or hardware and software). SFEP will then assign these keywords or categories to the requirements as they are extracted (a synonym list can be specified also; for example, if the words "code", "source code", or "software" are found, assign the category "software"). Once the requirements and associated keywords are imported into SuperTrace, printouts can be made based on these categories. These printouts were used by the experimental group. The control group manually grouped the requirements, reading the text of each requirement and deciding whether or not it was part of the category.

Requirements tracing involved the requirements, the design, and the knowledge base of each of the two systems. For both systems, the Functional Requirements Document was the source of the Requirements. In the case of System A, the Emergency Operating Procedure messages (EOP messages) constituted the design information. For System B, the graphs (or trees) supplied the design information. For both systems, the rules (knowledge base) constituted the actual system (lowest level of the trace).

SuperTrace was used to trace the requirements to the design and the design to the rules for both systems. As a result of this process, a printout was made that contained the requirement text and information on the design and rules that satisfied the requirement for every requirement in both systems. These printout maps were broken into several parts: information pertaining to the requirement (e.g., ID, text, and section number); information on the traceability of the requirement (design, rules); the actual design data (EOP messages or graphs) pertaining to the requirement; and the appropriate rules for the requirement. The experimental group used these "maps". The control group manually traced the requirements, using keywords and phrases in the requirement text to find the appropriate design information and rules.

3.2.1.2 Syntax-Checking Method

The syntax-checking tool VERITE' was made available for experimental use by its developer, Dr. Alun Preece of Concordia University. VERITE' is a recent offshoot of a previous tool, COVER Version 3, based on Alun Preece's dissertation and tested in connection with commercial utility expert systems (cf. Preece, 1990). VERITE' is based on first-order logic and is designed to take as input the facts and rules of a knowledge base and produce as output a description of any **anomalies** that were detected in the knowledge base.

Examination of the rule knowledge bases by VERITE' required writing special rule-parsing front-ends for both expert systems which permitted the tool to identify and interpret all of the IF and THEN clauses which made up each rule. This is something that would normally be required the first time any automated tool was applied to a new rule syntax.

Once all rules are pre-processed into recognizable formats, VERITE' searches for the following kinds of anomalies:

- Conflicting (contradictory) rules
- Redundant rules
- Rules which are subsumed by other rules
- Cyclic inference chains (e.g., Rule A Causes Rule B to fire, which causes Rule A to fire)
- Useless rules (rules which can never be fired)

- Dead-end rules (rules with THEN Actions which never facilitate any other rule)
- Unsatisfiable conditions (a test for a variable-value which could never occur)
- Missing values (values of a variable that could be assigned, logically, but never actually occur in any rule)
- Missing rules (the situation in which some rule is needed to satisfy a requirement of the system, but no existing rules fulfill that function)
- Fundamental syntax errors (e.g., missing definition and clashes between variable types)

The output of VERITE' consisted of error reports for every anomaly or problem detected. The report provided a short description of the problem found and the identifiers of the rule(s) involved in the problem. When only one rule was thought by VERITE' to be causing the problem, the specific IF or THEN clause was identified with an arrow pointing to the culprit clause. A complete listing was provided of the complete rules involved in the problem.

For 11 of the 21 rule errors purposefully introduced into System A, VERITE' was able to identify either the single rule containing an error or a set of rules which included the error-containing one. No detection was made of the remaining 10 errors. For System B, within which there was no possibility of rule-chaining, VERITE' identified all 11 purposefully introduced rule errors. The empirical question is whether the experimental participants can utilize the VERITE' output effectively, and whether their performance is improved over participants using the desk-checking method.

3.2.1.3 Knowledge-Checking Method

The primary knowledge checking capability that appeared most promising in the review of expert systems V&V is an aspect of the experimental DARPA Expert Systems Validation Associate ((D)EVA) tool developed by Stachowitz and others (1987). However, this tool was not available for use in this experiment. The knowledge checking capability of the tool was simulated and augmented, based on an encoding of specific, readily-obtained, engineering knowledge about the two expert systems into a database, and assuming a very straight forward automated capability for accessing this database in the evaluation of each of the systems' rules. A description of the tool follows.

The simulated knowledge-checking tool introduced in the experiment was Meta-Check, a tool which checked the rules against the "meta-knowledge" about the nuclear power plant (operational and design engineering knowledge). The output of the tool was presented in a form analogous to that of VERITE'. An overall diagnostic message was given, the specific rule was listed, and the problematic IF clause(s) and/or THEN action(s) were indicated by an arrow.

Two types of information were encoded by the tool. First, all of the plant variables used in any way in the expert systems (e.g., RPV pressure, PC pressure) were described in terms of the sub-systems they were associated with, the units of their measurement, their allowable range of values, and the reference set points or values to which they could legitimately be compared. Descriptions and units were also provided for these references. Second, a list of all the legitimate comparisons between variables and references were developed. These comparisons included the actual operators which the expert system designer had decided were appropriate (e.g., expressing the notion of a variable exceeding a reference by the text string "greater than" rather than using the equivalent symbol ">"). In addition, the implication of every particular test or comparison being TRUE (e.g., "ALL RODS ARE INSERTED AT OR BEYOND 06") was evaluated in terms of the impending consequences of that state, using the following set of descriptors: Normal, Improving, Degrading, Serious, and Catastrophic.

A major difference between the capability and output of VERITE' vs. Meta-Check is the latter's capability to fault-locate to a specific clause in one rule, a problem that VERITE' would recognize only as a multiple-rule symptom. Given the engineering database, Meta-Check would be able to recognize 18 of 21 seeded errors for System A, compared to 11 for VERITE' for matched errors; for System B, it would recognize all 11, as would VERITE', but would also be able to locate the offending clause(s) in specific rules. In comparison with D(EVA), Meta-Check was judged, generally, to be more powerful because of the capability to utilize additional types of information.

3.2.1.4 Desk-Checking

This was the control condition in that no (simulated) tool was provided to support the mental examination of the Kbs, and yet this is an effective technique when accompanied by specific training on the specific fault targets, as was provided here.

3.2.2 Faults

Three sets of faults were developed for inserting into the three knowledge base partitions. These were derived on the basis of two principles: (1) the location of the error within a rule, and (2) the expected difficulty of detecting the error. This scheme was used only partially for System B, because of the simplicity of these rules and the absence of rule-chaining; it was fully employed for generating the System A errors.

Ten different error locations were specified, six involving the IF clauses and four involving the THEN actions. The six IF-clause locations for errors were: (1) in the variable, or left-hand-side, of a full IF clause (e.g., in "RPV PRESSURE" for the clause "RPV PRESSURE > PC PRESSURE"); (2) in the comparison operator for a full IF clause (e.g., ">"); (3) in the reference, or right-hand-side, of the negotiation of a full IF clause (e.g., "PC PRESSURE" in the above rule); (4) in the negotiation of a reduced (Boolean) IF clause; (5) in the rule presence of an additional IF clause added to the rule; and (6) in the absence of an additional IF clause deleted from a rule. The four THEN-action locations for errors were: (1) in the keyword of a THEN action (e.g., "SET"); (2) in some other part of a THEN action; (3) in the presence of an additional THEN action added to the rule; and (4) in the absence of a THEN action deleted from a rule.

One error was designed for each of the four deleted or added IF-clause and THEN-action locations, and the remaining errors were distributed roughly equally among the remaining six possibilities.

Seven levels of expected detectability ranging from "should be very obvious to anyone" to "would require a nuclear engineer intimately familiar with this expert system" were used. Three to four errors were chosen for each level of detectability. A total of 20 to 21 errors (20 errors seeded into System A for syntax checking, 21 errors for the knowledge checking method) were seeded into the System A knowledge base partitions for the syntax checking and knowledge checking methods, respectively. A total of 11 errors were seeded into each of the System B syntax-checking and knowledge checking methods. The seeded faults, along with their location and level of expected detectability, can be found in Appendix A of Volume 4 of this report. To simplify analysis of the participant responses, the seven levels of detectability were condensed into three categories: hard, medium, and easy, with hard being the most difficult to detect, and easy being the simplest to detect.

3.2.3 Training

During the course of the experiment, considerable time was spent providing the participants with both general and specific training. This training included the topics of: AI, V&V, System A and B, requirements tracing, syntax checking, and knowledge checking. Sometimes all the participants received the same instruction while, at other times, the groups were

divided according to their assigned system or whether they were in the control or experimental groups. The same basic training on the overall methods, AI, V&V, and error detection and classification was provided to all participants. Differences were introduced for the experimental vs. control groups and the information about the specific AI system (i.e., A or B) that were only imparted to the appropriate group(s). Each member of the experiment was given training reference materials along with worksheets to record details about the errors that they detected such as time to find, type of error, and confidence that the error is real. Each V&V method was allotted three hours for the participants to detect the seeded errors.

3.2.4 Hypotheses Under Test

The primary experimental hypothesis is that relative performance in the tool-aided conditions would be superior to the control (desk-checking) condition. Secondary hypotheses were 1) that performance with the Meta-Check tool would be superior to that with Verite because the former directly identifies the nature of the fault while the latter identifies fault symptoms, and 2) that nuclear experience per se should not be associated with superior performance, since the information provided by the tools is sufficiently precise that engineering knowledge should contribute little additional value.

3.3 Experimental Results

All conclusions and finding on the KB certification experimental results are based on Sections 5.0 and 6.0 of Volume 4. This experiment involved the manipulation of three principal parameters: the expert system being examined (i.e., System A or B), whether the group used control methods or experimental methods, and the particular V&V method itself (i.e., requirements tracing vs. syntax checking vs. knowledge checking). The analysis of results refers to these three parameters as system, condition, and method, respectively. The data for the experiment consisted of worksheets which were filled out by the participants. This data was analyzed by the analysis-of-variance (ANOVA) model for 24 dependent variables which included such parameters as: number of correctly found problems (i.e., SAIC seeded errors), total time for exercise, total number of false alarms (a false alarm is a participant's response which is not truly a problem or error), confidence for correct problems, and average time for problems identified.

Results of the ANOVA analysis were evaluated in terms of the probability of a result being due to chance. If this probability of the result being due to chance was less than 0.05, the value is considered to be statistically significant. One of the most important results from this experiment was that the experimental group outperformed the control group by a significant margin in every parameter. Specifically, the experimental group found twice as many correct problems, missed half as many problems, had half as many false alarms, and finished in 18% less time than the control group. This analysis shows that the tools are a powerful means of improving Knowledge-base (KB) certification.

In comparing the experiment results between Systems A and B, the System B group outperformed the System A group in all areas. This was expected because the System B KB is less complex than that in System A. The importance of this conclusion is not that more complex systems are harder to certify, but rather that the other effects such as the value of experimental methods are not masked by the complexity of a system.

Between methods, the most important result was that the knowledge checking tool was more effective than the syntax checking tool in such areas as number and percentage of correctly found problems, fewer false alarms, and time to complete the exercise. In fact, knowledge checking as a V&V method proved to be a very powerful technique for identifying errors in the KB.

An evaluation of requirements tracing revealed that the experimental group with tools outperformed the control group without tools. There was no conclusive difference in requirements tracing between the two systems. In the case of

syntax checking, again, the experimental group clearly outperformed the control group, but, in addition, the System B participants performed significantly better than those assigned to System A. For knowledge checking, the better performance was determined for the experimental group and System B (with and without tool as compared to System A) participants. Some representative significant results of this experiment are graphically presented in Figure 3.3-1.

The effects of the participants' backgrounds and experience on their results is of interest. Particularly, how did the tool assist participants with a large amount of testing experience but not much nuclear engineering experience? To answer this question, the participants' results were subsequently analyzed using student t-tests. Three sets of tests were run. The first set of t-tests examined a group of four subjects who all had testing experience, but little or no nuclear engineering experience. All four participants came from the experimental (tool) condition. They were compared to the remaining members of the experimental group (6 participants). Next, two members of the experimental group who had testing experience, but also had nuclear engineering experience were added to the group, increasing it from 4 to 6. These 6 participants' scores were compared to the remaining 4 members of the experimental group. Finally, the 10 participants with the least amount of nuclear engineering experience were compared to the 10 having the most experience. This was based on years of nuclear engineering experience. The "low experience" group had between 0 and 4 years of nuclear experience for an average of 2.4 years of experience. The "high experience" group had between 7 and 22 years of experience for an average of 13.6 years of experience. In view of the small number of participants involved, all of the following results should be viewed with caution.

The results of an analysis of participant experience have important implications for V&V. The participants with testing experience but no nuclear experience did not outperform the remainder of the experimental group. The participants with testing experience and some nuclear experience did not outperform the remainder of the experimental group. The participants with 7-22 years of nuclear experience did not outperform the participants with only 0-4 years of nuclear experience. Thus, the ability to certify these KBs was not enhanced or affected by the amount of testing and/or subject matter experience of the participants. Only in the cases of the very difficult errors would nuclear experience be expected to make an important difference (see discussion below.) The implication of this result is that junior level people can be used to effectively V&V knowledge bases rather than more expensive senior level people with no loss in performance. A possible exception to this is in the case of detecting hard errors.

Not all the errors seeded by the experimenters were detected. As one would expect, the undetected errors were in the more complex of the two systems, System A, and were for methods 3 and 4. These errors were categorized as "hard to detect" problems. For method 3, syntax checking, three errors went unnoticed. The first error involved substituting "RPV Water Level" for "S/Pool Water Level". This error, though not detected by VERITE', would have been detected by Meta-Check. There were no obvious clues from the dimensions or wording of the variable or reference, but a nuclear engineer very familiar with BWRs could detect this error since "RPV Water Level" is never compared to "suppression pool load limit". Similarly, the second error could have been caught by a highly experienced nuclear engineer. There were no clues to this error in previous rules, and it would have been discovered by Meta-Check. This second error substituted "NPSH Limit" (Net Positive Suction Head) for "TAF" (Top of Active Fuel). However, the third error, which involved the removal of an IF clause "RCIC is not running", would not have been caught by Meta-Check and would require an expert on EOPs to detect.

For method 4, knowledge checking, no errors went undetected. However, two of the hard problems found were not correctly categorized (cause ID) by any of the participants. That is, the participants wrote down the correct rule ID, but wrote down a cause ID that was unequivocally wrong. This shows that the participants didn't really catch these hard errors. The first error involved deleting the second THEN clause "set caution 6". This would not have been detected by Meta-Check. There were no obvious clues from the dimensions or wording of the variable or reference. Only an expert on EOPs could catch this error. The same holds true for the second error which deleted the 3rd IF clause "H2 Recombiner Max Operating Limit Has Been Exceeded." Hundreds of dynamic cases may have to be developed and run in order to find these

errors. So it seems practical that instead, a new static tool or method should be devised to detect these types of errors. Even given an unlimited amount of time to analyze the rules, it is highly unlikely that these types of errors would have been detected without such a new tool.²

The following conclusions are drawn from these results (based on Sections 5.0 and 6.0 of Volume 4):

- 1) The use of tools or aids for use in static knowledge base certification results in a significant improvement in detecting all types of faults (including hard-to-find problems), avoiding false alarms and completing this effort in less time.
- 2) Of the four tools developed and used in this KB certification experiment, knowledge checking and requirements grouping showed the greatest improvement in performance.
- 3) Testing experience and extensive (7 to 22 years) subject matter experience did not improve the performance of the KB certification participants as compared with less (2 to 4 years) subject matter experience. (It should be noted, however, that 95% of the participants had a college degree in a technical field and the remaining 5% was highly trained in a technical field). Were this finding to be verified in an experiment with a larger number of participants, this discovery could be capitalized on by the nuclear industry to lower the costs of performing V&V on expert systems.
- 4) Even with the tools and aids provided, some hard "seeded" errors were still not detected by anyone. Other methods or tools may be warranted for KB certification.
- 5) In addition to the errors described in item 4, a very small percentage of hard "seeded" errors would not have been caught by existing tools and would require an expert to detect.

On the basis of the present findings, the recommendation is made that tools comparable in function to those tested here be strongly considered to be among those ultimately selected for certification of expert systems knowledge bases.

² Another way these errors might be detected is by comparing the rules to the original hard-copy EOPs. In the case of this system A, the rule-base and knowledge source are highly similar in structure, with the hard copy EOPs being very much in a logic-like format. However, in general, this will definitely not be the case, and such a comparison would not be feasible.

Figure 3.3-1 Representative Significant Results for System and Condition

Figure 3.3-1 (Continued).

Figure 3.3-1

4 VERIFICATION AND VALIDATION GUIDELINES FOR INTELLIGENCE SYSTEMS

The guidelines presented here are based on work performed for Activity 4. There is no supplemental volume describing this work.

4.1 Factors for Selecting Verification and Methods

4.1.1 Applications and Systems Covered

Applications

Expert systems and other Artificial Intelligence (AI) systems could be used for the entire spectrum of nuclear power industry software applications including such functions as design, operation, and maintenance. **Design functions** encompass such nuclear engineering and scientific disciplines as: heat transfer, fluid flow, reactor physics, materials engineering, radiochemistry, health physics, structural analysis, shielding, instrumentation and control, and system design. Artificial Intelligence systems could assist in any of these areas. **Operation** applications include emergency operating procedure identification, alarm signal processing, refueling outage management, and real-time monitoring of safety and/or economic plant performance parameters. Model-based and qualitative reasoning AI systems are well suited for such operational assistance.

AI systems can play a greater role in determining the condition of plant components and the needed **maintenance** activities. They are ideally suited for diagnosing precursors of equipment fault by assessing the meaning of changes in measured performance parameters and can assist in determining the impact on safety of proposed changes to plant hardware or processes. The present guidelines could apply to all these and all potential nuclear applications, since nothing in the guidelines is dependent on the nature of the specific application. Only the complexity of the system and anticipated need for its reliability and integrity are guideline considerations. For this reason, the present guidelines could apply equally well to any AI system or conventional software application, within the nuclear industry or without.

Systems

The guidelines cover a variety of AI software approaches as discussed below. The guidelines can also be applied to conventional non-AI systems. This is because many of the components of AI systems are implemented in conventional procedural programming languages so there is no real distinction. As discussed in earlier sections, the knowledge base is the only component that requires other than conventional V&V techniques. AI systems covered by these guidelines include: 1) Rule-Based Systems - AI systems containing knowledge represented in the form of IF-THEN rules; 2) Frame-Based Systems - AI systems containing knowledge represented in the form of a structured collection of knowledge-attributes and their values called frames; 3) Model-Based Systems, Case-Based Reasoning Systems, or Qualitative Causal Reasoning Systems - AI systems with rules, frames, or other types of declarative representation knowledge bases organized into models or examples.

Object-Oriented Programming Systems (OOPS) are also addressed by these guidelines. The five components of OOPS are: 1) objects; 2) messages and methods associated with objects; 3) classes of objects; 4) external interfaces; and 5) tools and utilities. Discussed at length in Section 7, these components have much in common with the components of expert systems. Neural Nets and Genetic Algorithms are two other important AI systems. Neural nets use biologic neurons as an architectural metaphor for classification and control problems, while genetic algorithms use the metaphor of genetics and are often used for optimization problems. Both are generally developed in conventional procedural programming languages, so the guidelines would also apply to these systems to the extent that their design or overall performance is to be

evaluated against their initial requirements. However, these guidelines do not specifically provide for testing the data structures and subprocesses unique to these types of systems.

System Aspects to be Tested

The total set of system components including the overall system itself can be aggregated into three classes, with a different V&V approach appropriate for each. This classification is shown in Table 4.1.1-1. The Type 1 component comprises the various forms of declarative knowledge structures in AI systems. These typically encode the vast majority of the application-specific information necessary to achieve the system's required functioning. The Type 2 component includes all aspects of the final system and all associated tools and utilities used with the system or in its development which are very generic and highly reusable. The Type 3 component includes everything else including the total overall system.

4.1.2 V&V Methods and the Developmental Life-Cycle

Activity 1 identified 153 different types of methods for conventional systems and these methods (or their extensions) provide the basis for the recommended conventional techniques. Activity 2 surveyed the existing V&V methods for AI systems and provided only four new techniques. Activity 3 provided first-hand experience of the application of conventional methods to AI systems, particularly requirements tracing. Activity 6 evaluated the actual behavioral effectiveness of two conventional techniques and of two of the new AI system techniques. These sources all contributed to the V&V methods recommended in the guidelines.

