NUREG/CR-6316
SAIC-95/1028
Vol. 5

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

Rationale and Description of V&V Guideline Packages and Procedures

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

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

NUREG/CR-6316
SAIC-95/1028
Vol. 5

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

Evaluation of Knowledge Base Certification Methods

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

ABSTRACT

This report is the fifth volume in a series of reports describing the results of the Expert System Verification and Validation (V&V) project which is jointly funded by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute toward the objective of formulating guidelines for the V&V of expert systems for use in nuclear power applications. This report provides the rationale for and description of those guidelines. The actual guidelines themselves (and the accompanying 11 step by step procedures) are presented in Volume 7, "User's Manual."

Three factors determine what V&V is needed: (1) the stage of the development life cycle (requirements, design, or implementation), (2) whether the overall system or a specialized component needs to be tested (knowledge base component, inference engine or other highly reusable element, or a component involving conventional software), and (3) the stringency of V&V that is needed (as judged from an assessment of the system's complexity and the requirement for its integrity to form three Classes). A V&V guideline package is provided for each of the combinations of these three variables. The package specifies the V&V methods recommended and the order in which they should be administered, the assurances each method provides, the qualifications needed by the V&V team to employ each particular method, the degree to which the methods should be applied, the performance measures that should be taken, and the decision criteria for accepting, conditionally accepting, or rejecting an evaluated system. In addition to the guideline packages, highly detailed step-by-step procedures are provided for 11 of the more important methods, to ensure that they can be implemented correctly. The guidelines can apply to conventional procedural software systems as well as all kinds of AI systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This report is the fifth volume in a series of reports describing the results of the Expert System Verification and Validation (V&V) project which is jointly funded by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute toward the objective of formulating guidelines for the V&V of expert systems for use in nuclear power applications. This report provides rationale and description of those guidelines. The actual guidelines themselves are given in the User's Manual, Volume 7.

Three factors determine what V&V is needed. First, the stage of the developmental life cycle determines what "artifacts" of the system are available for evaluation, whether they are directly the representation of the system (e.g., as design or code) or whether they are the procedures and plans associated with those (e.g., the developmental plan, the V&V plan). These artifacts, in turn, determine what kinds of problems, or *faults*, are logically possible. Such faults are probably the single most important factor influencing the choice of V&V methods. The recommended guidelines are oriented explicitly toward providing assurances that specific types of faults are absent from the software. Although different life cycles identify different types of phases, the present guidelines focused on the main three developmental stages: requirements, design, and implementation.

The second factor in determining what V&V is needed is the type of component being tested. Three categories were identified: (1) either the overall system or else system components which were implemented in conventional procedural programming languages; for these, the conventional V&V techniques can all be utilized; (2) the specialized knowledge base component of AI systems, whether rule-based, frame-based, or other; for this type of component special testing approaches are recommended; and (3) components which are highly reusable from application to application and typically are not modified for particular applications; these would include the inference engine and commercial tools and utilities; procedures which permit certification of these components are the recommended V&V techniques.

The third and last factor identified as a determinant of the V&V strategy is the stringency of V&V that is needed, as based on judgments of the system's complexity and its required integrity (the latter variable refers to the extent to which the system must avoid failures because of their various negative consequences, including safety hazards). Three such Classes of V&V stringency were identified: Class 3 calls for a minimum amount of V&V and is associated with low judged system complexity and low required integrity; Class 2 calls for substantial V&V and is associated with medium-high complexity and medium required integrity; and Class 1 entails maximum levels of V&V, being associated with high integrity requirements and high system complexity.

The present guidelines provide assistance in determining one's level with respect to all three of these factors. A table is then provided which provides the identification of a V&V Guideline "Package" for each of the 27 combinations of the three factors (an additional Package is provided for the case of system maintenance). The V&V Guideline Packages specify all of the V&V methods recommended for that particular combination of factors, the order in which the V&V methods should be administered, the assurances each method provides, the qualifications needed by the V&V team to employ each particular method, the degree to which the methods should be applied, the performance measures that should be taken, and the decision criteria for accepting, conditionally accepting, or rejecting an evaluated system. All of the recommended V&V methods have been identified in the review of V&V techniques -- both conventional and AI -- given in Volume 2 of this series. Because the V&V Guidelines cover both overall system testing and the testing of conventional-language components, the guidelines can readily be used for completely conventional software systems as well as for AI systems. 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.

Concerning the last two aspects of the guidelines, the performance measures and decision criteria, generic measures and criteria were specified which are common to all situations. The three common measures identified for all life cycle phases were (1) *number of show- stoppers* (discovered errors which are so pervasive or general that they bring into question the whole design structure of the system); (2) *number of major problems* (problems which, while not show stoppers, still require substantial corrective action over a number of elements or modules of the system); and (3) *number of minor problems* (problems which are confined to highly localized parts of the system and which are easily corrected). Generic procedures for defining these measures were provided. A standard set of thresholds was established for all V&V methods, one set for each of the three Classes of V&V, for each of the three possible outcomes: (unconditional) Acceptance, Conditional Acceptance, and Rejection. The first outcome, Acceptance, requires the developers only to fix any detected errors before continuing to the next activity. The second outcome, Conditional Acceptance, requires fixing of errors and then a repeat of the appropriate V&V Guideline Package. The last outcome, Rejection, calls for a cessation of further V&V and development until the issues identified by the V&V methods are resolved by management discussion.

The V&V Guideline Packages are supplemented by very highly detailed procedures which provide step-by-step instruction for implementing 11 of the more important V&V methods. These procedures describe the conditions that must hold before a particular method should be used, the setup tasks, usually a high-level overview of the major steps in the method, and the very specific actions composing each of these. These procedures are presented in Volume 7, User's Manual.

Although project development and management aspects are not a part of the guidelines, these issues were addressed briefly. Nonetheless, a major section dealt with the explanation how the guidelines provide a very strong Risk-Management approach to the V&V of software systems. The key point is that the present approach is driven by two types of risks: (1) the consequences of failure, based on required integrity and the judged level of complexity, which determines the recommended V&V stringency, and (2) the risks associated with the faults possible for each life-cycle artifact, which determines the specific V&V method recommended. Use of the guidelines in the manner specified should result in a balanced approach to V&V which weighs the cost of this activity against the benefit of assurances that specific faults are not present.

1 INTRODUCTION

1.1 Purpose and Scope

This report provides the rationale for and description of the guidelines developed for the verification and validation (V&V) of Artificial Intelligence (AI) software systems, particularly Expert Systems and Knowledge Based Systems, but also for Frame-Based Systems, Model-Based Systems, Case-Based Systems, and Object-Oriented Systems. This work was jointly supported by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute. The guidelines are intended to apply to all potential nuclear industry applications, including those involving safety-critical aspects. The actual guidelines are presented in Volume 7, "User's Manual."

Because the guidelines cover the conventional software components of AI systems, and because there is no inherent restriction in the guidelines towards nuclear systems alone, *these guidelines can be used for the V&V of conventional software as well as AI systems for all types of applications.*

The scope of the guidelines is indicated by the steps in Figure 1.1-1 and covers determining what are the developed (or to-be-developed) system components, the recommended stringency of V&V, the relationship of the V&V techniques to the life-cycle steps, planning and execution of V&V activities, and the acceptance criteria for the system. Different choices for the first three steps will lead to use of different specific V&V guidelines. In particular, step two determines the stringency of V&V needed, as a function of the system complexity and the required integrity of the system. The highest stringency level is