Methods have been classified as belonging to one or more of three phases of a typical sequential developmental life-cycle of a software system, specifically Requirements, Design, and Implementation phases (no methods were found specifically designed for the Maintenance phase). Though no methods were found specifically designed for the Maintenance phase, all methods presented will be useful for that phase. Maintenance involves the re-visiting of all the phases of the Life-cycle, so therefore these guidelines apply to the V&V of already developed as well as evolving systems. The Implementation phase has two main subclasses of methods: Dynamic testing - involves actual execution of the code on an appropriate platform and operating system; and Static testing - involves various types of analyses either manual or automated, but no code-execution. The recommendations for specific methods are organized in terms of these three life-cycle phases.

4.1.3 Management Aspects of V&V

The quality of V&V that can be obtained is influenced, directly or indirectly, by several software management topics such as management of the software Life-cycle, development of a software system to meet industry standards, development of software using standardized software engineering practices, and management of the quality assurance activities for the software. A brief discussion of each topic follows.

Management of the Software Development Life-Cycle

The most opportune time to find errors and inexpensively correct them is during the crucial early phases of the Life-cycle. Tavolato and Vincena [1984] found that 56% of all bugs detected during testing could be traced to errors made in the requirements phase. There is a variety of evidence which shows that the relative cost of fixing an error increases drastically as a function of the Life-cycle phase in which it is detected. For example, Boehm found that errors not found until acceptance testing cost 48-100 times more to correct than errors found during requirements specification (Boehm, 1976). Therefore it is important to "front-load" the V&V activity. The cost savings can more than justify the expenditure for the V&V itself.

Standards

Eight key standards and guideline documents are currently the primary source of V&V guidance in the nuclear industry and are discussed at length in Volume Two (Section 2.3). These standards discuss a range of topics including

**Table 4.1.1-1 Three types of AI, and conventional, system components
and the recommended V&V philosophy for each**

Types of System Components	Recommended V&V Philosophy		
	Formal Verification	Certification	Traditional System V&V
Type 1 Knowledge Structures: <ul style="list-style-type: none"> • Rule Bases • Frames • Class Hierarchies • Objects Stored in Databases 	X		
Type 2 Highly Reusable Components: <ul style="list-style-type: none"> • Tools & Utilities • Some Interfaces • Some Functions • All Commercial Software Packages used 		X	
Type 3 Mostly Non-reusable Components: <ul style="list-style-type: none"> • Overall System • Most Interfaces • Conventional Databases • All Remaining Application Code 			X

software quality assurance requirements for thermal-hydraulic safety analysis software, mapping of recommended software quality assurance practices against the criteria for a complete nuclear quality assurance program, etc. These standards are most important sources for information on how and why to establish a software development Life-cycle, the software quality assurance practices, conventions, and procedures that should be followed at each step, the documentation to be produced/revised at each step, criteria for testing and validating nuclear power software applications, and testing techniques. In addition to these eight standards, there are an additional 66 standards which are relevant in one way or another to quality assurance, V&V, and software development practices in general.

Software Engineering Practices

Standardized, proven software engineering practices should be used in developing software. For example, the use of structured walkthroughs to review source code prior to commencing unit testing of that code is an established practice that should be considered. The software engineering practices that are planned for use should be documented in the Software Development Plan and then this plan must be adhered to. The continuous monitoring of the software development process is just as important as setting up the initial strategy for what software engineering practices to use. Specifically, areas which can be improved should be identified and then be aggressively pursued.

Quality Assurance and Configuration Management

Quality Assurance (QA) ensures that the software product has undergone all of the specified procedures that accompany the design and implementation. V&V is only one aspect of QA, which includes other elements such as configuration management, QA reviews and audits, etc. Configuration management (CM) controls the manner in which a system's components are modified and improved and integrated. The four major CM activities are configuration identification, configuration change control, configuration status accounting and reporting, and configuration audits and reviews. Excellent guidance for CM can be found in ANSI/IEEE Standard 1042-1987.

4.1.4 Risk Management Approach

Two types of risks can be identified when developing a system: development risks and fault risks. Development risks are those associated with developing the system and insuring that it is deployed and used. Fault risks are associated with a fault or mis-operation of a deployed system. Development risks are not covered by the guidelines, though they are avoidable in great part by paying close attention to the management issues discussed in the previous section. Fault risks will be considered further.

4.1.4.1 Complexity, Faults, Failures, Risk, and Required Integrity

It is postulated that system faults and their potential severity are a monotonically increasing function of system complexity. As the complexity of a system increases so, it is proposed, does the probability that faults will exist.

A fault must exist in the code for a system failure to occur. A fault is defined as an error of some kind which could lead to a failure of the system. While a failure always entails a fault in the code, the reverse is not true. Faults may be detected by a variety of static testing means.

4.1.4.2 System Faults as a Function of Life-cycle Phase

There are a wide variety of possible faults for each Life-cycle phase. Faults increase with system complexity and the desirability of removing them increases with degree of required integrity. The manner in which they should be prevented and/or removed will be determined by the specific nature of the faults. It is important, therefore, to understand the types of faults which can occur and how to prevent/remove them.

The faults for the requirements stage are organized into nine main types: incompleteness, incorrectness, lack of clarity, lack of explicitness, lack of consistency, lack of testability, not separately identifiable, inadequate human-computer

interface, and inadequate operational concept. Table 4.1.4.2-1 (on page 80) lists and describes these failure. Subtypes of the first seven requirements failure are presented in Table 4.1.4.2-2.

Eight of the nine main requirements failure also carry over into the design phase. In addition there are seven new types of faults which are shown in Table 4.1.4.2-3. The subtypes of the design faults are shown in Table 4.1.4.2-4. The implementation faults are described in Table 4.1.4.2-5. Five of the nine main types of fault carry over with nine new types for a total of eighteen types. The subtypes of the implementation faults are shown in Table 4.1.4.2-6.

4.1.4.3 Three Classes of V&V

The three classes of V&V defined in Section 2.1 are critical to the proper development of guidelines.

4.1.4.4 Quality Assurances for Each V&V Class

The quality assurances provided for the three classes of V&V are identified in terms of positive assurances that the particular faults as detailed in Tables 4.1.4.2-1 and 4.1.4.2-2 are absent. For example, completeness becomes the assurance that there are no incompleteness faults. The identification of the specific quality assurances provided by specific levels of V&V defined for well-characterized systems is a major, novel contribution of these guidelines.

The assurances provided for each class of V&V are shown separately for the three Life-cycle phases of requirements, design, and implementation in the second column of Tables 4.1.4.4-1 through 4.1.4.4-3. The third column identifies the most appropriate V&V methods to ensure that the assurances are achieved.

4.2 Guidelines

The "roadmap" for accomplishing V&V commences with the decision to perform V&V on a system. As discussed earlier, this decision is most effective when made at the very outset of a project. The user must then determine the developmental phase the system is in (Section 4.1.2) and what aspect of the system is to be tested (Section 4.1.1). Next, it must be determined how much V&V is appropriate for the system. A three-part questionnaire, shown in Figure 4.2-1, is to be completed to determine the appropriate class of V&V: first the system control/complexity is assessed; second, the Required Integrity is assessed; finally, the Class of V&V is determined from these results.

4.2.1 Guideline Packages

The results of the above three decisions will permit one to select an appropriate "guideline package" (a set of V&V procedures to be applied to a particular aspect of the system at a particular developmental phase). The matrix used to determine which guideline package should be used is given in Table 4.2.1-1. Systems in the requirements phase should use part (a) of the Table; those in the design phase should use part (b); and those in implementation should use part (c) of the table. A single package for the maintenance phase has been provided also (Package P). A sample guideline package has been included as Table 4.2.1-2. The complete guideline packages are not reproduced here but can be found in Volume 7. Each package is in the form of two parts: a table listing all of the V&V methods that are recommended for the situation and a table (Guideline Plan) describing how one should apply these methods. In the first table (Table 4.2.1-2), the first column provides an ID number for the V&V method by which it will be referred to in the second table. The second column (Method) lists the names of the recommended V&V methods, and the third column (Type) classifies the V&V method as either Static or Dynamic. The fourth column (Assurances) lists the quality assurances afforded by the method, the fifth column (Agents) describes the recommended qualifications of the personnel applying the methods, and the sixth column (Intensity) describes the amount of effort required in applying the method (limited, moderate, or extended). Column seven (Criteria/Action) establishes the thresholds for the three possible outcomes: accept, conditional acceptance, and reject. A detailed discussion of the standard decision thresholds can be found in Volume Five. Column eight (Ref) provides a reference for the method. An entry beginning with an "R" can be found at the end of the first table. A "P" entry indicates that a Guideline Procedure can be found which describes the method. Measures describes the quantitative measures or judgments that have to be taken or made during application of the method. The second table describes how the methods would be applied (what order). Table 4.2.1-3 presents a sample guideline plan.

4.2.2 Guideline Procedures

The V&V guideline procedures provide information on the step-by-step execution of a specific V&V method included in a Guideline Package. A sample template for a guideline procedure is shown in Figure 4.2.2-1. Some guideline procedures are self-contained. Others are broken into several well-defined activities. The format of the lower level activities is identical to that of the highest level procedures. Throughout the procedures, a hollow bullet (o) refers to an optional step, whereas a typical filled bullet (●) refers to a mandatory step. The complete guideline procedures (eleven in total) can be found in Volume Five. The guideline procedures are broken into the following sections:

WHEN TO USE THIS GUIDELINE - This section contains the overall goal for the guideline procedure, such as "To perform desk checking on the system source code." The section also presents a description of the scenario in which this method would be used. For activities within a procedure, this section is titled "GOAL OF THIS ACTIVITY" and contains only the goal.

PRE-CONDITIONS/TRIGGER CONDITIONS - This section lists the "entry criteria" for a guideline procedure. The items listed here are prerequisites for performing the procedure, or are events which will cause a procedure to be undertaken.

PLANNING - Any necessary planning which must occur in order to perform the guideline procedure is described in this section.

PROGRAM MANAGER APPROVAL - Some guideline procedures represent such large undertakings that Program Management approval should be received prior to commencement.

SETUP - This section describes the necessary setup required to perform a method.

EXECUTION - For top level guideline procedures (such as P1.0, Requirements Tracing), a graphic may be used to depict the activities within a guideline procedure. Lower level activities will provide a textual description of the steps required to execute the method.

REFERENCES AND METRICS: Within the Execution section, bibliographic references will be indicated as RXX (e.g., R2). The full reference can be found in the endnotes. Similarly, a pressure gauge icon with an MXX identifier is used to depict special metrics of interest (e.g., M10). The actual metric name can be found in the end notes.

INTERRUPTS: It is possible that during execution of a particular step of the method, V&V must be halted. This is referred to as an interrupt. For top level guideline procedures, the interrupt is depicted as a dashed line pointing to the graphical interrupt box. In lower level activity procedures, the interrupt is noted by square brackets [].

ANALYSIS - This section describes the metrics that will be determined as a result of performing the method, as well as describes any other suggested analysis that could be performed. As in the Execution section, metrics and/or references may be found here.

ACCEPT/REJECT CRITERIA - For top level guideline procedures there are three possible results: ACCEPT, CONDITIONALLY ACCEPT, AND REJECT. The notion here is that in rare cases a V&V method will uncover only very few, minor problems (ACCEPT). Also, hopefully rarely, a method will detect a large quantity (or severity) of problems (REJECT). Most often, a method will detect problems which need to be resolved before continuing, but which do not constitute rejection (CONDITIONALLY ACCEPT). For lower level activities, there are only two possible results: ACCEPT and move on to the next activity, or REJECT (correct problems and repeat this activity).

REPORTING - This section describes any reports that are written or any reporting to the Program Management that occurs as a result of this V&V method.

PROGRAM MANAGER REVIEW - This is analogous to the Program Manager Review that occurred prior to the method being undertaken. This is the "debriefing".

OUTPUT/COMPLETION STATUS - Depending on the results of the procedure, the Accept/Reject criteria will result in a particular completion status. It is this completion status that determines the next course of action.

Table 4.1.4.2-1 Types of faults that can occur during the requirements stage¹

Type of Fault	Description
Incompleteness	Client's concerns/needs have not been covered completely, or are not exhausted. Terms and phrases in the requirements are not defined.
Incorrectness	Client's concerns/needs have not been accurately described in the specifications or the specifications disagree with known facts.
Lack of Clarity	Specifications are ambiguous or are not described in language which is familiar and understandable to all of the clients, users, and developers.
Lack of Explicitness	Necessary (or presumed or assumed or implicit or implied) requirements or requirement-aspects are not present as a clear part of some requirement specification.
Lack of Consistency	Some of the requirements logically conflict with other requirements.
Lack of Testability	The requirement has been stated in such a way that it will be virtually impossible to determine whether or not it has been satisfied.
Not Separately Identifiable	Specifications contain compound requirements.
Inadequate Human Computer Interface	The requirements do not adequately specify the human-computer interaction required of the system.
Inadequate Operational Concept	The requirements do not adequately specify the operational concept for the system, or else the specified concept violates standards, guidelines, or ease-of-use precepts.

¹This and the following five tables are replicated from Tables 6.2-1 through 6.2-6 in Volume 5.

TABLE 4.1.4.2-2. Decomposition of identified faults for the requirements stage

Major Fault	SubFaults
Incompleteness	<ul style="list-style-type: none"> • Incomplete sections • Terms not defined • Logic not fully articulated • Detailed requirement not fully decomposed and therefore ambiguous • System goals not fully specified • Needed function is missing • Abstract specification not adequately decomposed • Next lower level of abstraction of a higher level not adequately specified
Incorrectness	<ul style="list-style-type: none"> • Failure to specify initial system state when not equal to zero • Requirements do not express customers' needs <ul style="list-style-type: none"> -factual errors -incorrect references to Standards or other works -errors in logic • An incorrect (or missing) value or variable is specified • Requirements violate accepted principles of good engineering • Requirement over or under states the computing resources assigned to a specification • System input or output not fully described • Requirement does not accommodate the operational environment (e.g., data rates) • Requirement is not feasible given other system factors (e.g., memory available) • Requirements are excessive for the operational need
Lack of Clarity	<ul style="list-style-type: none"> • Ambiguities in scope of qualifiers • Ambiguities in meaning of words • Ambiguities in reference • Imprecise phrases • Use of passive where agent not clear • Fuzzy quantifiers
Lack of Explicitness	<ul style="list-style-type: none"> • Something is stylistically implied by the sentence construction but not explicitly stated • Something is logically implied but not stated
Lack of Consistency	<ul style="list-style-type: none"> • One statement directly contradicts another • A statement indirectly is inconsistent with another • A frame of reference, or context, in one part of the document is inconsistent with a frame of reference in another part • A statement is inconsistent with known states of affairs external to the requirements document • Requirements of cooperating systems (taken pairwise) are incompatible
Lack of Testability	<ul style="list-style-type: none"> • A goal condition is stated but no measurement is stated or implied for determining whether goal is met • A requirement is logically not testable

Table 4.1.4.2-2 Decomposition of Identified Faults for the Requirements Stage (Continued).

Major Fault	SubFaults
Not Separately Identifiable	<ul style="list-style-type: none"> • Compound sentences containing two or more requirements • Partial information about a single requirement distributed in several places • Requirements expressed as bullets, each completing the same introductory sentence
Inadequate Human Computer Interface (HCI)	<ul style="list-style-type: none"> • Menus and menu items are not consistently in the same locations on different displays • Novice users or expert users have not been considered in the HCI design • Displays, menus, reports are not adequately labelled • Required functionality is missing or incorrect
Inadequate Concept of Operations	<ul style="list-style-type: none"> • System causes extra steps to be taken to perform a previously manual task • Concept of Operations(CONOPS) not supported by the system or key steps have been omitted • CONOPS violates standards or guidelines • CONOPS violates ease-of-use precepts

**Table 4.1.4.2-3. Types of faults that can occur
during the design stage**

Type of Fault	Description
Incompleteness	Design elements do not implement all specified requirements.
Unfounded Functionality	Design elements exist which are not traceable to the requirements specification.
Incorrectness	Design element would not work, if implemented, and/or is not reasonable.
Lack of Clarity	Design specifications are ambiguous, or are not described in language which is familiar and understandable to all of the clients, users, and developers.
Lack of Explicitness	Necessary (or presumed or assumed or implicit or implied) design element or design-aspects are <u>not</u> present as a clear part of some design specification.
Lack of Consistency	Some of the design elements logically conflict with other design elements. Data flow conflict exists between interfacing design elements.
Inadequate Operational Concept	The requirements do not adequately specify the operational concept for the system.
Inadequate Human Computer Interface	The design does not adequately describe the human-computer interaction required of the system.
Inadequate System Architecture	Proposed system architecture is not sufficient to support the overall system mission.
Inadequate Function Partitioning	The design fails to logically partition functions between hardware, software, and human personnel.
Lack of Sufficient Database Design	Database design is not a logical, consistent, and/or normalized (if applicable) representation of the data to be stored/retrieved/ manipulated.
Safety Problem/Hazard Conditions	Design lacks robustness to handle potential failures, specifically safety problems and hazard conditions.
Lack of "Fail-Safe/Soft-Failure" Operations	Design lacks ways (safeguards) to degrade gracefully in the event of a failure.
Lack of Good Software Engineering Practices	Good software engineering practices, such as use of exception handlers, have not been applied in producing the design.

**Table 4.1.4.2-4. Decomposition of identified faults
for the design stage**

Major Fault	SubFaults
Incompleteness	<ul style="list-style-type: none"> • Requirement or specification not addressed by design • Processing priorities are not established or are not modifiable due to control structured design • Top-level design modifications are not reflected in the detail design or vice versa
Unfounded Functionality	<ul style="list-style-type: none"> • Design elements that cannot be traced to a requirement or specification have been included
Incorrectness	<ul style="list-style-type: none"> • Requirement not accurately represented in design • Full range of possible data not accommodated by design • Commercial off the shelf (COTS) hardware resources proposed are beyond those available (e.g., memory) • Computational errors such as round-off, truncation, etc. not handled • Design does not conform to standards • Boundary and/or unique conditions have not been handled
Lack of Clarity ¹	<ul style="list-style-type: none"> • Ambiguities in scope of qualifiers • Ambiguities in meaning of words • Ambiguities in reference • Imprecise phrases • Use of passive where agent not clear • Fuzzy quantifiers
Lack of Explicitness ¹	<ul style="list-style-type: none"> • Something is stylistically implied by the sentence construction but not explicitly stated • Something is logically implied but not stated

¹ The subFaults are identical to those of the Requirement Stage

Table 4.1.4.2-4 (Continued).

Major Fault	SubFaults
Lack of Consistency	<ul style="list-style-type: none"> • One statement directly contradicts another • A statement indirectly is inconsistent with another • A frame of reference, or context, in one part of the document is inconsistent with a frame of reference in another part • A statement is inconsistent with known states of affairs external to the design document • Requirements of cooperating systems (taken pairwise) are incompatible • Internally interfacing elements have data parameters/arguments which directly contradict each other • Internally interfacing elements have data parameters/arguments which indirectly contradict each other • Externally interfacing elements have data parameters/arguments which directly contradict each other • Externally interfacing elements have data parameters/arguments which indirectly contradict each other
Inadequate Operational Concept ¹	<ul style="list-style-type: none"> • System causes extra steps to be taken to perform a previously manual task • Operational concept not supported by the system or key steps have been omitted
Inadequate Human Computer Interface	<ul style="list-style-type: none"> • Human Computer Interface design would be difficult to learn/user, see/read, etc. • Menus and menu items are not designed to be consistently in the same locations on different displays • Novice users or expert users have not been considered in the HCI design • Displays, menus, reports are not labelled
Inadequate System Architecture	<ul style="list-style-type: none"> • System architecture not sufficient for overall system mission (under-designed) • System architecture is excessive for the operational need (over-designed) • System architecture lacks robustness
Inadequate Function Partitioning	<ul style="list-style-type: none"> • Design elements which should be allocated to hardware have been wrongly allocated to software or human personnel • Design elements which should be allocated to software have been wrongly allocated to hardware or human personnel • Design elements which should be allocated to human personnel have been wrongly allocated to hardware or software

Table 4.1.4.2-4 (Continued).

Major Fault	SubFaults
Inadequate Knowledge Structure Design	<ul style="list-style-type: none"> • Knowledge structure is inappropriate for required functionality • Knowledge structure is insufficient to represent all needed knowledge • Knowledge structure design will cause performance problems
Inadequate Database Design	<ul style="list-style-type: none"> • Database design fails to represent all data required • Database design incorrectly represents data (e.g., format) • Database design represents extraneous data not required by the system • Database design not normalized (if relational)
Safety Problem/Hazard Conditions	<ul style="list-style-type: none"> • Design lacks ability to handle safety problems • Design lacks ability to handle hazard conditions • Design fails to account for failure modes • Design lacks exception handling features
Lack of "Fail-Safe/Soft-Failure" Operations	<ul style="list-style-type: none"> • Design lacks "fail-safe" operation • Design lacks "soft-failure" operation • Design does not provide a means of degrading gracefully in the event of a failure
Lack of Good Software Engineering Practices	<ul style="list-style-type: none"> • Design possesses weak modularity • Design possesses excessively rigid control structure

Table 4.1.4.2-5. Types of faults that can occur during the implementation stage

Type of Fault	Description
Incompleteness	System does not implement all specified requirements.
Unfounded Functionality	Functions exist which are not traceable to the requirements specification.
Logic and Control Faults	Logically incorrect implementations of the design exist or control structures incorrectly implement the design.
Data Operations and Computations Faults	Data operations incorrectly implement the design or computations are erroneous.
Lack of Consistency	Inconsistencies exist such as user interface inconsistencies (Help option is not always found in the same place on different displays) and output inconsistencies (report formats vary widely). Conflicts exist in passing data between modules (whether intra- or inter-system interfaces).
Inadequate Operational Concept	The implemented system does not conform to the specified operational concept.
Inadequate Human Computer Interface	The implemented system does not provide a sufficient human-computer interface.
Lack of Sufficient Database Design	The implemented database structure does not capably handle data storage, retrieval, and modification.
Safety Problem/Hazard Conditions	System lacks robustness to handle potential failures, specifically safety problems and hazard conditions.
Lack of "Fail-Safe/Soft-Failure" Operations	System lacks ways (safeguards) to degrade gracefully in the event of a failure.
Good Software Engineering Practices	Good software engineering practices, such as use of exception handlers, have not been applied in producing the design.
Inadequate System Documentation	Source code, in-line documentation or user documentation is ambiguous, difficult to understand, or non-existent.
Knowledge Base Faults	An error or inconsistency exists in the knowledge (rules, frames, etc.) which results in incorrect, unpredictable, or undesired system behavior.

**Table 4.1.4.2-6. Decomposition of identified faults
for the implementation stage**

Major Fault	SubFaults
Incompleteness	<ul style="list-style-type: none"> • Requirement or specification not addressed by code • Failure to test data imported by a procedure • Failure to implement a design element • Failure to comply with a levied standard such as POSIX, GOSIP, etc.
Unfounded Functionality	<ul style="list-style-type: none"> • "Extra" code that cannot be mapped to any design element exists • Code that does not conform to the definition of its corresponding design element
Logic and Control Faults	<ul style="list-style-type: none"> • Unreachable code • Flow control constructs not used correctly • A predicate (e.g., "if" statement) is incomplete, incorrect or transposed • Improper process sequencing • A loop or recursion has no exit or halting condition • Dynamic instruction modification exists • Failure to save/restore process communication • Recursion used improperly or language doesn't support needed recursion • Referenced but uncoded statement label or a missing statement label remains in the code • Call to a non-existent subprogram
Data Operations and Computations Faults	<ul style="list-style-type: none"> • Commercial off-the-shelf software capabilities proposed are not available or function contrary to assumptions • Incorrect data referencing • Mismatched parameter list • Definition or initialization fault • Data relating to the current iteration and previous iterations have been mixed • Error in use of data handling constructs • Variable misuse • Poor handling of input and output operations which affects throughput • Improper program linkages

Table 4.1.4.2-6 (Continued).

Major Fault	SubFaults
Lack of Consistency	<ul style="list-style-type: none"> Internally interfacing procedures have data arameters/arguments which directly contradict each other Internally interfacing procedures have data parameters/arguments which indirectly contradict each other Externally interfacing procedures have data parameters/arguments which directly contradict each other Externally interfacing procedures have data parameters/arguments which indirectly contradict each other One procedure directly contradicts another One procedure indirectly contradicts another Inconsistency in units of data (e.g. pounds or kilograms) Incorrect communication protocols and external data mismatches
Inadequate Operational Concept	<ul style="list-style-type: none"> System causes extra steps to be taken to perform a previously manual task Operational concept not supported by the system or key steps have been omitted
Inadequate Human Computer Interface	<ul style="list-style-type: none"> During execution, it is not clear what the system is doing based on its displays and outputs Menus and menu items are not consistently in the same locations on different displays Displays, menus, reports are not adequately labelled
Lack of Sufficient Database Design	<ul style="list-style-type: none"> Database structure fails to represent all data required Database structure incorrectly represents data (e.g., format) Database structure represents extraneous data not required by the system Database structure not normalized if relational
Safety Problem/Hazard Conditions	<ul style="list-style-type: none"> Code lacks ability to handle safety problems Code lacks ability to handle hazard conditions Code fails to account for failure modes Code lacks exception handling features

Table 4.1.4.2-6 (Continued).

Major Fault	SubFaults
Lack of "Fail-Safe/Soft-Failure" Operations	<ul style="list-style-type: none"> • Code lacks "fail-safe" operation • Code lacks "soft-failure" operation • Code does not provide a means of degrading gracefully in the event of a failure
Lack of Good Software Engineering Practices	<ul style="list-style-type: none"> • Frequent error traps have not been provided or do not have recovery mechanism • Modules possess weak modularity • Modules possess excessively rigid control structure
Inadequate System Documentation	<ul style="list-style-type: none"> • Source code, in-line documentation, or user documentation does not exist • Source code, in-line documentation, or user documentation is ambiguous or difficult to understand • Source code, in-line documentation, or user documentation is incorrect

Table 4.1.4.2-6 (Continued).

Major Fault	SubFaults
Knowledge Base Faults	<p><u>Syntax Faults</u></p> <ul style="list-style-type: none"> • Missing goal in rule • Unobtainable data item • Unsatisfiable condition • Unused askable declaration • Useless rule • Dead-end rule • Missing values • Cyclic inference chain (direct cycle) • Duplicate rule pair (duplication) • Subsumed rule pair (subsumption) • Conflicting rule pair (direct inconsistency) • Redundant condition • Unobtainable environment (unreachable node) • Redundant rule (relevance) • Dead-end nodes • Indirect cycle defects • Indirect inconsistency defects • Extended structure check duplication defects • Extended structure check subsumption defects • Contradiction in rule • Contradiction in IF clause • Contradiction in THEN actions • Violation of daemon reference syntax • Attribute value syntax error <p><u>Knowledge Faults</u></p> <ul style="list-style-type: none"> • Ambiguous rules • Missing rules • Semantic inconsistency defects • Logical completeness defects • Lower/upper numeric range defects • Set of legal values defects • Violations of restrictions on number and type of values for knowledge attributes • Incompatibility of attribute values • Disagreement with argument data type specifications • Incompatible variable and reference combination in rule • Keyword not recognized • Variable not recognized • Comparison not recognized • Reference not recognized • Semantic type or class violation

Table 4.1.4.2-6 (Continued).

Major Fault	SubFaults
Knowledge Base Faults	<ul style="list-style-type: none"> • Necessary semantic knowledge is missing • Rule condition redundant with present state-path <p><u>Structure Faults</u></p> <ul style="list-style-type: none"> • Knowledge structure is inappropriate for required functionality • Knowledge structure is insufficient to represent all needed knowledge • Knowledge structure design will cause performance problems • Hierarchical structure is awkward or needlessly redundant <p><u>Object-Oriented Programming Faults¹</u></p> <ul style="list-style-type: none"> • "Excess" number of methods in class • "Excess" number of instance variables in class • "Excessively" long method (number of function calls) • Module excessively larger than others • "Excessively" broad class graph • "Excessively" deep class graph • "Excessively" disjoint class graph (number of top classes) • "Excessively" unbalanced class graph (standard deviation of (subclasses/class)) • Parents at different abstract levels (number of levels) • Different length multiple inherit paths (number of levels) • Excessive number of parent classes • Class contains non-local method • Incomplete specialization • Public interface to class other than via class methods • Implicit class to class communication • Direct reference to object instance variable from another object • Incomplete specialization • Method not used • Instance not used • Module/class/instance variable not used • Module contains no classes or method contains no code

¹Note: What is considered "excessive" depends on the particular application, user, and analysis/development method

Table 4.1.4.2-6 (Continued).

Major Fault	SubFaults
Knowledge Base Faults	<ul style="list-style-type: none">• Class contains no methods / no instance variables• Class contains no instances• Circular object instance graph• No constructor defined for class• No destructor defined for class

Table 4.1.4.4-1 Types of Assurances Provided by Each V&V Class and appropriate V&V Methods: at the Requirement Stage⁷

V&V Class	Assurances Provided	Appropriate V&V Methods ⁸
Class 3	<p><u>Limited</u> assurance that the requirements meet the seven key criteria ⁹ i.e., the requirements must be:</p> <ol style="list-style-type: none"> 1) <i>complete</i> – they exhaust the client's considerations of concern; all terms/phrases in requirements are defined (e.g., if data driven targeting is mentioned, it must be defined). 2) <i>correct</i> – their specifications accurately capture the client's intent and understanding, and they agree with known facts, where relevant 3) <i>clear</i> – their specifications are expressed unambiguously, using language (or a formalism) that is familiar and understandable to all of the clients, the intended users, and the developers 4) <i>explicit</i> – every necessary (or presumed or assumed or implicit or implied) requirement or requirement-aspect is somewhere present as a clear aspect of some requirement specification 5) <i>consistent</i> – none of the requirements logically conflict with any other requirement 6) <i>testable</i> – the means for determining whether the requirement has been satisfied is either explicitly stated or can be readily, and uncontroversially, determined 7) <i>separately identifiable</i> – each separate requirement is allotted to a single specification and is prioritized; specifications do not contain compound requirements 	<p>Formal Requirements Review (FRR) (Subject Matter Experts (SMEs) and users, editorial) Requirements Analysis</p> <p>FRR (More SMEs than users) Requirements Analysis</p> <p>FRR (editorial) Requirements Analysis</p> <p>FRR (requirements analysis specialist, recommend automating with a CASE tool) Requirements Analysis</p> <p>FRR (requirements analysis specialist, recommend automating with a CASE tool) Requirements Analysis</p> <p>FRR (requirements analysis specialist) Requirements Analysis</p> <p>FRR (editorial followed by requirements analysis specialist performing requirements "break up") Requirements Analysis</p>

⁷This and the following two tables are replications of Tables 6.4-1 through 6.4-3 in Volume 5. ⁸Refer to Table 2.2.1-1

⁹These are considered a minimum set of the criteria concerning requirements as synthesized from a variety of key sources (e.g., Do-2167A; NSAC-39; IEEE7-4.3.2; Davis, 1990)

Table 4.1.4.4-1 (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods ^a
Class 2	<p>All Class 3 assurances.</p> <p><u>Moderate</u> assurance that the requirements meet the seven key criteria.</p> <p>Moderate assurance that requirements concerning the Human Computer Interface and the Operational Concept are adequately specified.</p>	<p>All Class 3 methods</p> <p>Semi-formal methods</p> <p>FRR (requirements analysis specialist)</p>
Class 1	<p>All Class 3 and Class 2 assurances.</p> <p><u>Extended</u> assurance that the requirements meet the seven key criteria, particularly by use of formal representations and animations.</p> <p>Extended assurance that requirements concerning the Human Computer Interface and the Operational Concept are adequately specified.</p>	<p>All Class 3 and Class 2 methods</p> <p>For <u>complete</u>: FRR (top notch SMEs)</p> <p>All others: Formal methods</p> <p>Semi-formal methods</p> <p>Prototyping</p>

^aRefer to Table 2.2.1-1

Table 4.1.4.4-2. Types of assurances provided by each V&V Class and appropriate V&V methods: at the Design Stage

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 3	<u>Limited</u> assurance that the design meets the seven key criteria; i.e., the requirements must be:	
	1) <i>complete</i> – identification of elements of the design purporting to implement each specific requirement.	Requirements Tracing
	2) <i>correct</i> – limited evaluation that each design element, corresponding to a required functionality, is reasonable and would work if implemented.	Formal Design Review
	3) <i>clear</i> – design specifications are expressed unambiguously, using language (or a formalism) that is familiar and understandable.	Formal Design Review
	4) <i>explicit</i> – every necessary (or presumed or assumed or implicit or implied) design element or design element-aspect is somewhere present as a clear aspect of some design specification.	Formal Design Review
	5) <i>consistent</i> – none of the design elements logically conflict with any other design element (e.g., units used, significance, precision, etc.).	Formal Design Review
	6) <i>interfaces consistent</i> – Internal: none of the design elements have data flow conflict with interfacing design elements in the system; External: none of the design elements have data flow conflict with interfacing design elements in other external systems.	Formal Design Review
	7) <i>database design sufficient</i> – the database structure is a logical, consistent, normalized (if applicable) representation of the data to be stored/retrieved/manipulated.	Formal Design Review
	Identification of all design elements not traceable to a requirement (unintended functions).	Requirements Tracing
	Limited evaluation of the adequacy of the operational concept.	Formal Design Review
	Limited assurance of the adequacy of the knowledge structures.	Knowledge Engineering Analysis
	Limited evaluation of the adequacy of the Human-Computer Interface.	Formal Design Review

Table 4.1.4.4-2 (Continued).

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 2	<p>All Class 3 Assurances</p> <p>Moderate assurance that the design meets the seven key criteria.</p> <p>Moderate evaluation of the adequacy of the operational concept.</p> <p>Moderate evaluation of the adequacy of the Human-Computer Interface</p> <p>Limited evaluation of the adequacy of the proposed system architecture.</p> <p>Limited evaluation of the adequacy of the logical partitioning of functions to be accomplished by hardware, software, and human personnel.</p> <p>Moderate analysis of the potential failure modes and robustness of proposed architecture/design, with special attention to hazard conditons and safety problems.</p> <p>Moderate assurance of the adequacy of the knowledge structures.</p>	<p>All Class 3 techniques</p> <p>All Class 3 techniques, design review, data and control flow analysis</p> <p>Operational Concept Analysis</p> <p>User Interface Inspection</p> <p>System Engineering Review</p> <p>System Engineering Review</p> <p>Failure Mode, Effects, Causality Analysis (FMECA)</p> <p>Knowledge Engineering Analysis</p>

Table 4.1.4.4-2 (Continued).

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 1	<p>All Class 3 and Class 2 assurances.</p> <p><u>Extended</u> assurance that the design meets the seven key criteria.</p> <p>Extended evaluation of the adequacy of the operational concept.</p> <p>Extended evaluation of the adequacy of the Human-Computer Interface.</p> <p>Moderate to extended evaluation of the adequacy of the proposed system architecture.</p> <p>Moderate to extended evaluation of the adequacy of the logical partitioning of functions to be accomplished by hardware, software, and human personnel</p> <p>Extended analysis of the potential failure modes and robustness of proposed architecture/design, with special attention to hazard conditions and safety problems.</p> <p>Extended evaluation of the safeguards designed to provide "fail-safe" and "soft-failure" operation.</p> <p>Moderate to extended evaluation of the developer's use of good software engineering practices.</p> <p>Extended assurance of the adequacy of the knowledge structures.</p>	<p>All Class 3 and Class 2 methods</p> <p>Design Animation (DA) via SREM or other automated semi-formal method tools</p> <p>Design Animation, Operational Concept Analysis</p> <p>Design Animation, User-Interface Inspection</p> <p>System Engineering Review</p> <p>System Engineering Review Uses of automated System Requirements Engineering Method (SREM) tool (or other semi-formal method)</p> <p>FMECA, Design Animation</p> <p>Design Animation, Formal Design Review</p> <p>Formal Design Review</p> <p>Knowledge Engineering Analysis</p>

¹ Refer to Table 2.2.2-1

Table 4.1.4.4-3. Types of Assurances Provided by Each V&V Class and Appropriate V&V Methods: At Implementation Stage

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 3	<p><u>Limited</u> assurance that the implemented system meets these criteria; i.e., the system must be:</p> <ol style="list-style-type: none"> 1) <i>complete</i> – identification of system elements purporting to implement each specific requirement. 2) <i>unfounded functionality</i> – identification of all system elements not traceable to a requirement. 3) <i>logic and control correct</i> – limited evaluation of logic and control. 4) <i>data operations and computations correct</i> – limited evaluation of data operations and computations. 5) <i>consistent</i> – the operational system is free of conflicts, such as user interface inconsistencies (Help or Exit options are not always found in the same place on different displays), output inconsistencies (report formats vary widely), parallel equivalent function inconsistencies (same function being performed on different pieces of hardware are not consistent), etc. <u>Internal</u>: no conflicts exist in passing data (variables) between modules of the system. <u>External</u>: no conflicts exist in passing data between modules and interfacing modules of external systems. 6) limited evaluation of the adequacy of the Operational Concept. 7) limited evaluation of the adequacy of the Human-Computer Interface. 8) <i>database sufficient</i> -- the implemented database structure capably handles data storage/retrieval and manipulation. <p>Limited assurance that system will perform correctly under random operating conditions.</p> <p>Limited assurance that good software engineering practices are in use.</p> <p>Limited evaluation of the adequacy of the system's documentation (in-line, user, source code).</p> <p>Limited assurance that the knowledge base is correct and sufficient.</p>	<p>Requirements Tracing</p> <p>Requirements Tracing</p> <p>Formal Customer Review Functional Testing</p> <p>Formal Customer Review Functional Testing</p> <p>Formal Customer Review Functional Testing Scenario Testing</p> <p>Operational Concept Analysis Formal Customer Review Functional Testing</p> <p>User Interface Inspection Formal Customer Review Functional Testing</p> <p>Functional Testing</p> <p>Random Testing</p> <p>Process audits, inspections inspections</p> <p>Desk Checking</p> <p>Knowledge Engineering Review</p>

Table 4.1.4.4-3. (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 2	All Class 3 assurances.	All Class 3 methods
	<u>Moderate</u> assurance that the system meets the criteria from Class 3.	Formal Customer Review Structured Walk-throughs Deck Checking Functional Testing Syntax Checking Knowledge Checking Anomaly Testing (e.g., lint) Structural Testing Random Testing Boundary Testing Scenario Testing
	Limited assurance that no hazards or safety problems exist.	Formal Customer Review Structured Walk-throughs Deck Checking FMECA Heuristic Testing
	Limited assurance of robustness ("fail-safe", "soft-failure").	FMECA Robustness Testing
	Moderate assurance that good software engineering practices are in use.	Process Audits, Code Inspections
	Moderate evaluation of the adequacy of the system's documentation. Moderate assurance that the knowledge base is correct and sufficient.	Desk Checking Desk Checking

Table 4.1.4.4-3. (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods ¹
Class 1	All Class 3 and Class 2 assurances.	All Class 3 and 2 methods
	<u>Extended</u> assurance that the system meets the Class 3 criteria.	Algorithm Analysis Functional Testing Data-Flow Analysis User Interface Inspection Data Interface Inspection Process Trigger/Timing Analysis Operational Concept Analysis Syntax Checking Knowledge Checking Anomaly Testing (e.g., lint) Structural Testing Control-Flow Analysis Model Analysis Formal Customer Review Random Testing Boundary Testing Scenario Testing
	Extended assurance that no hazards or safety problems exist.	Algorithm Analysis FMECA
	Extended assurance of robustness. Assurance of soft failure.	Robustness Testing
	Extended assurance that good software engineering practices are in use.	Process Oriented Audits
	Extended evaluation of the adequacy of the system's documentation.	Desk Checking
	Extended assurance that the knowledge base is correct and sufficient.	MetaCheck

1. Refer to Table 2.2.3-1

Figure 4.2-1 Questionnaire to Determine the Recommended Class of V&V

<p>STEP 3</p> <p>OVERVIEW OF STEPS TO DETERMINE APPROPRIATE LEVEL OF V&V</p>
--

<p>Purpose:</p>

To determine how rigorously your system needs to be examined.

<p>What to do:</p>

- Complete the three-part questionnaire given on the following pages.
- Use the first part to assess the complexity/control aspects of your system.
- Use the second part to assess the concern you may have if your system fails, under worst-condition assumptions.
- Use the results of these two parts to determine, in the third part, what the suggested level of V&V is for your system:

Figure 4.2-1 (continued)

Figure 4.2-1 (continued)

STEP 3 (PART 2)
ASSESSING REQUIRED INTEGRITY

Purpose:

To determine the concern you would have if your system failed, under worst condition assumptions. Your level of concern determines the "required integrity" of the system -- how important it is that the system not fail. There are three levels of required integrity: low, medium, and high.

What to do:

- Consider the statements below. Begin at "start" and continue until "stop."
- Record the required integrity rating:

START

A Consider some of the economic, legal, environmental, ethical, and business consequences of a failure or incorrect operation of this system:

• injury or death to plants and animals	• destruction or pollution of the environment or ecosystem	• loss of human lives
• interruption of system service	• disruption of the system's mission	• human injuries
• financial loss	• impact on the availability or operation of other systems	• long-term health problems
• loss of information	• loss of opportunity	• discomfort to people
• inconvenience to people	• impact on an organization's capability to perform	

B If you consider the consequences of A to be unacceptable from any perspective, then the required integrity rating is High.

C If you consider the consequences of A to be reasonably acceptable from all perspectives, then the required integrity rating is Low.

D If you consider the consequences of A to be somewhere between B and C, then the required integrity rating is Medium.

STOP

Figure 4.2-1 (continued)

STEP 3 (PART 3)
DETERMINING SUGGESTED V&V CLASS

Purpose:

- To determine the suggested class of V&V for your system. There are three classes, from least to most rigorous: Class 3, Class 2, and Class 1. The more rigorous the class, the higher your assurances that the system does not contain faults. However, the more rigorous the class, the higher the expense and (usually) the longer the testing.

What to do:

- Examine the row and column headings of the table below.
- Find the row that corresponds to the judged control/complexity rating in Part 1.
- Find the column that corresponds to your decision concerning the required integrity from Part 2.
- Record the number at the intersection of this row and column. This is the suggested V&V class for your situation. That is, if judged control/complexity is Low to Moderate, and required integrity is High, then the V&V class is Class 2.

Control/Complexity	REQUIRED INTEGRITY		
	Low	Medium	High
Quite High	2	2	1
Moderately High	2	2	2
Low to Moderate	3	3	2

Table 4.2.1-1 Determining the Appropriate Guideline Package

STEP 4:
DETERMINING THE APPROPRIATE "SUGGESTED GUIDELINE PACKAGE"

Purpose:

To determine the appropriate set of suggested methods. The sets are called the "Suggested Guideline Packages A-P". These Guideline Packages are contained in Section 2.

What to do:

- Review the results from the first three steps.
- If your stage of development, from Step 1, is Modification then do not use the tables on the next page. You will use Suggested Guideline Package P. Go to Section 2 now.
- Find the row in the table below that corresponds to the aspects of the system you need to examine (from Step 2).
- Find the column of that table that corresponds to your development stage (from Step 1) and the V&V class (from Step 3).
- Find the cell for that table, that row, and that column. This is the letter of your Suggested Guideline Package.
- Record the Guideline Package suggested.
- Go to Section 2.

Type of System Component	Requirements Stage			Design Stage			Implementation Stage		
	V&V Class			V&V Class			V&V Class		
	Class 3	Class 2	Class 1	Class 3	Class 2	Class 1	Class 3	Class 2	Class
Knowledge Base	A	B	C	D	E	F	G	H	I
Highly Reusable	A	B	C	D	E	F	J	K	L
System and Other	A	B	C	D	E	F	M	N	O

Table 4.2.1-2 Sample Listing of Recommended V&V Methods

GUIDELINE PACKAGE: A

Lifecycle Phase: Requirements

V&V Class: 3

Component Type: All

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action ²	Ref ¹
	Formal Requirements Review (FRR) applies to all assurances	S = Static D = Dynamic	All assurances are considered at the same time when reviewing the Requirements Document with Requirements Analysis and FRR	Small Team: 1- IV&V Leader, 1- Subject Matter Expert, 1- System/ Software Engineer Team A: 1- IV&V Requirements Analyst, 1- Editorial Expert	Limited: Reasonably thorough review of requirements document via independent study followed by limited group discussion involving at least IV&V and SE	A = Accept: fix problems, exit to next method or else stop V&V. CA = Conditional Acceptance: keep V&V team in place, fix problems, repeat accelerated V&V with same method, focusing on problems encountered. R = Reject: Cease V&V activity, do not fix problems, turn problems over to program manager to consider major overhaul.	
1	Requirements Analysis	S	Completeness, Correctness, Clarity, Explicitness, Consistency, Adequate Concept of Operations, Adequate Human-Computer Interface, Adequate System Architecture, Adequate Function Partitioning, Adequate Database Design, Contained Safety and Hazard Conditions, Fail-Safe/Soft-Failure, Good Software Design Practices	Team A	Limited	A: #CRD = 0 AND #NRD ≤ 4 AND %NRD < 8 CA: #CRD ≤ 4 AND #NRD ≤ 6 AND %NRD ≤ 12 R: #CRD > 4 OR #NRD > 6 OR %NRD > 12	R32

¹ References: Entries beginning with an "R" are literature citations found preceding the guideline plan. Entries beginning with a "P" are Guideline Procedures found in Section 3.

² Measures: All measures concerning requirements involve the findings of reviews and analyses concerning deficiencies in the text of the Requirements which need to be classified as critical or non-critical. All measures concerning system quality of the design or implementation involve the following three measures: Show-Stoppers (SS): Very severe problems which represent a major flaw in conceptualization or practice. They indicate a need for thorough re-evaluation and revision. Majors (Maj): Problems which cut across several broad system aspects or subsystems, which will require extensive rework. However, they do not invalidate the conceptualization or implementation approach. Minors (Min): Problems which are typically restricted to a specific function or structure element.

#CRD = Number of critical requirements that are deficient

#NRD = Number of non-critical requirements that are deficient

#Maj = Major Problems #Min = Minor Problems

%NRD = Percentage of the total set of requirements that are NRDs (#NRD/N, where N = total set of requirements)

%UF = Percentage of unintended functions (#UF/N, where #UF = number of unintended functions)

#SS = Show-Shoppers

Table 4.2.1-3 Sample Guideline Plan

Guideline Plan: *for Guideline Package A*

INSTRUCTIONS: Execute the numbered methods in the Guideline Package according to this table.

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Stop the V&V activity at "Exit" column.

NOTE: A "REJECT" decision by any method will terminate the V&V plan at that point.

Start Here	Do Next	Exit V&V Activity
1	2	Yes

P.0

WHEN TO USE THIS GUIDELINE:

GOAL:

Pre-Conditions/Trigger Conditions

PLANNING
PROGRAM MANAGER APPROVAL
SETUP
EXECUTION
ANALYSIS
ACCEPT/REJECT CRITERIA
REPORTING
PROGRAM MANAGEMENT REVIEW
OUTPUT/COMPLETION STATUS

Terminal, lowest level activity, is not discussed on subsequent pages

High level (non-terminal) activity (decomposes to lower level activities), discussed on subsequent pages

- o -- optional steps
- -- obligatory

the process is governed by a special metric, M2, which will be defined on the lowest level activity page, also defined in endnotes as item M2

-- see reference 23 in endnotes

Figure 4.2.2-1 Sample of V&V Guideline Procedure

5 EVALUATION OF V&V METHODS AS APPLIED TO EXPERT SYSTEMS

The USNRC and EPRI each provided an actual nuclear power related expert system (ES) for use in this project. One ES, denoted System A, was designed to assist boiling water reactor (BWR) operators in monitoring transient plant conditions and selecting appropriate emergency operating procedures (EOPs). The other ES, System B, aids a reactor safety team in monitoring and projecting the status of six critical safety functions at a plant during an event. This aspect of the project involved the application of selected conventional software V&V methods to Systems A and B and evaluation of an early version of the ES V&V guidelines as they specifically applied to these two systems. Conventional methods were applied in an early task prior to the development of draft ES V&V guidelines (Activity 3). Following the guideline development, techniques from these guidelines were applied to both systems (Activity 4). There is no supplemental volume describing this work.

Section 5.1 describes the selection of V&V methods as candidates to apply to Systems A and B. The remaining two sections discuss the results of applying the chosen methods.

5.1 Evaluation of V&V Guidelines for Application to Systems A and B

Earlier tasks of this project identified, classified, analyzed, and rated existing conventional and ES software V&V methods. In addition, an innovative system for determining a V&V class based on software complexity and required integrity was developed to aid in the selection of V&V methods.

The first step in selecting V&V methods for Systems A and B was the determination of the V&V Class for each system. Using V&V guidelines questionnaires and confirming with the system's users, both Systems A and B were determined to be V&V Class 2. The designation of Class 2 was based on an analysis showing that both systems had a moderately high complexity and either low or medium degree of required integrity.

Another important factor in selecting V&V techniques is the stage in the software development Life-cycle for each of these systems. For both Systems A and B, it was determined that they were in the implementation phase since development continued and on-site use, refinement, and debugging were in progress.

The components of each system were analyzed to determine the nature (i.e., formal verification, certification, and/or traditional system V&V) of V&V to be performed on them. Both systems possess declarative knowledge structures which implies the use of formal verification. System A contains no reusable components whereas System B utilizes MEDLEY, a development environment that is highly reusable. Neither of the inference engines for the two systems were reusable since they were specifically developed for these applications. Both systems were found to contain mostly nonreusable components such as interfaces and databases. This implies the use of traditional system V&V.

Using the aforementioned information on the V&V class, Life-cycle stage, and component structure for Systems A and B, the V&V guidelines that were developed in this project were used to assign specific V&V methods. For the assignment of conventional V&V techniques, four factors played an important role. These factors are:

- Life-cycle phase (due to the current Life-cycle phase of these systems, it is too late to apply some techniques)
- balanced mix of techniques to optimize the detection of a wide range of errors
- system specific characteristics
- on-going or planned V&V activities, techniques already used or planned for use should be assessed

The recommended conventional V&V techniques for Systems A and B are delineated in Table 5.1-1 along with their status, effectiveness and cost-benefit metric rankings.

In selecting V&V techniques for Systems A and B from the draft ES guidelines, Table 4.1.2-1 was used in conjunction with knowledge about individual system components, Life-cycle phase, and V&V class. The result of applying the V&V guidelines was the determination that guideline packages H, K, and N should be applied to System B while packages H and N should be applied to System A. The methods within these packages are presented in Table 5.1-2.

A comparison of the conventional V&V techniques selected for System A and B before the V&V guidelines with those selected after the development of the V&V guidelines was made to assess the effect of these guidelines. The earlier selection (see Section 3.0) of V&V methods resulted in a total of 18 techniques for both systems whereas the ES V&V guideline selection resulted in a total of 16 techniques for System A and 17 techniques for System B. Only five V&V methods were common to both selection processes: requirements tracing, functional testing, random testing, regression testing, and robustness testing. The purely conventional V&V techniques were almost exclusively dynamic with the exception of requirements tracing, but the post-ES V&V guideline development techniques included purely ES V&V techniques and a number of static testing techniques. This shift is substantiated by the work in later tasks regarding the identification and evaluation of unique or innovative ES V&V techniques and the greater importance given to static testing as an effective means of uncovering errors. Similarly, a mix of techniques with emphasis on static testing will be more cost-effective than a dynamic testing-oriented approach. Thus, changes in the selected V&V techniques for Systems A and B directly reflect the increase in knowledge and experience during the course of the project.

5.2 Methods as Applied to System A

The results of applying the recommended V&V techniques to System A were analyzed by evaluating the previous V&V activities already performed by the developer and contractor as well as actual use of the techniques on segments of the software. The intent was to evaluate the methods as applied to the system, not to evaluate the development contractor. Requirements tracing was found to be a highly effective, cost efficient method which detected numerous failure. Dynamic testing uncovered only four errors in the system, but was able to provide close to full coverage of the rules. This was deemed an effective method. Engineering checking (desk checking) was found to be a useful, but not very powerful method. This method might not be included in future method mixes. A test driver was used to facilitate functional and regression testing. These tests were more powerful than engineering checking, but the test driver was difficult to use. In general, the methods used were found to be effective at finding failure. The most powerful and easy to use methods were requirements tracing, followed by dynamic testing (stand-alone and with a test driver). Engineering checking was used, but was not a preferred method and could be omitted in future efforts.

Requirements tracing was performed by SAIC on an early generic version of the System A rule base. Requirements tracing utilized two documents describing System A which constituted an approximation to the requirements specification document and a listing of the System A rules. The rules written in "IF-THEN" format were compared to the documents. The purpose of this trace was to identify ambiguities, errors of omission or commission, inaccuracies, and extraneous conditions that occurred during the mapping from the requirements documents to the System A rules.

The general approach was to segment the requirements document into single-topic text blocks and try to find the rule(s) corresponding to these. Each text block, and rule, was characterized as having a clear (or unclear, or no) mapping, and various kinds of errors and problems were also coded. The requirements tracing procedure used an independent numbering and labelling scheme to eliminate dependencies on individual document labels. Some assumptions were necessary to perform this task including the system's capability to understand abbreviations and the accuracy of comments in the rule text.

The requirements tracing was performed on two sections of the reactor pressure vessel (RPV) Control Text Document (i.e., requirements specification) for System A that were entitled RPV Control and RPV Flooding. Tables 5.2-1 and 5.2-2 summarize the results of the requirements tracing performed on the System A rule base. The requirements tracing procedure identified a high number of instances in which specified actions in the requirements specification were not implemented in the rules. For example, 13 out of 67 (19%) of the RPV Control actions in the document were not present in the System A rules. Similarly, 6 out of 17 (35%) of the RPV Flooding actions were not present in the System A rules. The RPV Flooding entry conditions as well as graphs or tables from the RPV Control Text were not present in the rules.

Table 5.1-1 Selection of V&V techniques appropriate for System A and System B

V&V Technique Name	Ranking Against Criteria	Planned or Already Applied?	Final Selection
Requirements:			(too late)
Requirements Language Analysis	3	No	
Requirements Language Processing	3	No	
Mathematical Verification of Requirements	3	No	
Formal Requirements Review	1	No	
Requirements Tracing/Traceability Analysis	1	No	✓ (post hoc)
Design:			(too late)
Design Compliance Analysis	1	No	
Design Simulation	1	No	
PDL Analysis and Processing	3	No	
Formal Design Review	1	No	
Critical Timing/Flow Analysis	3	No	
Dynamic Testing:			
General/Statistical:			
Unit/Module Testing	2	No	(too late)
System Testing	1	Yes (System A)	✓
Random Testing	1	Yes (System B)	✓
Compilation Testing	1	No	(too late)
Reliability Testing	3	No	
Regression Testing	1	Yes (System B)	✓
Metric-Based Testing	3	No	
Statistical Record Keeping	3	No	
Software Reliability Estimation	3	No	
Ad hoc Testing	3	Yes (Both)	
Domain Testing	2	No	
Data-Flow Testing	3	No	
Functional Testing:			
Specific Functional Requirement Testing	1	Yes (System A)	✓
Simulation	3	No	
Model-Based Testing	3	No	
Assertion Checking	3	No	
Transaction-Flow Testing	3	No	

Table 5.1-1 Selection of V&V techniques appropriate for System A and System B (Continued).

V&V Technique Name	Ranking Against Criteria	Planned or Already Applied?	Final Selection
Dynamic Testing (cont.):			
Realistic Testing:			
Field Testing	1	Yes (System B)	✓
Scenario Testing	1	Yes (Both)	✓
Qualification/Certification Testing	3	No	
Simulator-Based Testing	1	Yes (Both)	✓
Stress Testing:			
Stress/Accelerated Life Testing	3	No	
Robustness Testing	1	No	✓
Limit/Range Testing	2	No	
Parameter Violation	2	No	
Performance Testing:			
Sizing/Memory Testing	3	No	
Timing/Flow Testing	3	No	
Bottleneck Testing	3	No	
Other (queue size, paging etc.)	3	No	
Execution Testing:			
Activity Tracing	3	No	
Incremental Execution	3	No	
Results Monitoring	3	No	
Thread Testing	3	No	
Using Generated Explanations	3	No	
Competency Testing:			
Gold Standard	1	Yes (Both)	✓
Effectiveness Procedures	3	No	
Workplace Averages	3	Yes (System A)	
Interface Testing:			
Data Interface Testing	2	No	✓
User Interface Testing	1	Yes (Both)	✓
Information System Analysis	1	No	✓
Concept of Operations Testing	1	No	✓
Organizational Impact Analysis/Testing	3	No	

Table 5.1-1 Selection of V&V techniques appropriate for System A and System B (Continued).

V&V Technique Name	Ranking Against Criteria	Planned or Already Applied?	Final Selection
Dynamic Testing (cont.):			
Structural Testing:			
Branch Testing	1	Yes (System B)	✓
Path Testing	3	No	
Statement Testing	1	Yes (System B)	✓
Call-Pair Testing	3	No	
Linear Code Sequence and Jump	3	No	
Test-Coverage Analyzer	1	Yes (System B)	✓
Conditional Testing	3	No	
Error-Introduction Testing:			
Error Seeding	3	No	
Fault Insertion	3	No	
Mutation Testing	2	No	✓

NOTE: The criteria against which the techniques were ranked are defined in Section 2.1.

Table 5.1-2 Suggested methods for V&V of Systems A and B.

Method	Knowledge Structure (Package H)	Highly Reusable (Package K)	Other (Package N)
Requirement Tracing	X		X
Desk Checking	X		
Automated Syntax Checking	X		
Semantic/Meta-Level Knowledge Checking	X		
Formal Customer Review			X
User Interface Inspection		X	X
Operational Concept Analysis		X	X
Algorithm Analysis			X
Data Interface Inspection			X
Automated Anomaly Testing			X
Process Oriented Audits			X
Functional Testing	X	X	X
Boundary Testing	X		X
Random Testing	X	X	
Regression Testing	X	X	X
Benchmarking		X	
Robustness Testing		X	X

Table 5.2-1 Summary of RPV control text document labeling

RPV Control Text Document Section	Number of Text Blocks Checked	Correct Mapping	Unimplemented Requirement	Ambiguity Exists	Discrepancy Found
RPV Control:					
Purpose	1		1		
Entry Conditions	5	5			
Operator Actions					
Actions	67	54	13 - (19%) 13	4	
Notes/Graphs	17	4	<u>7</u>		
Cautions	<u>18</u>	<u>11</u>	34 (31%)	—	<u>2</u>
Subtotal	108	74 (69%)		4	2
RPV Flooding:					
Purpose	1		1		
Entry conditions	11		11		
Operator Actions					
Actions	17	10	6 (35%)	1	
Notes/Graphs	5	1	4		
Cautions	<u>1</u>	<u>1</u>	—	—	—
Subtotal	35	12 (34%)	22 (63%)	1	0
Total	<u>143</u>	<u>86</u> (60%)	<u>54</u> (38%)		

Table 5.2-2 Summary of System A rule labeling

System A Rules Section	Number of Rules Checked	No Requirement	Additions in Rule(*)	Correct Mapping	Ambiguity Exists
RPV Control:					
RPV_ENTRY	8	2		6	
RC/L	18	3	2	13	
RC/LL	4	2	1	1	
RC/P	12	8	4		
RC/Q	<u>11</u>	<u>8</u>	<u>2</u>	<u>1</u>	—
Subtotal	53	23 (43%)	9	21	0
RPV Flooding:					
C_8_A	10	2		4	4
C_8_B	7	2		3	3
C_8_C	4			3	1
C_8_C_1	8	5			3
C_8_C_2	<u>3</u>	<u>1</u>	—	<u>2</u>	—
Subtotal	32	10 (31%)	0	12	11
Total	<u><u>85</u></u>	<u><u>33 (39%)</u></u>	<u><u>9</u></u>	<u><u>33</u></u>	<u><u>11</u></u>

*Note: A requirement was found for the rule, but rule contains additional antecedent(s) and/or consequent(s).

Overall, 54 out of the 143 text blocks (38%) from the requirements specification (RPV Control Text Document) were not represented in the rules in the early generic version of System A. Conversely, 33 of the 85 rules (39%) had no corresponding requirement (i.e., unintended functions). This requirements tracing was performed for about 30% of the System A rule base with a labor expenditure of only 20 staff hours.

A number of conventional V&V methods applied to System A by the developers included: statistical dynamic testing, functional dynamic testing, realistic dynamic testing, competency dynamic testing, and user interface dynamic testing. Emphasis was placed on the inference engine, system interfaces, and the knowledge base. Using nine test scenarios, 57% of the rules were exercised resulting in the identification of four errors. Additional regression, random, and ad hoc tests covered 97.5% of the rules without identifying any additional errors.

The System A team has found this test driver very useful. It has resulted in the detection of approximately three significant items, mainly incorrect programming assumptions. Due to the tabular, textual format of the output, verifying the actual results is a lengthy process. In general, the development contractor found this method to be somewhat powerful, but difficult to use.

The factory acceptance test and site acceptance tests involved comparison to output from the plant simulator. A "gold standard", the actual EOPs, was also used in some tests, but did not reveal any errors.

Another technique used by the development contractor was "engineering checking" of the rule base, which falls under the category of desk checking. For example, an engineer examined the naming of the rules and parameters to ensure that non-computer personnel would be able to read and understand the rules. In addition, the engineer examined the rulebase as a whole to identify efficient, easier ways to implement logic (such as how to handle entry conditions). This method was found to be useful, but not very powerful. It did help to identify approximately 10 problems, such as misstated output variables. It is definitely viewed as an easy-to-use method.

The development contractor has written a test driver to facilitate testing of the system. Input files of plant parameters such as RC pressure (files contain one "cycle" of data) are used to fire rules. The user can also enter data interactively. After the inputs have been accepted by the system, the test driver can be used to view the results (what rules were fired, what boxes on the flowcharts should be turned on, etc.). The output data is written to a file in tabular, textual format. Once the flowchart displays and the inference engine have been integrated, the actual flowchart displays will be used to check actual results versus expected results (instead of the textual output file used now).

The test driver described above has basically been used for debugging purposes to facilitate modification/correction of the inference engine. In fact, the test driver has been used to perform many types of testing. To test specific functions of the inference engine and/or rule base, input data files are created and run using the test driver. This represents functional testing. Plant parameters are set at values just below or just above set points, which constitutes boundary testing. Plant parameters are selected randomly and entered interactively or in a data input file, constituting random testing. Finally, a set of data input files have been maintained and are run after major changes to the inference engine and/or rulebase, which constitutes regression testing.

The System A team has found this test driver very useful. It has resulted in the detection of approximately three significant items, mainly incorrect programming assumptions. Due to the tabular, textual format of the output, verifying the actual results is a lengthy process. In general, the development contractor found this method to be somewhat powerful, but difficult to use.

5.3 Methods as Applied to System B

Analysis of the results of applying the recommended V&V techniques to System B was performed by evaluating the previous V&V activities already performed by the developer and contractor as well as actual use of the techniques on segments of the software. The intent was to evaluate the methods as applied to the system, not the development contractor or the V&V agent. Formal requirements and design reviews were found to be effective methods when applied to System B. Requirements tracing uncovered only five errors, but was found to be an effective, cost-efficient method. Of the many

dynamic testing techniques applied, functional testing was found to be the most powerful, followed by boundary and random testing.

The USNRC held a formal requirements review and formal design review during the early part of the System B Life-cycle. Much analysis of the informally documented requirements was performed. Next, the user interface screens were designed on paper and were analyzed. The main goal of this review was to ensure that the user could understand what the system was advising. Human factors were kept in mind while performing this review. As the system design began to evolve from the requirements and user interface screens, it was decided to use critical safety function trees and rules of thumb to implement the system. The rules of thumb were put into a large table (paper chart, wall mural size), and a requirements/design review was performed. As part of the review, some members of the Reactor Safety Team used the table for a paper exercise. It was determined that the system would be useful, and also that it truly needed to be automated due to the amount of data and number of rules of thumb to be handled. Both formal review techniques were found to apply well to System B.

Requirements tracing has been performed informally for System B by the USNRC. As there is only one programmer working on the system, the designer/System B leader would describe desired system functionality verbally to the programmer. The programmer would then document and submit his understanding of the requirement. The System B leader kept these requirements in a list, and would use the list to ensure that desired functionality existed in the system when a new version was released. Similarly, at a year end review, the designer/System B leader would make sure that every item on the list had been completed. (Note that the requirements list has since been formalize. This technique was quite useful as applied to System B with no specific limitations.

A more formal requirements tracing was performed on a generic, non plant-specific version of the System B rule base by SAIC. This process involved using three documents: the System B Requirements Specification, the System B critical safety function graphs (or trees), and the listing of the System B rules. The Requirements Specification lacked information relating to the critical safety functions and the application-specific rules. The graphs were used to supplement this information, and were also used as a pseudo second level document. Thus, the Requirements Specification represented the top level requirements, the graphs represented the second level requirements, and the rules represented the third level requirements.

Critical safety function definition text in the Requirements Specification was compared to the critical safety function graphs. Other sections of the System B Requirements Specification were used as information sources, when necessary. The graphs were then compared to the listing of the System B rules. The purpose of this process was to identify unimplemented requirements, incorrectly implemented requirements, and unintended functions. For the Requirements Specification, 45 (18%) of the 253 requirements were traced resulting in the identification of one (2.2%) unimplemented or incorrectly implemented requirement. For the critical safety function graphs, all 18 requirements were traced with a finding of no errors. All 203 of the rules were traced resulting in the discovery of four (2%) unintended or anomalous functions.

A requirements tracing tool called SuperTrace and its related pre-processor, the SuperTrace Front End Processor (SFEP) were used to trace the System B requirements through the graphs to the rule base (as a precursor to the knowledge base certification, see Section 3). The SFEP was used to automatically mark and extract requirements from the softcopy System B Requirements Specification based on user defined search words such as "shall", "must", and "should". Output from SFEP was an ASCII file of all requirements uniquely numbered with a requirements ID. SFEP was also used to categorize requirements based on user-defined key words.

After their synthesis, the ASCII files created by SFEP (requirement text and keywords) were input to SuperTrace which then generated candidate links using the assigned keywords. Candidate links are a list of the parent requirements and their potential children requirements (or graphs and rules in this case). These links were then analyzed which is also referred to as verifying traceability. Thus, the analyst verified that all children requirements trace back to a parent level requirement.

SuperTrace was also used to verify completeness which is defined as ensuring that the top level requirement is fully satisfied by all children requirements (or rules, in this case) which are linked to it. As with candidate links, SuperTrace was used to simultaneously view the parent and children requirements using split and scrolling screens.

After completing the aforementioned procedures with SuperTrace and SFEP, a traceability anomaly and completeness anomaly report were generated. The traceability anomaly report listed all the System B rules and graph nodes which did not link back to at least one requirement from the Requirements Specification. The completeness anomaly report listed all the System B Requirements Specification requirements which were only partially satisfied or which were totally omitted from the graphs and/or rules.

The requirements tracing using SFEP and SuperTrace examined 45 of the 253 requirements (18%). These 45 requirements were traced to 18 critical safety function graphs which, in turn, were traced to 203 rules. One requirement from the Requirements Specification was found to be anomalous. In addition, four rules were found to point to a non-existent critical safety function graph node and were thus marked as anomalous.

Early in the development cycle of System B, the USNRC held formal requirements reviews and formal design reviews. These included analysis of informally documented requirements, paper design of user interface screens and human factors considerations to ensure that the user could understand the output from System B. During this process, rules of thumb were tabulated and critical safety function trees were created. This documentation proved to be useful in verifying some aspects of the system.

System B developers, users, and V&V agent almost exclusively utilized dynamic testing techniques for V&V. One technique that was used, denoted random testing, involved having the tester, who had no previous knowledge of System B, input random cases and then determine if the correct rules fired for those cases. These tests did not uncover any errors, but it was performed after "per-rule" tests which did identify errors. Another test method used a random number generator to input random setpoint values for each case. This method did not discover any errors.

During user testing, a partially domain driven automated test tracking system or tool, written in Lisp and C++ and operational on an IBM personal computer, was developed by the V&V agent. Focusing on the code, the system generated data to test the inference engine's ability to handle different types of rules (equivalence partitioning). This tool provides four major capabilities: logic/syntax checking, functional (positive) testing, boundary/random testing, and a testing "slide demo."

The System B tool performs static logic/syntax checking of the knowledge based on the MYCIN standard from Stanford University. The following logic checks can be performed by this tool: consistency of rules with the tree, consistency of the tree with the rules, assurance that all parent nodes have children, assurance that all children nodes have parents, etc. All identified anomalies are reported to the user for further investigation. This is a powerful and easy to use method.

Functional testing with the V&V tool involves the generation of test data files which are compatible as simulated input to System B which emulate the actual input during an event from the Emergency Response Data System (ERDS). This V&V method requires the tool to select all the rules which lead to a conclusion of a particular plant state (e.g., "NORMAL"), select suitable parameter values to fire those rules and initialize the goal tree-status tree, and create a test data file with the ability to fire 200 rules. Nine rules (events) are fired simultaneously to test the combination of rules. A file of expected results is also generated for validation by a nuclear subject matter expert and used to compare to the actual output from System B. This method is powerful, but not particularly easy to use.

The boundary/random testing capability of the System B V&V tool is achieved by the generation of 200 test cases or scenarios using a random number generator which picks values within the range of each variable, thus building input files for System B. It should also be noted that these 200 test cases form a suite of scenarios for subsequent regression testing of later versions of System B. Boundary testing is accomplished by the same method as random testing except that the values which are selected are slightly above or below the allowable range. Both random and boundary testing are powerful and relatively easy to use methods.

The aforementioned tool's slide demonstration capability presents a step-by-step tutorial on how to test the system which includes actual System B screens, instructions, and a means of keeping track of the status of testing. Using a laptop computer set up adjacent to the System B computer, the user can record results of each test in parallel with the testing process.

Since System B is already "in the field", for all practical purposes, field, scenario, and simulator based testing have been performed by the developer and V&V agent. Scenarios were input to System B using a simulator at the USNRC Technology Training Center. The scenarios represented real world situations that would be expected during actual use of System B. No limitations were noted.

Using the previously discussed random number generator, the developer exercised System B continuously for a two day period by constantly feeding random setpoint values to the system. This type of testing is characterized as stress/accelerated life testing. Another related test involved determining if System B could "keep up" with the data stream which was being fed to it on two second intervals from another system. This type of testing is denoted timing/flow testing and did identify inadequacies in the hardware which was later replaced by a faster computer that solved this problem. Both stress/accelerated life and timing/flow testing were found to be directly applicable to System B with no limitations.

A Gold Standard Testing program was planned by the System B project leader (USNRC). This will involve a paper review by experts of the rules, trees, and set points followed by a review by other experts of scenarios with their own decision as to results. Finally, the project leader, a subject matter expert himself, would compare System B scenario results to his assessment.

An indirect form of interface dynamic testing was performed by the users and developers of System B. During the execution of previously discussed dynamic tests, a record was made in the test log whenever: (1) the system crashed; (2) a problem was found with the user interface; or (3) inadequate instruction/help was provided. Also, during the "function tree" tests, low level user interface functions such as "open window" were tested. This method seems to work equally well for conventional and expert systems.

A number of structural dynamic testing techniques were applied to System B by the developer and V&V agent. Statement testing consisted of testing each individual rule by running sufficient cases to fire each rule. This uncovered numerous errors, but it was also the first type of testing and it was performed on an early version of System B. Branch testing entailed checking that the inputs at the bottom of each AND/OR tree propagated up the right nodes to produce the desired status at the top node. Future plans include test-coverage analysis which involves tracking which rules and tree branches were executed by each scenario test case.

5.4 Method Ranking by System A and B Developers/Users

As a means of independently validating the ratings given to the V&V methods selected for both Systems A and B, the developers and users of these two systems were provided with the methodology for assessing the effectiveness of each of these techniques based on factors of power, ease-of-use, and cost-benefit. The results of the users' ratings as compared to the original ratings determined during this project is presented in Table 5.4-1. The results of this comparison, based on the final effectiveness ratings, show that, for those methods that were both recommended and actually applied to Systems A and B, the users generally rated them equal or better than the earlier activity in this project.

5.5 Conclusions

There were a number of valuable lessons learned through the V&V of Systems A and B. First, the ratings of methods will change from those presented in section 2 when being evaluated for application to a specific system. Second, requirements tracing was found to be a very powerful method which needs to be performed early in the Life-cycle in order

to positively influence a development effort. This method relies on a requirements document, so an ancillary lesson learned was that a formal requirements document must be written early and then be maintained throughout the Life-cycle. Third, dynamic testing was found to be effective at detecting failure in the software, but was not generally easy to apply. Functional testing appeared to be the most promising testing technique. Finally, engineering checking was found to be useful, but not very powerful, and would not be recommended for a future V&V effort for System A.

Table 5.4-1 Evaluation of V&V methods

		Power Rating (4 to 20) From Users (see Note 3)	Power Rating (4 to 20) From Volume 2 Report	Ease-of-Use Rating (4 to 20) From Users (see Note 4)	Ease-of-Use Rating (4 to 20) From Volume 2 Report	Relative Cost-Benefit (-16 to 16) From Users	Original Cost Benefit Measure for Class 2 From Volume 2 (or 6) Report	Effectiveness (100 to 600) From Users	Original Effectiveness Measure for Class 2 From Volume 2 Report
Method	Application								
Requirement	on Both Systems by	16	12	12	15	4	3	360	341
Desk Checking	on Both Systems	10	8	19	19	5	3	340	312
Automated	on System B	12	16	18	16	6	8	360	N/A
Semantic/Meta-	planned for System	N/A	N/A	N/A	N/A	N/A	6	N/A	N/A
Formal	planned for System	N/A	N/A	N/A	N/A	N/A	3	N/A	342
User Interface	planned for System	N/A	N/A	N/A	N/A	N/A	3	N/A	318
Operational	not applied or	N/A	N/A	N/A	N/A	N/A	0	N/A	296
Algorithm	not applied or	N/A	N/A	N/A	N/A	N/A	-7.45	N/A	211.5 (See Note 1)
Data Interface	not applied or	N/A	N/A	N/A	N/A	N/A	2	N/A	322
Automated	not applied or	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Process	not applied or	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Functional	on both systems	11	12	10	10	-3	-2 (See Note	265	288 (See Note 1)
Boundry	on both systems	9	10	12	13	-3	-1 (See Note	255	286
Random	on both systems	14	12	17	16	7	4	380	368
Regression	on both systems	13	11	18	19	7	6	375	356
Benchmarking	not applied or	N/A	N/A	N/A	N/A	N/A	-7	N/A	213
Robustness	not applied or	N/A	N/A	N/A	N/A	N/A	1	N/A	333
<p>Note:</p> <p>1) Averaged all sub-categories</p> <p>2) Used the measures for the top level category</p> <p>3) Grouping the factors hard power, broad power, formalizability, and HCI teatability</p> <p>4) Grouping the factors ease of mastery, ease of setup, ease of running, usage</p> <p>Note:</p>									

6 VALIDATION SCENARIO GENERATION

All conclusions and findings on validation scenario generation are based on Volume 6. The guidelines for generating validation scenarios were developed based on a strategy that included a broad assessment of nuclear power industry organization practices followed by an analysis of these validation scenario procedures.

In order to understand the purpose of this facet of V&V, the following definition of validation scenario is presented:

Validation scenarios are realistic dynamic tests of a software (or software and hardware) system which cover only the intended range of application of the software and are designed to sample important subsets of functions, usually for selected situations known to be challenging or problematic, to provide assurance that the system performs the tested functions with the required accuracy and performance. Validation scenarios are used after the extensive V&V activities described in Section 4 have been completed.

The activity of validation scenario generation is discussed in five sections. Section 6.1 presents the results of the assessment of current nuclear industry practices in selecting the type and number of validation scenarios for real nuclear software. Section 6.2 describes the analysis of the assessment in Section 6.1 with a resulting classification of validation scenario types along with their concomitant strengths and weaknesses. Section 6.3 delineates the methodology for selecting the type and number of validation scenarios based on the Previous two sections. Section 6.4 applies the methodology from Section 6.3 to Systems A and B, the two ESs used for many activities within this project. Finally, Section 6.5 discusses an innovative new, but as yet untested, method for generating knowledge based validation scenarios.

6.1 Industry Practices

An assessment of nuclear industry practices and the current accepted state of the art in software validation scenarios was performed by contacting nuclear industry software V&V personnel and reviewing pertinent literature on nuclear software validation scenarios. A total of 11 nuclear industry representatives were directly contacted and 29 nuclear software documents with validation scenario data were reviewed for this effort. The organizations included in this data base include nuclear utilities, national laboratories, EPRI, consultants, standards, guidelines, and computer software developers.

The key information which was obtained from each nuclear industry contact or document review can be summarized as answers to the following questions:

- 1) For specific software, what are the number and types of scenarios which are run before the organization is satisfied that the software has been adequately validated ?
- 2) What is the basis for determining the specific set of validation scenarios ?
- 3) How is the validation process, especially the selection of scenarios, controlled ?

The answer to the above three questions provided considerable insight and understanding of the current state of the art and acceptable practice in validation scenario application to nuclear power industry software.

Several common underlying themes emerged from the assessment contacts and document reviews. There is no set formula or rule-of-thumb that is used in determining an acceptable number and type of validation scenarios for nuclear software. The closest common thread is that these scenarios should represent the expected range of applications for the software. Thus, for example, if a pressurized water reactor (PWR) nuclear fuel management core physics code is being validated, it would be expected to be validated with fuel loadings of PWR fuel types in the range of uranium-235 enrichment,

power levels, burnup, boric acid concentration, and burnable poison that might be seen at an operating PWR. Validation scenarios outside the realm of expected application are not considered or used. This strategy is also applied to the use of options on software. Only option combinations which are physically realistic for the particular software's intended application are included in validation scenarios. Failure that could be detected outside the expected range of applications are searched for by the prior V&V described in Section 4.

Another important conclusion from this assessment is that the selection of validation scenarios is left up to either the code developer, if it is new software, or the end user(s), if it is a revised version of existing software. The subject matter expert usually decides on the validation scenarios, although final acceptance of the validation process is left up to a different individual. The extensive earlier V&V described in Section 4 is performed by a person or organization other than the developer. Revised versions of already validated software rely on a regression suite of validation test cases from the previous version which may be augmented by additional scenarios supplied by the user.

The validation process is usually controlled by a general software quality assurance and/or verification and validation procedure. This procedure states, in the most general sense, that software validation will be performed and sometimes assigns the functional responsibility for who develops scenarios and runs them as opposed to who is responsible for the review and acceptance of the validation process. Procedures do not provide any specific guidance on the nature or number of validation scenarios, but usually reference a general software QA or V&V standard such as ASME NQA 2.7.

One trend that was noted in this assessment is that software developed outside the end user's organization (e.g., software developed by EPRI or a consultant) would be delivered to the user with a set of standard validation cases which had been run and could be duplicated on the customer's computer. The customer would typically run additional validation scenarios as a means of ensuring the competency of specific analysts and plant-specific models with the software that may have different features than those provided with the developer's validation scenarios.

The assessment also found that, whenever feasible, the validation scenarios encompassed a mix of different types (as defined in Section 6.2 below), such as already validated code cases with plant data and regulatory requirements. Although importance was given to the comparison to actual measurements either at a nuclear power plant or experimental facility, the resulting data was also treated as less than perfect because of uncertainties in initial conditions and instrumentation accuracy. Thus, actual measurements are not treated as the panacea for acceptable software validation in the nuclear industry, but still have the greatest level of credibility.

A compilation of reported numbers and types of validation scenarios for nuclear power plant related conventional software shows that the validation scenario "mix" emphasizes Licensing, Plant, and Test types over Regression, Code and Basics types (as defined in the next section).

6.2 Classification and Characterization of Scenarios

An analysis of the results of the validation scenario assessment led to two important conclusions. First, scenarios can be grouped into several distinct categories. Second, several factors exist which can affect scenario selection.

The nature of validation scenarios can be classified as one of the following types:

- 1) **BASICS** -- classical scenarios with "textbook" basic analytical solutions, but which usually are too simple to match the actual situation,
- 2) **CODE** -- scenarios which have been executed on identical subject matter software that has already been verified and validated.
- 3) **TESTS** -- test results from an instrumented experimental facility,
- 4) **PLANT** -- actual measured data from operating nuclear power plants,

- 5) LICENSING -- scenarios required or recommended by licensing regulations or guidelines, and
- 6) REGRESSION -- cases that have been run on other previously validated and accepted software with the same function including previous versions of the software being validated. They are composed of any of the above five types and are not truly a separate type.

In the context of this project, BASICS scenarios are defined as those validation cases which are derived by the theoretical solution of first principles equations or relationships which accurately describe physical processes. Since first principles are generally not directly applicable to nuclear power plant modeling, BASICS scenarios are assumed to be those scenarios which make approximations or simplifications to either the real plant situation or first principles so as to predict specific nuclear power plant behavior.

CODE scenarios represent the use of results from scenarios executed on another system which has been verified and validated and is generally accepted in the nuclear industry as a tool for performing the same functions as the software being validated. It should be noted that the system which has been verified and validated may not necessarily have received the nature and extent of V&V that is presented in this project. In addition, the acceptance of the system by the nuclear industry is not meant to imply that it has been officially accepted by the USNRC, but rather that subject matter experts in the nuclear industry generally accept the use of this particular software for performing certain analyses or functions related to nuclear power plants.

Usually, the already verified and validated identical subject matter system utilizes a different methodology to calculate the results. Thus, some differences between the two systems scenario results are to be expected, but should be explainable in terms of the difference in solution techniques. CODE scenarios are important in that they demonstrate a system's capabilities in terms of another known and accepted system. They are limited by the fact that the system being compared to may still have some errors or other shortcomings and is, in itself, only a means of modeling physical phenomena.

TEST scenarios are most often based on facilities which are designed specifically to provide data for phenomena assessment. These facilities have extensive calibrated instrumentation, data acquisition systems, and other design features intended to measure a wide range of parameters of interest in establishing validation scenarios. One disadvantage of TEST scenarios is instrumentation fault, inaccuracy, or drift. Another problem with TEST scenarios is the fact that these costly facilities are almost always constructed on a reduced scale with other design compromises when compared to actual operating nuclear power plants. Thus, even a good comparison between the software predictions and experimental facilities leads to questions as to the adequacy of the software in modeling full scale nuclear power plant behavior.

REGRESSION cases provide assurance that previous scenarios can be re-executed by a later code version with similar results when compared to the previous code version. Regression scenarios are actually made up of some combination of the other five types of scenarios. The user is familiar with these scenarios and their expected output. Thus, they represent a known quantity for which the software has demonstrated acceptable performance. The limitation of REGRESSION scenarios is that they are based on assumptions about the software capability and, in some cases, reflect an outdated state of understanding of the subject matter. For example, in PWR (pressurized water reactor) fuel management, a ten year old computer code used in designing a new fuel cycle may not have any scenarios that include integral burnable poison in the fuel rods since that technology was not in use, but today it is available and used in some plants. Therefore, the regression set of scenarios would not test this particular capability of the software.

PLANT scenarios are generally viewed as one of the most important means of validating software since the data does not represent any compromise from the actual expected nuclear plant's behavior. Certain tests and measurements are routinely performed or required for operating nuclear power plants. These include a wide range of startup tests involving pump flow, turbine-generator load, core power distribution, and reactivity coefficients. During operation, the core power distribution is periodically monitored as well as key thermal-hydraulic parameters which are inputs to technical specifications and protection systems. This data, with its proper qualification, is a valuable source of validation scenarios.

The disadvantages of PLANT scenarios include the availability of useful data for some software. A particular type of software may involve a technical area that is not directly measured by instrumentation. For example, nuclear fuel rod fault can be analyzed and predicted by some computer codes, but there is no direct way to measure either if a specific fuel rod has

failed or any details of its fault mechanism while it is operating. Only costly laboratory analyses can provide that type of data. Another negative aspect of plant data is the very nature of the data. In real world operating situations, instruments fail, lose accuracy and precision, drift. Data collection may have gaps or unexplained oscillations and the exact status of equipment may not always be known. Documentation of plant data can vary greatly in both quantity and quality. Also, operator actions may occur which influence the usefulness of the data. An additional limitation is that PLANT data is usually only available for a limited range of the situations (i.e. normal operating conditions) needed to validate the software. All these factors need to be accounted for in order to use plant scenarios for useful software validation.

LICENSING scenarios are only applicable to validation of software which involves disciplines directly under the auspices of specific USNRC regulations or guidelines such as the Standard Review Plan (NUREG-0800) or the Code of Federal Regulations. In such cases, regulations specify scenarios that need to be analyzed. A prudent validation would include some of these regulatory scenarios to demonstrate both code capability and user understanding of nuclear power plant modeling and behavior. Since licensing scenarios are conservative and not realistic, the only point of comparison for the software results would be the results of other software which has been accepted by the USNRC (e.g., Nuclear Steam Supply System (NSSS) and fuel vendor software for reload safety analyses). Thus LICENSING validation scenarios are important for that subset of software which are within the purvey of regulations that delineate scenarios.

As the aforementioned analysis indicates, each of the six types of validation scenarios has its merits and drawbacks or limitations. Factors that will affect the type (or mix) of validation scenario include:

- 1) Applicability to specific software;
- 2) Availability of plant or experiment scenario data;
- 3) Software V&V class (as defined in Volume 5);
- 4) Presence of previous code version validation scenarios; and
- 5) Existence of identical subject matter validated software.

The first factor of specific software applicability affects the relevance of LICENSING scenarios since, if the software does not fall under the auspices of any specific regulatory guidance that sets scenarios, this type is not applicable. The second factor of data availability will dictate if plant or experimental facility scenarios can be used for validation. Software complexity and degree of required integrity, the third factor, should be used as a guide in selecting the range of validation scenarios. A code that deals with one parameter (e.g., pump vibration implied performance) would require fewer scenarios than a complex code with a wider range of application (e.g., pump and connected piping system transient behavior to postulated accidents). If other already validated and accepted software exists which analyzes the same subject matter, comparison to its validation scenarios represents a valid component of the validation program. Finally, the fifth factor specifically addresses the case where the software being validated is a modification or revision of existing validated software. In this case, the regression suite of validation cases from the previous version is a logical set of scenarios which may or may not be further augmented by other cases.

6.3 Methodology For Selecting Scenarios

It should be noted that the guidelines discussed in this section were derived by a combination of an assessment of the information gathered for this task and the technical judgement of the authors. While considerable information and a high degree of confidence exist regarding V&V Class 2 and 3 system validation scenarios, there is less information and confidence on the guidelines for V&V Class 1 software.

Validation scenarios are typically the final means by which software is shown to implement the approved design. A good set of scenarios must consider the range of application for the software and a comparison to irrefutable data whether it is in the form of plant or experimental facility measurements, other accepted software results, or basic first principles.

For validation scenario selection, the software must first be classified so that the relevance of each type of scenario which is discussed in Section 6.2 can be evaluated. The overriding consideration is applicability. If the nature of the software is such that no basic first principles solutions apply, then this type of scenario cannot be used. The same rationale will determine whether a software's validation can utilize experimental, plant, or licensing data. Regression tests are only of value if the software represents a modified version of an already validated code or if already validated software with the same function exists with its own scenarios.

6.3.1 Validation Scenarios for New Software

After classifying the software to ascertain which types of scenarios can be used, a hierarchy of scenario types is applied. Within certain boundaries and data limitations, the following order of importance of validation scenarios is delineated for NEW software (in descending order of importance):

- 1) PLANT
- 2) TEST
- 3) BASICS
- 4) CODE
- 5) LICENSING

The desirability of PLANT data for scenarios is directly related to the quality of the data. This includes detailed knowledge of the plant configuration and design, initial conditions, instrumentation accuracy, location, and precision, and operator actions. Without highly qualified plant data, this type of validation scenario is still useful, but subject to engineering judgment and limited applicability.

The same range and level of detail of information is needed for experimental facility TEST information. If the test was not properly instrumented or instrumented failed or exhibited unexplained erratic behavior, the data is still useful, but subject to engineering judgement and limited applicability.

BASICS first principle validation scenarios are useful only in evaluation the same level of sophistication in the software. They are important in looking at small parts of the software rather than the entire code, but they can also be useful in validating overall trends without considering the veracity of specific numerical parameter values.

CODE scenarios offer an ability to validate system functions or capabilities that can not be confirmed with PLANT, TEST, or BASIC scenarios. The strength of this scenario type is based on the perceived fidelity of the other software system which it is being compared to in terms of its own V&V and acceptability in the nuclear industry. The weakness of CODE scenarios is the fact that they represent a software-to-software comparison rather than a comparison to measured real world data or well known first principles solutions.

The value of LICENSING scenarios must be taken within the context of the results being available from another computer code or analytical methodology that has been validated and approved for the same application.

6.3.2 Validation Scenarios for Modified Software

To ascertain if software is modified and not new, engineering judgment is required in evaluating the nature and extent of the changes in capability of the software. For MODIFIED software which does not represent a major change to the code, the following order of importance for validation scenario types is applicable:

- 1) REGRESSION
- 2) PLANT
- 3) TEST
- 4) BASICS
- 5) CODE
- 6) LICENSING

REGRESSION scenarios are valuable as long as the modified software version does not represent a large deviation from the previous version's capabilities. Major modifications will require further validation beyond the last version's set of scenarios, accepted software which performs the same function, but uses different methods. Another interpretation of Regression scenarios is from already validated and accepted software which performs the same function, but uses different methods.

For this list, it is presumed that the regression set of scenarios includes a previously acceptable mix of plant, test, and licensing scenarios as they apply to the application.

6.3.3 Selection Philosophy

The underlying philosophy for selecting validation scenarios, within the constraints of well qualified data, is to place the highest importance to real world measurements either at operating nuclear power plants or experimental test facilities. Since the ultimate application of the software is to support the operation of nuclear plants, validation of their behavior is paramount. The desirability for these scenario types, however, must be tempered with the knowledge that such data may not exist or be sufficiently qualified. Also, some software applications are not based on best estimate modeling, but rather licensing bases. In these cases, licensing scenarios constitute a good source of validation scenarios. BASIC first principles scenarios have a place in the mix of validation scenarios in that they can be used to prove individual parts of the system and validate trends. Finally, REGRESSION cases, which may actually consist of some mix of the previously discussed five types, are of the highest importance for modified versions of existing software, but play a smaller role for completely new software.

The guidelines for selecting validation scenarios are based on the key concepts described in Section 6.2: type of scenario and factors affecting scenario selection. This information was used to develop a guideline procedure in the same style and format as those provided in Volume Five. Following this procedure will result in knowing what types of validation scenarios to use and their range of application. The actual selection of specific test cases, however, requires the assistance of an engineer (or similarly experienced and trained professional) to pick the specific test-case parameters appropriate for the system context.

6.4 Methodology Applied To Systems A and B

In determining validation scenarios for System A, which is new software, the guideline procedure was followed resulting in a minimum of 12 to 20 scenarios with a mix of qualified plant and licensing cases. The selection of licensing cases is somewhat subjective since there are no specific regulatory guidelines on emergency operating procedure tracking systems, but there are guidelines on emergency operating procedures and the nature of abnormal plant events which should be covered by these procedures. The next step in this process was to identify the specific scenarios for validation of System A from the available data on plant events and licensing guidance. The availability of fully documented qualified plant tests applicable to System A is small.

For System A PLANT validation scenarios, the obvious source of data for a utility would be its own nuclear power plant's data files for startup tests and unusual occurrences. Each nuclear power plant performs extensive tests during its initial startup for commercial operation to ensure that all important systems and components operate in accordance with their intended design. In addition, during operation, most plants have experienced abnormal conditions which must be reported to the USNRC in accordance with Title 10 Part 50.73 of the Code of Federal Regulations (i.e., 10 CFR 50.73) in the form of a Licensee Event Report (LER). These LERs are analyzed by the USNRC, and if significant from a safety standpoint, the USNRC performs further detailed investigations and evaluations.

Openly available sources of plant data were investigated by examining key USNRC and EPRI documents. The USNRC's Office for Analysis and Evaluation of Operational Data (AEOD) is responsible for analyzing all nuclear events and evaluating their significance for nuclear safety. AEOD issues an annual report (NUREG-1272) which summarizes events at operating nuclear plants in the United States and highlights those events which were deemed significant.

Using Incident Investigation Teams (IITs) and Augmented Inspection Teams (AITs), the USNRC evaluates some events in great detail to determine their causes and broad industry safety implications. Although the USNRC receives a large number of notifications and LERs from nuclear power plant operating utilities, only a very small fraction warrant IIT or AIT

attention. For example, in 1991, AEOD received about 1900 LERs which resulted in one IIT and 15 AITs. The level of detail and importance of an AIT or IIT make documents issued by these USNRC teams a prime source of plant data for validation scenarios. The USNRC Office of Nuclear Reactor Regulation (NRR) maintains a database of all AIT reports since 1985.

In addition to NUREG-1272, AEOD has issued a document that investigates the importance of actual plant events as a precursor to severe accidents (NUREG/CR-4674). This document uses probabilistic risk assessment techniques to evaluate the importance of reported plant events.

Information from NUREG-1272, NUREG/CR-4674, and a listing provided by USNRC-NRR of AIT reports was analyzed to select a subset of reports which could provide real plant data for System A validation scenarios. The following PLANT scenarios and their reference documents resulted from this analysis for System A:

- 1) Peach Bottom Unit 2 Cycle 2 Transient and Stability Tests, EPRI NP-564, June, 1978;
- 2) Excessive Cooldown Rate Event at LaSalle Unit 1, USNRC AEOD Technical Review No. T417, August, 1984;
- 3) Feedwater Oscillations Resulting in Reactor Trip at Dresden Unit 3, USNRC AIT Report No. 8729, October 16, 1987;
- 4) Loss of Offsite Power at Vermont Yankee, USNRC AIT Report No. 9113, June 6, 1991;
- 5) Automatic Depressurization System (ADS) - Reactor Core Isolation Cooling (RCIC) System Interaction Events at River Bend Unit 1, USNRC AEOD Technical Review No. T610, December, 1986; and
- 6) Brown's Ferry Unit 3 Partial Fault to Scram Event, USNRC AEOD Case Study Report No. C001, July, 1980.

These six were considered to be the best PLANT validation scenarios applicable to System A. A minimum of six additional scenarios are suggested according to the guideline procedure which recommends they be sampled first from TEST scenarios, then BASICS, then CODE, and finally LICENSING. No validation scenarios were judged appropriate from the first three of these categories, so the remaining scenarios were selected from the last, LICENSING, category.

For licensing cases applicable to System A, two USNRC documents were consulted. Regulatory Guide (R.G.) 1.70 describes the format and content of Safety Analysis Reports (SARs) for nuclear power plants including a section on accident analysis (Chapter 15 of R.G. 1.70). In addition, NUREG-0800 comprises the USNRC Standard Review Plan (SRP) which includes sections on specific transients in Chapter 15 of the SAR. Both of these regulatory documents discuss the same specific accidents or transients, but from different perspectives and level of detail. The events can be categorized into the following different basic initiating phenomena:

- 1) Increase in heat removal by the secondary system;
- 2) Decrease in heat removal by the secondary system;
- 3) Decrease in reactor coolant system flow rate;
- 4) Reactivity and power distribution anomalies;
- 5) Increase in reactor coolant inventory;
- 6) Decrease in reactor coolant inventory;
- 7) Radioactive release from a subsystem or component; and

8) Anticipated transients without Scram (ATWS).

Each of these categories actually encompass a number of postulated transient events which are described in greater detail in NUREG-0800. Since each category would test different features of System A, a sample of at least one from each group was selected for the LICENSING validation scenarios. The selected scenarios are:

- 1) Steam pressure regulator malfunction or fault that results in increasing steam flow;
- 2) Loss of normal feedwater flow;
- 3) Coincident loss of onsite and offsite alternating current power to the plant;
- 4) Malfunction of the recirculation loop controller that results in decreasing flow rate;
- 5) Control rod drop;
- 6) Inadvertent closure of main steam isolation valves (MSIVs);
- 7) Main steam line break;
- 8) MSIV closure ATWS;
- 9) Loss of feedwater ATWS; and
- 10) Main feedwater line break.

With the above delineated plant events, a total of 16 validation scenarios have been selected for System A.

As with System A, the guideline was used to select validation scenarios for System B with the result that a minimum of 12 to 20 scenarios in a mix of plant, basics, and licensing cases should be used. In selecting PLANT scenarios for System B, the same process described for System A was used. This involved an analysis of openly available documents from the USNRC and EPRI. The only difference was that, whereas for System A, BWR plant events were evaluated, in the case of System B, only PWR events were examined for applicability. The selected PLANT scenarios for System B are:

- 1) Loss of Integrated Control System (ICS) Power and Overcooling Transient at Rancho Seco on December 26, 1985, NUREG-1195, February, 1986;
- 2) Steam Generator Tube Rupture Event at Ginna on January 25, 1982, NUREG-0909, April, 1982;
- 3) Power Operated Relief Valve (PORV) Actuation Resulting in Safety Injection (SI) Actuation at Calvert Cliffs, USNRC AEOD Engineering Evaluation Report No. E320, September, 1983;
- 4) Loss of Main and Auxiliary Feedwater Event at the Davis-Besse Plant on June 9, 1985, NUREG-1154, July, 1985;
- 5) McGuire Overpressurization Event of August 27, 1981, USNRC AEOD Engineering Evaluation Report No. E248, November, 1982;
- 6) Main Feedwater (MFW) Pump Suction Line Rupture at Surry Unit 2 on December 9, 1986, USNRC AIT Report No. 8642, February 10, 1987;
- 7) Loss of Offsite Power and Reactor Trip at Zion Unit 2 on March 21, 1991, USNRC AIT Report No. 91006, April 17, 1991; and

- 8) Steam Generator Boiled Dry Event at Indian Point Unit 2 on January 3, 1988, USNRC AIT Report No. 8803, March 14, 1988.

Suggested BASICS scenarios for System B involve introducing a malfunction of a single plant component or deviation in a single plant parameter that is well understood to affect the status of one of the critical safety functions which is the important output of this system. Variations of the BASICS scenarios listed below could be used, if the variation represented a significantly different validation test of the software. The six CSFs are: reactivity control, reactor coolant system (RCS) inventory control, (RCS) pressure control, RCS transport control, RCS integrity control, and RCS heat sink control. The six BASICS scenarios, all initiated from a steady state 100% power operation state for a four loop PWR, selected for these CSFs are:

- 1) Five control rods stuck out and fault of boron injection;
- 2) Pressurizer level at technical specification maximum level and RCS Charging System operating without letdown or shutoff capability;
- 3) Stuck open PORV resulting in a cooldown rate greater than the cooldown pressure limits;
- 4) Plant trips from full power, three loops in natural circulation, the third loop hot-to-cold leg temperature difference equals 100 °F;
- 5) RCS Pressure rises above technical specification limit; and
- 6) Steam generator A level drops below minimum natural circulation level after plant trip with the remaining three steam generators at above minimum level.

The same USNRC documents applicable to System A are applicable to System B, and the same eight initiating phenomena apply. However, given that System B is intended for PWRs and evaluation of critical safety functions, a different set of scenarios are appropriate. The selected scenarios are:

- 1) Equipment fault that results in increasing main feedwater flow rate;
- 2) Loss of normal feedwater flow;
- 3) Coincident loss of onsite and offsite alternating current power to the plant;
- 4) Inadvertent opening of a steam generator safety valve;
- 5) Control rod ejection;
- 6) Reactor coolant pump shaft break (locked rotor);
- 7) Main steam line break;
- 8) Uncontrolled control rod assembly withdrawal at power;
- 9) Inadvertent operation of high pressure safety injection system during power operation; and
- 10) Steam generator tube rupture.
- 11) Loss of coolant accident (primary coolant pipe guillotine break) at power.
- 12) Loss of load ATWS.

A total of 26 validation scenarios have been recommended for System B which meets the suggested range of 12 to 20 from the guidelines and includes a mix of BASICS, LICENSING, and PLANT scenarios. This number of scenarios was arbitrarily chosen to be greater than 12 to 20 just to provide an example of the methodology for obtaining scenario data for three types of scenarios.

6.5 Novel Method for Knowledge-Based Scenario-Generation

A new method is described in this section which should be extremely cost-effective in that it does not require the system to be actually operating but rather relies on an analysis of the knowledge-base to generate possible scenarios for review by a knowledgeable specialist. It should be noted that this method has neither been completely designed nor tested; its status is that of a potentially promising but untried technique.

Traditional validation scenarios involve dynamic tests of a system -- setting up certain startup and data conditions, and then having the system under test perform in normal operation mode. In contrast, the method proposed here does not actually involve actual operation of the system. Rather, the method uses the knowledge-base component of the system under test (e.g., the rule-base or frames for expert systems, or the set of class and object-definitions for object-oriented systems). The knowledge-base is analyzed extensively by a separate program, and then scenario-like descriptions are generated for evaluation by someone familiar with the application domain.

The descriptions provided by the Knowledge-Base Scenario Generation (KBSG) method are, in their simplest form, a sequence of state-changes that the system could plausibly go through for some starting input-state and subsequent states of data and user input. For example consider a system which is supposed to monitor safety parameters and other data of a nuclear plant and, when there is an emergency, invoke the appropriate Emergency Operating Procedure (EOP) for the operator to follow, showing the state of the plant and the actions appropriate and suggested for that state. The EOPs involved in the system are taken directly from the hard-copy manual EOPs. Under a dynamic test of such a system, the system would be reading data directly from actual plant data-interfaces or from some kind of simulator, and the EOPs, when invoked, would be represented in a carefully-designed user-interface for operator processing. The operator would be given the plant status, the relevant EOP section and suggested simulator action(s), and then asked for a decision. The operator would enter the decision, the system would implement it (or simulate its implementation), and then new data would eventually trigger another aspect of that EOP (or another EOP). This user-system interaction would continue until the plant was completely stable again.

If the KBSG method were used for the above situation, the operator (or some other user) would, generally, be looking at a display in which the initial status of the plant, the data-changes, the EOP, and the recommended actions were all described by a series of text statements rather than the nice user-interface display of the actual system. The system would not actually be running, and the KBSG session could actually be conducted almost on any mini- or personal computer located anywhere. The sequence of states presented to the user would not be determined by an actual plant or plant simulator. Rather, they would be possible states reachable from the prior conditions, and they would be selected according to some kind of simplified programmed heuristic. For example, the heuristic might arbitrarily decide to declare that the plant parameters are such that an EOP "RPV Control Procedure Entry" is entered; subsequently, it might select the plant conditions to invoke the "Primary Containment Control Procedure." Each decision taken by the KBSG would be accompanied by an explanation, stating in words the rules or other knowledge-base elements that justified such a decision. To continue with the example, at this point of +PC ENTRY (Primary Containment section), the EOP embodied in the knowledge base calls for emergency Depressurization actions (e.g., INITIATE HPCS SYSTEM, INITIATE LPCI-C, LOWER SUPPRESSION POOL TEMPERATURE BELOW 35 DEG C, etc.) which would be detailed for review by the user (and selected among if a choice is needed).

It is the user's task during the above KBSG scenario to review each sequential statement of plant-states, EOP invocations, offered explanations, and recommended operator actions and assess whether they are reasonable or not, whether they conform to the hardcopy EOPs, or whether the plant could ever be in such a state described. Anything suspect or deemed unreasonable should be flagged (perhaps by pressing a function key, entering a comment, etc., depending on how the KBSG was set up). This is very much the same task that is required when an actual validation scenario is dynamically executed on a real plant control environment, or on a simulator. The operator viewing the system performance is keeping

a careful eye out for false, inconsistent, non-conforming, or otherwise puzzling outputs or actions of the system. The difference in the two cases is that, during actual execution of a validation scenario, the simulator or the actual plant will control the timing and value of the data parameters, while in the KBSG situation these will be generated or selected by a special program.

In addition, of course, the KBSG user-interface will typically be impoverished relative to the actual system, since it is really not justified to dummy up a realistic user-interface for KBSG trials. However, with KBSG the operator will be given an explanation of each action of the system, according to the relevant rule, where this is typically only an option under normal operating conditions. Also, the KBSG user will explicitly be instructed to examine to see whether all of the IF conditions and THEN actions are necessary or whether any are missing.

The reader should keep in mind that the above-described method has actually neither been fully designed, has not been tested, and that the claims of automatability are therefore not shown. Nevertheless, were such a method as KBSG to be developed and automated, it would appear to have the following strong desirable features:

- It is concerned specifically with assessing the key component of expert systems, the knowledge-base, which is judged to be under-evaluated by conventional V&V methods.
- It does not require a participant to directly examine the system's knowledge-base, which may be very difficult to comprehend for a non-expert; rather, it requires only following the description of a sequence of states and actions, and reasons therefore, with the participant making judgments of reasonability about these. Any application professional can therefore use this method to assess the knowledge-base.
- It does not require dynamic execution of the whole system, usually in the context of an actual larger system or simulator. This avoids the cost and difficulties of scheduling a real validation scenario execution, assembling all the necessary persons to support the run, etc.
- The information provided the participant of the KBSG method is typically much richer and more informative than given a participant of an actual execution run. Therefore, the participant is more likely to detect problems with the KBSG method.
- The method can be run much more quickly, covering dozens or even hundreds more test conditions in the same time that it takes to run one condition in the standard dynamic-testing environment.
- The easy possibility of adding or deleting conditions or actions from rules, or even of introducing changes within these, makes possible a variety of additional formats for assessing potentially problematic rules, such as presenting a multiple choice of rule alternatives and asking participants to select and justify their choice. Such a procedure could not be emulated in the dynamic scenario situation if there is any time-based generation of data, and the procedure could elicit much useful expert information.

For these reasons KBSG holds considerable promise as an additional technique for the testing of AI systems.

7 OTHER V&V ACTIVITIES

In addition to the major activities described in the preceding sections of this document (Sections 2.0-6.0), there were three relatively limited additional activities, which are described here. Two of these activities relate to issues of performing V&V on systems whose knowledge bases are represented in other than IF-THEN rules: i.e. frame-based and object-oriented systems. These activities are described first. The third activity provides a summary of the committees of various professional organizations which are concerned with testing, V&V, or standards for AI systems.

7.1 V&V of Frame-Based Systems

Frame-based systems can be characterized as AI systems whose knowledge bases are represented, in some significant proportion, as frames -- a hierarchically-organized collection of attributes and values which constitute the essential defining characteristics of some concept, whether an entity, an action, or some kind of relation.

7.1.1 Description of Frames

A frame-based system supporting a diagnostic model, say, of the emergency diesel generators for a nuclear plant, would probably have the bulk of its knowledge concerned with the various sub-systems of generators -- electrical, diesel engine, physical mount, and so forth. For each of these sub-systems there would be a top-level frame which provided the highest-level description, connected to a set of more and more detailed frames which described the subordinate elements of each of these sub-systems. In all probability there would be a set of attributes which appeared on all frames -- such as C_ID (the unique identification number of the concept), C_NAME (the name of the concept), C_DESCRIPTION, PARTS (the separate elements which make up the whole), PART_OF (the larger concept that the present one is a part of), C_GENZN (generalization of the concept -- a more general concept), C_SPECZN (specialization of the concept -- more specific concepts for which the present concept is the generalization) and so forth. Other attributes would be much more specific to particular concepts. Many of the values of attributes will be values of a specific type: C_ID may be an integer number; C_NAME a text-string of no more than 20 characters; C_DESCRIPTION, a text-string of no more than 50 words. For other attributes, the values might be constrained to be the NAMES of other concepts: the PARTS attribute will have as values the NAMES of frames which are the various parts of this concept. Similarly the PART_OF attribute will specify the NAME of the concept that this present concept is a part of; likewise, the C_GENZN attribute provides the NAME of a more general concept (for example, GENERATOR may have as a more general concept the C_GENZN-value of ENGINE) and the C_SPECZN attribute will list all the more specialized concepts (e.g., DIESEL_GENERATOR, GAS_GENERATOR).

7.1.2 Errors and Anomalies in Frames

Frame-based systems will have the same other components as other AI systems -- i.e., inference engine, interfaces, and tools/utilities (see Section 2.4). Therefore, the question of V&V of frame-based system is really the question of the V&V of the frames themselves, not the other components.

The expert systems used in the knowledge base certification experiment were both rule-based systems, and a number of rule errors or anomalies were described that could be detected by the syntax and knowledge checking methods used in the experiment (cf. Section 3.2). Similarly, a variety of errors and anomalies (hereafter, jointly referred to as "anomalies") can be generally described for frame-based systems.

In terms of the probable likelihood of occurrence, there are two primary classes of frame-based anomalies: incompleteness and incorrect-syntax (two secondary anomaly classes are described later). Of course, every specific expert system shell (or specialized language) that permits definition of frames will have its own special procedures and may have checks for various types of incomplete or syntactical incorrect forms. The examples of possible anomalies given here are

quite generic and generally suppose that a set of frames is incomplete unless everything contained within is defined in some way.

The incompleteness anomalies are illustrated by the set of frames in Figure 7.1.2-1 which define some age-related concepts concerning people. Here, an **OLD_PERSON** is a specialization (**c_speczn**) of an **ADULT**, based on the criterion that the age is greater than 65 years. Inversely, the generalization, **c_genzn**, of an old person is an adult, the generalization of an adult is a person, the generalization of a person is an animal, and the generalization of an animal is an entity.³ For these set of frames, the name and type attributes, **c_name** and **c_type**, are considered to be essential attributes for all frames. With certain exceptions for the highest frame concepts (those of **ENTITY**, **RELATION**, and **ACTION**), the attribute **c_genzn** which gives the more general concept is also considered to be essential.

It should be noted that any attribute which begins with the characters "c_" is a privileged concept, a so-called "primitive" concept of the frame system which does not need to be defined elsewhere. However, all attributes without such a prefix do need to be defined. Similarly, for these frames, everything that appears as an argument (or value) of an attribute must also be defined elsewhere within the total set of frames in order to be complete. With such a set of frame constraints, the following possibilities for incompleteness anomalies can be identified:

- an essential attribute is missing
- an argument (value) of an attribute does not have its own definitional frame
- a non-primitive attribute appearing on a frame does not have its own definitional frame
- an attribute is missing an essential argument (value)

Incompleteness anomalies have to be checked for with respect to each individual frame. Thus, for the **ADULT** frame, each attribute and its values would be checked for all of the above anomalies. In doing so, one finds that there are two occasions of incompleteness: there are no definitional frames shown for the concepts of **legal_adult** or **gt** (greater than). Every other argument, and non-primitive attribute (e.g., **age**) has its own defining frame, no essential argument is missing, and all essential attributes are present (again, with respect to the **ADULT** frame).

The frames in Figure 7.1.2-1 clearly have a definite consistent style of construction, or *syntax*, as indicated by the figures and the preceding text. Given these, the following possibilities for incorrect-syntax anomalies can be identified:

- an attribute has too many arguments (values)
- a primitive attribute has other than "c_" as a prefix
- a non-primitive attribute has a "c_" prefix
- multiple-values of an argument (e.g., the second argument for **c_speczn**) are not enclosed in brackets
- the arguments of an attribute are not enclosed in parentheses
- there is a space or other character between the attribute-name and the left-parenthesis of the arguments
- the separators between attribute arguments are something other than commas

³ In the syntax of these frames, the argument \$0 represents a local variable for the name of the variable as defined in the first attribute **c_name**. All other attributes on the frames take two or more arguments, and the first argument is always the name of the concept that the attribute applies to, as given by the variable \$0. Arguments which can be a list of values -- e.g., for the **c_speczn** and **c_genzn** primitive attributes -- are enclosed in brackets.

Figure 7.1.2-1 Example of Person-Concept Frames Based on Age

- an attribute is placed other than in a left-justified position on the frame

The above are examples of syntax anomalies for this particular mode of representing frames. Every language or shell that provides for the representation of frames will have its own set of stylistic writing principles, or syntactic constraints, and each way there could be a violation of these constitutes an example of an *incorrect-syntax* anomaly for that system. These anomalies would replace those listed above, but the class of incorrect-syntax would still be a major class of anomalies. Looking at the **ADULT** frame again, there appear to be no examples of any of the above syntax anomalies.

In addition to the major classes of incompleteness and incorrect-syntax frame anomalies, there are two secondary anomaly classes whose frequency of occurrence is expected to be less than the first two types described above: *inconsistency* and *name-conflict*. There are two sub-classes of *inconsistency* anomalies -- those that occur within the set of defining concept frames, so-called *class inconsistencies*, and those that occur between a defining concept frame and a particular instance of that frame -- *instance inconsistencies*. This distinction can be clarified by referring again to Figure 7.1.2-1. All of the frames shown in this figure are *class* frames, since they define general concepts, not particular examples of those concepts. The concept of an **ADULT**, for example, is defined in a rather impoverished manner in terms of the age of a person being greater than 18 years. Now, suppose this frame is used to describe real-world circumstances, specifically a 17-year old person "Harry." A special *instance* frame is created which has all of the primitive "c_" attributes copied onto it, plus the single additional attribute for *age*. However, instead of specifying the general relation of greater than 18 years, the value of *age* on the *instance* frame for **ADULT** just takes a single numerical value that is, in fact, Harry's age (17), so that its frame would look as follows:

Instance(adult): HARRY

c_name(\$0=adult)
c_type(\$0,entity)
c_genzn(\$0,[person])
c_speczn(\$0,[legal_adult, old_person])

age(17)

Given this instance of **ADULT** there is an *instance inconsistency* between the instance and the defining class, since Harry's instance frame specifies via the *c_name* attribute that he is an adult but his age, 17, is not greater than 18 as required in the class, or concept, frame.

An inconsistency among classes occurs whenever there is a discrepancy between any two class, or concept, frames. For example, consider the frames of **ADULT** and **OLD_PERSON**, as shown in Figure 7.1.2-1. The *age* attribute of **OLD_PERSON** is now consistent with that of **ADULT**, since 65 or more years is indeed greater than 18 years. Suppose, however, that the values in the *age* attribute of the **OLD_PERSON** frame were changed as shown by the four examples below:

age(\$0, GT(ANIMAL, years))	(Ex. 1)
age(\$0, LT('65', years))	(Ex. 2)
age(\$0, EQ(DIVIDE('100','10'), years))	(Ex. 3)
age(\$0, GT('65', pounds))	(Ex. 4)

Although **GT** is not shown on Figure 7.1.2-1, its definitional frame would be something like the following:

The first four primitive attributes are familiar. The fifth specifies that the concept **GT** has two arguments, as indicated by the local variables \$1 and \$2. The sixth attribute, **c_class**, is used to define restrictions on arguments, specifying in this line that the first argument of **GT** has to come from the class of **number**. The last attribute specifies that the second argument of **GT** will be a **time_unit**; in particular, it will be one of the specializations of **time_unit** that has no further specializations, the so-called "leaves" of the time-unit hierarchy.

Given this definition of the concept **GT**, then, the *class inconsistencies* of each of the four examples can be explained. In the first example, the first argument of the concept **GT** is not a number, but another type of concept, thus violating the class restriction for the first, \$1, argument of **GT**. One doesn't know this immediately but has to refer to the frame of **ANIMAL** and learn that the defining attributes make it other than a number. The second example illustrates a partial range problem, in that for numbers less than 65 down to 19, the definition of **OLD_PERSON** is in fact consistent with the constraints of its generalization. However, there are 18 positive integers for which this definition would be inconsistent (for age from 1-18). In the third example, an **OLD_PERSON** is defined as being one specific age, equal to the result of the division of 100 by 10. Only when this division is actually carried out, with 10 as the result, can it be recognized as a violation of the **ADULT** age restriction of >18 years. Finally, in the fourth example, **pounds** has been substituted for **years** as the unit of measure. Checking the definitional frame of **pounds** would reveal that **time_unit** is nowhere on its **c_genzn** path, and therefore that the use of **pounds** is inconsistent.

There could be many other examples of *class inconsistencies* for **GT** by violating the defining characteristics of **GT** or **AGE** in other ways, but these examples should suffice to illustrate this type of anomaly. Note that the detection of class inconsistencies is a much broader problem than the detection of instance inconsistencies since every attribute and value of every frame must have their uses in frames compared against their definitions. Of course, class inconsistencies are possible only to the extent that frame elements are themselves defined somehow.

The fourth anomaly class of *name conflict* refers to two situations: (1) when two frames (either instance or class) have the same name but different defining attributes and values, and (2) when two frames have exactly the same defining attributes and values but have different names.

These four types of frame anomalies are summarized in Table 7.1.2-1.

7.1.3 Detecting Frame Anomalies

As discussed in the survey of V&V methods (Section 2.0), there are two primary classes: static analysis techniques, and dynamic testing techniques. The former methods examine the system artifacts (e.g., requirements and design documents, source code) directly, while the latter methods are appropriate for implemented systems only and involve actual execution

of the software on some operating system on some computer hardware platform. Static methods, it was argued, are much cheaper and arguably much more effective at finding problems than dynamic methods. In particular, static methods were judged to be especially applicable to the knowledge base in general, and this finding holds specifically for all of the anomaly classes of frame-based representations. That is, static analysis methods can be envisioned which could check for all of the identified types of problems.

In actual practice, there are no existing shells or languages which embody procedures which check for all anomaly classes, although a number do check for a variety of *incorrect syntax* anomalies. Nonetheless, tools like *VERITE'* based on the AI language PROLOG (cf. Preece, 1991) could readily be extended to implement detection of *incompleteness*, *inconsistent*, and *name-conflict* anomalies.

7.2 V&V of Object-Oriented Systems

Object-oriented systems, as discussed elsewhere (Volume 5, Section 2.2.2) are similar to frame-based systems in two important respects: (1) they both have *tools and utilities* and *interfaces* as two system components, and (2) the class and object hierarchies of object-oriented systems are very similar to a hierarchically-organized set of frames (with generic frames being the equivalent of classes, and frame instances being the equivalent of objects). That part of class and object definitions which deals with the description of their defining characteristics will be quite similar in structure to the frame examples given previously (e.g., Figure 7.1.2-1). Instead of an inference engine, however, object-oriented systems have *methods* and *message-passing*, conventional software procedures for executing any arbitrary function and a communication-method for activating these methods, respectively.

The majority of components of object-oriented systems -- methods, message-passing mechanisms, tools and utilities, and interfaces -- are typically written in conventional procedural programming languages involving conventional programming data structures, and therefore these can be tested, validated, and verified using the conventional V&V methods described earlier in this report. The final component of object-oriented systems, the class and object hierarchies, are the focus of concern here.

The work is described in three sections which describe: (1) the approach taken to address the problem of V&V of the knowledge base component of object-oriented systems, the class and object hierarchies; (2) the identification of potential anomalies in these structures; and (3) how the anomalies might be detected.

7.2.1 Approach to the Problem of V&V of Class/Object Hierarchies

The V&V approach taken is the same as that for frame-based systems, namely to enumerate all of the specific failure that can be imagined to occur, and then to identify specific V&V techniques to detect each of the failure. This approach was taken for the development of the set of recommended V&V methods for particular Life-cycle phases, Class of V&V, and system component, as discussed previously in Section 4.0 (and Volume 5). This approach has been called *Fault-Based Verification* and has been proposed as an appropriate way to achieve ultra-high reliability for all types of software (cf. Miller & Hayes, 1993).

The compilation of a comprehensive set of failure began with reviews of a number of recent texts on object-oriented analysis, design, and programming for examples or ideas of problems, errors and anomalies (Korson, 1990, was particularly useful; see the references in Section 7.4). In addition, the CLOS programming standard for LISP (Common LISP Object Standard; e.g., Steele, 1990) was reviewed to determine what kinds of failure might be specific to programming conventions or standards. To this set was added those which, on analytical review of class and object hierarchies, would seem to provide additional types of questionable structure characteristics. None of the sources consulted (nor any of the many articles on object-oriented programming read by the authors) dealt other than obliquely with failure. There was even less direct concern for the V&V of object-oriented systems.

Table 7.1.2-1 Summary of frame-based anomalies

Anomaly Type	Description
Incompleteness	A frame is missing necessary elements (such as a required attribute or a needed argument) or some non-primitive element on a frame (attributes or values) are undefined elsewhere.
Incorrect Syntax	There is some deviation from the permitted, well-defined manner of constructing frames or attributes and values. These deviations can be in the form of incorrect spellings, improper or missing use of punctuation and symbols, unrecognized structures or symbols, and unallowed permutations of required elements. These anomalies do not at all depend upon the meaning or logical content of the frame.
Inconsistency	These anomalies depend on a mismatch between two frames (or frame elements) in terms of their meaning or logical contents. <i>Instance Inconsistencies</i> are conflicts between instances of a concept frame and the concept frame itself. <i>Class Inconsistencies</i> are conflicts between two or more concept frames (or their elements).
Name Conflict	This anomaly covers two situations: (1) calling different things by the same name (when two frames have the same name but different defining elements), and (2) calling the same thing by different names (when two frames have the same defining attributes and values but have different names).

7.2.2 Failure of Object-Oriented Class/Object Hierarchies

A total of 32 failures were identified, and their names and descriptions are given in Table 7.2.2-1. These failures are intended to be generic and not characteristic of any particular object-oriented product or approach. At least some of the naming and incompleteness types of anomalies identified for frame-based systems (cf. Table 7.1.2-1) are included. This compilation of failures for object-oriented systems is the first known of its kind and is considered here as a precursor to the selection and development of effective techniques for assuring the quality of object-oriented systems.

The two types of incompleteness anomalies would certainly also be characteristic of object-oriented systems. Finally, the incorrect-syntax anomalies would similarly apply to object-oriented systems, but they would be specific to a particular language or development-shell product.

This set of failures is almost certainly incomplete; an extended effort would surely produce a number more. However, the set is believed to be rather comprehensive and representative of the types of problems and anomalies one would encounter with the class and object hierarchies of object-oriented systems.

7.2.3 Detecting Object-Oriented Failure

The approach recommended for detecting object-oriented problems is exactly the same as that suggested for frame-based anomalies previously (Section 7.1.3), namely automated static-analysis procedures implemented in a generic knowledge-base checking tool like VERITE' (cf. Preece, 1991).

7.3 V&V Activities in Professional Organizations⁴

There are three professional organizations who have activities or standing committees which relate to the V&V of AI systems:

- American Association for the Advancement of Artificial Intelligence (AAAI),
- American Institute of Aeronautics and Astronautics (AIAA), and
- Institute of Electrical and Electronics Engineers (IEEE).

In addition to these organizations there are four international conferences (usually held in Europe) which sponsor special sessions dedicated to the V&V of AI systems:

- International Joint Conference on Artificial Intelligence (IJCAI),
- International World Congress on Expert Systems,
- European Conference on Artificial Intelligence (ECAI), and
- European Symposium on Verification and Validation of Knowledge Based Systems (EROVAV).

Finally, there is an international INTERNET bulletin board on Verification and Validation of AI systems called VAVTALK on which professionals of these organizations and other institutions, exchange views.

⁴ The first author has been active, in one way or another, with almost all of the activities and organizations discussed.

Table 7.2.2-1 Name and description of 32 object-oriented class/object hierarchy faults

Name	Description
Method Inherited Twice	<p>The same method is inherited from different classes along at least 2 different paths in a class that inherits from multiple parent classes.</p> <p>For languages that support this, it is deterministic but potentially confusing. To know what code is actually invoked, one must know the specifics of conflict resolution for the Object Oriented (OO) language being used.</p> <p>This is somewhat similar issues of name-space shadowing that occur in other types of computer languages (e.g., name scoping rules).</p>
Instance Variable Inherited Twice	<p>The same instance variable is inherited through at least 2 different paths in a class that inherits from multiple parent classes.</p> <p>For languages that support this, it is deterministic but potentially confusing. To know what code is actually invoked, one must know the specifics of conflict resolution for the OO language being used.</p> <p>This is somewhat similar issues of name-space shadowing that occur in other types of computer languages (e.g., name scoping rules).</p>
Class Contains No Instances	<p>The class is never instantiated. This is quite reasonable if the class belongs to a class library that was not explicitly created for the current application. Also, some classes are only meant to be used as a source for derived classes. (in this case we might continue on down the class graph looking for instantiation of the sub-classes).</p> <p>This "anomaly" might occur quite a bit. It probably shouldn't be reported every time. Rather, it could contribute to other evidence that there is a problem. Only when several anomalies coincide with this do we report it.</p> <p>There should also be an exclude/include list mechanism to decide what classes to ignore or look at specifically. This is similar to a type being defined without having a variable being created of that type.</p>
Method Not Used	<p>A method of a class is not used anywhere in the system (see also "Class contains no instances").</p> <p>Similar to a procedure being defined but not called.</p>
Instance Not Used	<p>Similar to unused variable (declared but not referenced).</p>

Table 7.2.2-1 (Continued)

Name	Description
Excessively Broad Class Graph	The maximum number of sub-classes for any particular class is above some (user) defined threshold.
Excessively Deep Class Graph	The number of classes between a top-level class and a bottom-level class is above some defined threshold. Only significant if the path is also narrow. In other words, a inheritance path that has many "generations" but not many "siblings" is probably an indication of a poor choice in class hierarchy design.
Excessively Disjointed Class Graph	There are a large number (# greater than defined threshold) of independent (i.e., unconnected) subgraphs in the class graph.
Excessively Unbalanced Class Graph	Some classes have radically more sub-classes than others, as defined by a user-set threshold.
Possible Implicit Class Abstraction	Similar class subgraphs exist in different places in the total class graph. Two classes are similar if they have similar sets of parents and/or children that contain the same set of methods.
Circular Class Graph (Cycles)	Class inheritance path contains cycles. Roughly equivalent to an infinite loop.
Over Specialization of Sub-Class	A class shadows most of the methods and instance variables of at least one of its parents. This means there are a lot methods and instances variables that are named the same as in a parent class but act differently.
Peer to Peer Specialization	A class contains a strictly greater specialization then its peer. The only difference between the specializations of two peers is that one contains a higher degree of specialization than the other.
Overlapping Subclasses	Two or more classes have a user-defined degree of overlap in terms of common subclasses. This is not necessarily an error.
Incomplete (Non-Exhaustive) Specialization	A class contains most of a finite/complete set of specializations, but not all. Determining the full set up possible specializations probably requires some design information.

Table 7.2.2-1 (Continued)

Name	Description
Semantically Related Classes Not Close in Class Graph	Classes having an overlap or high similarity in attributes and values, and/or subclasses, are in distal parts of the class graph.
Excessively Large Number of Methods in a Class	Classes with more than a user-defined number of methods (recommended threshold is in range 20-24).
Excessively Large Numbers of Instance Variables in a Class	Number of instance variables in a class exceeds a user-defined threshold.
Excessively Long Methods	Methods which contain more than a user-set threshold of executable statements (e.g., 750).
A Non-Public Class Operator Was Defined	An operator is defined in the non-public part of an object but is used as a public class operator.
Public Operator Not Used by Any Class Instance	An operator was defined for use by class instances but was never used.
Class Contains Non-Local Operator	Class contains an operator that doesn't access or modify any data in the class.
Excessive Number of Parent Classes	A class has too many parents. A good design guideline is to link a class with as few other classes as possible (e.g., 2-4).
Implicit Class to Class Communication	Information is passed from one class to another through a global variable.
Inheritance Used to Develop the Representation of a Class	A subclass was created that used inheritance for representation rather than specification purposes.

Table 7.2.2-1 (Continued)

Name	Description
Class Inherits from Parents of Different Abstraction Levels	Class inherits from parents that are not of the same level in the class hierarchy.
Different Length Multiple Inheritance Paths	<p>There exists 2 paths of different lengths between a class and one of its ancestor classes. This suggests a poor design.</p> <p>EXAMPLE:</p> <p>A depends on B and C; B depends on C; therefore, A depends in one case on B and, in another case on B and C.</p> <pre> graph TD A --> B B --> C A --> C </pre>
Instance Variables, Methods, or an Instances Variable and a Method With the Same Name	Any two or more of these entities having the same name is an anomalous name-conflict situation.
Similar Classes with Different Names	Having "similar" class definitions means having the same (or very similar) number of attributes of each data type & the same number of attached methods. Also, the attributes & methods have the same or similar names.
Instances/Classes have Similar Names	Using the similar name pattern-matching routine described above, find two class names or two object instances or a class and instance with very similar names, that could lead to confusion by programmers further on.
Similar Names in Single Class	Two names (instance variables and/or methods) with similar names in the same object definition or in different object definitions where name can be referenced directly without referring to their object.
Instance Variable/Method Names Have Similar Names	Two names (instance variables and/or methods) are similar in the same object definition or in different object definitions where the names can be referenced directly without referring to their object.

7.3.1 IEEE Activities

There are two working groups and three IEEE standing committees that are of special interest. The working groups are:

- The ANSI/IEEE Standard 1012 Working Group, and
- The Nuclear Power Engineering Committee P-7-4.3.2 Working Group.

The ANSI/IEEE Standard 1012 is concerned with Software Verification and Validation Plans. This standard does not explicitly concern itself with AI systems software. However, the committee has heard presentations from several members of the AI community discussing problems. Of particular relevance to the present project is the committee's attention to the problem of determining a basis for recommending various levels of V&V as a function of a system's intended use and its characteristics. At the present time, their approach does delineate multiple classes of V&V stringency recommended, but only on the basis of the potential destructiveness or safety impact of the system. In contrast, the approach proposed here takes into account this factor and also the factor of system complexity.

The Nuclear Power Engineering Committee of the IEEE Power Engineering Society has developed the standard P-7-4.3.2-1993, Standard Criteria For Digital Computers In Safety Systems of Nuclear Power Generating Stations. This is the only standard which relates to digital controls in nuclear settings. As such, it can be expected to have an important impact on software development activities associated with the introduction of digital instruments and controls and their attendant software, including AI systems. It proposes a number of software V&V activities with an emphasis on identification and resolution of Abnormal Conditions and Events (ACEs). The discussion of software methods is quite general and not at the level of specific guidelines. The allocation of logical system function is to non-computer hardware, computer hardware, and computer software. From the point of view of AI systems, which predominantly are decision-support systems with many decision functions specifically to be accomplished by the operator or user, a more desirable approach is one which explicitly allocates function to the human user. Direct allocation to human users will insure that the requirements for human processing will be dealt with directly, on an equal footing with software, in the important processes of requirements analysis and tracing, design, fault/hazard analyses, configuration management, testing, and quality assurance.

The first of the three relevant IEEE committees of relevance to this project is P1232, the Artificial Intelligence Subcommittee of IEEE Standards Coordinating Committee-20, on Expert System Standards for Automatic Test Equipment. The key activity of this committee which is of interest to the present guideline activity is its goal to define a generic expert system model for use with test equipment. This definition would include the following activities:

- Identification of critical knowledge and types of knowledge,
- Analysis of knowledge representation requirements, and
- Determination of expert system requirements relating to advisor/assistant and command/control/functions.

These determinations could result in guidelines or standards concerning the development of test equipment AI systems and therefore have direct consequences for the V&V of such systems.

The second IEEE committee of interest is P1252 whose goal is to develop a standard for knowledge representation based on frame-based reasoning and standard languages and operating systems such as C, C++, POSIX, etc. Recommendations to IEEE concerning representation standards have not yet been made. The dynamic nature of this field makes it unlikely that standards, or even guidelines, will be accepted in the near term.

The third IEEE committee of interest is the Technical Committee on Expert System Applications formed under the auspices of the IEEE Computer Society. The objective of this committee is:

- To provide a common vocabulary for addressing issues of expert system V&V including testing;
- To encourage, support, and, as necessary, sponsor development of guidelines, recommended practices, and standards; and
- To communicate insights with other industry, professional, or regulatory organizations.

The primary activity of this committee appears to be in cooperative development of an AI Glossary (with the IEEE Computer Dictionary effort) towards the goal of adoption as IEEE Standard 610.8. However, it is clear that all of these activities are directly relevant to the present guideline effort.

7.3.2 AIAA Activities

The AIAA has an AI committee on standards which has recently (July, 1993) become active again, with a committee of persons active in the field of V&V of AI systems. The goal of this committee is to develop recommendations, beginning with guidelines, which will assist in the development of high-integrity AI systems as embedded in or interacting with conventional software systems. A high degree of collaboration with similar standards groups is envisioned, and a priority is to ensure the relevance of the committee's activities to the initiative for a National Information Infrastructure, in which initiative software is a key component.

7.3.3 AAAI Activities

The AAAI is the primary organization for AI activities and sponsors yearly a workshop on the V&V of AI Systems which attracts most of the active professionals in this field, both nationally and internationally. The workshop was begun in 1987 and can be seen as the key instrument in the establishment of V&V of AI systems as a defined field and as the progenitor of most of the related conferences. New insights and approaches to the problems of this area are typically first given as presentations in the yearly summer workshops. It is an organization closely to be monitored for impacts on the thinking underlying the present guidelines.

7.3.4 International Conferences

The international conferences have been monitored by discussions with participants, and by access to the papers and books resulting from the conferences. The focus of these conferences is almost entirely on the knowledge base component of AI systems, particularly rule-based representations, with a great emphasis on formal approaches. There is not at all a strong concern for relating to conventional V&V approaches or to principles of conventional software engineering and development.

8 SUMMARY AND CONCLUSIONS

Artificial Intelligence (AI) software is becoming more prevalent in the nuclear power industry because it performs a unique function of offering expert subject matter knowledge that is directly applicable to the design, maintenance, and operation of nuclear power plants. Currently, there are no verification and validation (V&V) guidelines for AI in the nuclear power industry. Recognizing this situation, the U.S. Nuclear Regulatory Commission (USNRC) and the Electric Power Research Institute (EPRI) jointly funded a three year project with Science Applications International Corporation (SAIC) to develop and document ES V&V guidelines for the nuclear power industry.

This project consisted of ten interrelated tasks:

- Survey and assessment of conventional software V&V methods;
- Survey and documentation of expert system V&V methods;
- Test applicability and limitations of conventional software verification methods;
- Develop a verification methodology for nuclear power plant applications;
- Test the developed verification methodology;
- Development of a knowledge base (KB) certification methodology;
- Evaluation of verification methods;
- Develop and evaluate validation scenario guidelines;
- Present a conference paper and publish a journal article; and
- User manual and final technical report.

Taken together, these ten tasks represent the most comprehensive compilation and analysis of AI V&V that has been undertaken in both the nuclear and computer software industries. The documentation for each task has received exhaustive review by numerous members of the technical and management staff of both USNRC and EPRI as well as other experts in the AI and V&V field. Many activities in this project utilized extensive information from the U.S. and international nuclear and non-nuclear commercial industry as well as national laboratories through the use of literature surveys, site visits, and telephone contacts. One activity (see Section 7) involved direct participation of representatives from three nuclear power plant utilities, a national laboratory, a university, and the USNRC. It should also be noted that significant innovative work was required in order to fulfill some of the unique V&V requirements of AI systems.

The major accomplishment of this project was its central goal -- the development of a set of V&V guidelines to be used to assure the quality of expert systems, and other AI systems, developed for the nuclear utility industry. These guidelines provide recommended information about the following:

- What issues are important in deciding how much V&V to perform;
- When V&V may be performed;
- What components (as well as the overall system) it may be performed on;
- For specific combinations of the above, what set of V&V methods are suggested;
- In what order the methods should be applied;
- Who should perform the investigation, with what qualifications;
- What metrics should be gathered as a result of applying the V&V methods;
- What decisions should be made, and what should the decision thresholds be;
- And finally, for 11 of the more important techniques, specific procedures laying out step by step how to apply the method.

All this information is provided in a form directly usable by a utility or other developer. In addition, the recommendations provide a roadmap concerning what V&V activities are believed appropriate for what circumstances, and what the decisions might best be based on.

It is critical to stress that the recommended guidelines are just that -- recommendations and guidance. While many professionals have reviewed various aspects of this work, with their recommendations incorporated, the guidelines have not yet been tested and validated in the field. Nor have professional bodies from either the utility or the regulator side been convened with the explicit charter of assessing whether the present recommendations should be adopted as reasonable workable guides. Until these two evaluations occur, the status of these guidelines is indeterminant. They are clearly the product of a most careful and reasoned development program, but judgment of the involved parties is necessary to truly validate their worth.

While expert systems were the nominal type of subject software of the guidelines, it has been argued at several points that the guidelines apply directly to other types of AI systems including so-called rule-based, frame-based, knowledge-based, model-based, and case-based systems as well as to object-oriented systems and perhaps neural networks and genetic algorithms as well (especially to the extent to which their application-specific knowledge is represented in a declarative, as opposed to procedural, form).

It has further been argued that most of the guideline packages can be applied directly and with great advantage to conventional software, since most of the techniques recommended are conventional methods. In this regard, it is of interest to note that guidelines of comparable level of detail and thoroughness of evaluation are not known to be available for conventional methods. While certain sets of methods are sometimes recommended, in government and other publications concerning V&V of conventional systems, the justification for the inclusion, or ordering, of specific methods is absent or impoverished relative to the present project. In fact, based upon the evaluations and studies of a wide variety of methods early in this project, a number of the most highly traditionally-recommended methods (e.g., structural testing, branch testing, test-coverage analysis, conditional testing) are judged to be much less effective than others recommended here. In particular, the emphasis on the cost-effectiveness of static analysis techniques over dynamic testing is a major broad distinction.

Aside from the development of the V&V guidelines, this project is believed to have four subsidiary accomplishments of note, relating to a broader set of issues. First, for the first time a methodology is available which characterizes a majority of individual V&V techniques on a number of different power and ease-of-use features to permit

a rational basis for selecting different methods to use as a function of differing needs. At the present time the characterizations are best-judgments, but in the future these may be replaced by consensus and empirically-based values.

Second, a more rational approach has been introduced for composing a set of V&V methods for use on a particular system, as a function of the need for specific types of assurances, and for subsequently assessing the reliability of a system after application of these techniques. Called *Fault-Specific Verification*, the approach identifies the specific failure which need to be avoided for a system and designs a set of test procedures to detect such failure.

The third subsidiary accomplishment of the project is the design and use of an effective behavioral methodology for empirically determining the usability and effectiveness of various V&V methods, existing or even only designed. This experimental method was judged to be highly successful and applicable to a wide variety of issues even beyond V&V.

Finally, the last subsidiary accomplishment is viewed to be the development of a general methodology for generating the validation scenarios of nuclear utility systems for use in the final stage of V&V, after application of other static and dynamic methods, to demonstrate the validity of the system in some range of its intended function known to be difficult or problematic.

9 REFERENCES

- Alagić, S., *Object-Oriented Database Programming*, Springer-Verlag, New York, New York 1988.
- Booch, G., *Object Oriented Design: With Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.
- Coad, P., and E. Yourdon. *Object-Oriented Design*, Yourdon Press, Englewood Cliffs, New Jersey, 1991.
- Cox, B.J., *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley Publishing Co., Reading, MA, 1987.
- Graham, I., *Object Oriented Methods*, Addison-Wesley Publishing Co., Wokingham, England, 1991.
- Kim, W., and F.H. Lochovsky (Eds.), *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, New York, 1989.
- Martin, J., and J.J. Odell, *Object-Oriented Analysis & Design*, Prentice Hall, Englewood Cliffs, NJ, 1992.
- Meyer, B., *Object-Oriented Software Construction*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
- Peterson, G.E., *Object-Oriented Computing. Volume 1: Concepts*, Computer Society Press of the IEEE, Washington, DC, 1987.
- Peterson, G.E., *Object-Oriented Computing. Volume 2: Implementations*, Computer Society Press of the IEEE, Washington, DC: 1987.
- Shlaer, S., and S.J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press Englewood Cliffs, New Jersey, 1988.
- Smith, D.N., *Concepts of Object-Oriented Programming*, McGraw-Hill, Inc., New York, New York, 1991.
- Tello, E.R., *Object-Oriented Programming for Artificial Intelligence: A Guide to Tools and System Design*, Addison-Wesley Publishing Co., Reading, MA, 1989.
- Wirfs-Brock, R., B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
- Zdonik, S.B., and D. Maier (Eds.), *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

10 GLOSSARY

Artificial Intelligence (AI) - computer software system which encompasses expert subject matter knowledge in a declarative format.

Expert System (ES) - a subject of artificial intelligence which consists of four components: knowledge base, inference engine, interfaces, and tools and utilities.

Genetic Algorithms - artificial intelligence systems which use genetics as an architectural metaphor for optimization problems.

Neural Networks - artificial intelligence systems which use biologic neurons as an architectural metaphor for classification and control problems.

Object-Oriented Programming Systems (OOPS) - a type of AI with five components: objects, associated messages and methods, hierarchically organized classes of objects, external interfaces, and tools and utilities. Objects are code modules that contain both data and procedures.

Validation - the process of evaluating software at the end of the software development process to ensure compliance with software requirements (IEEE Std 729-1983, 1983).

Verification - the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase (IEEE Std 729-1983, 1983).

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER
(Assigned by NRC. Add Vol., Supp., Rev.,
and Addendum Numbers, if any.)

NUREG/CR-6316
SAIC-95/1028
Vol. 1

2. TITLE AND SUBTITLE

Guidelines for the Verification and Validation of Expert
System Software and Conventional Software

3. DATE REPORT PUBLISHED

MONTH	YEAR
March	1995

4. FIN OR GRANT NUMBER

L1530

5. AUTHOR(S)

L. A. Miller, J. E. Hayes, S. M. Mirsky

6. TYPE OF REPORT

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION — NAME AND ADDRESS (If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Science Applications International Corporation
1710 Goodridge Drive
McLean, VA 22102

9. SPONSORING ORGANIZATION — NAME AND ADDRESS (If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)

Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Nuclear Power Division
Electric Power Research Institute
3412 Hillview Avenue
Palo Alto, CA 94303

10. SUPPLEMENTARY NOTES

11. ABSTRACT (200 words or less)

This report provides the project summary of the results of the Expert System Verification and Validation (V&V) activity that was jointly funded by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute to develop guidelines for the V&V of expert and other systems. This is the first volume of an eight-volume report. The project began with a survey of conventional V&V methods that covers 153 different techniques. Quantitative cost-benefit and an effectiveness measures were developed to permit comparisons among all the methods for three levels of stringency of V&V: low, medium, and high (Classes 3 to 1, respectively). A survey was conducted concerning V&V practices in use for expert systems, finding that they were not common, but that there was considerable activity in developing methods for knowledge bases. Selected V&V methods were applied to two existing expert systems used in nuclear power applications. Other V&V methods were investigated in an empirical experiment to assess their practical utility. A method for generating validation scenarios was developed. Finally, a set of guidelines recommending specific V&V methods for 16 different system-development situations was developed.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

validation, verification, V&V expert systems, knowledge base,
guidelines, scenarios, software quality assurance

13. AVAILABILITY STATEMENT

Unlimited

14. SECURITY CLASSIFICATION

(This Page)

Unclassified

(This Report)

Unclassified

15. NUMBER OF PAGES

16. PRICE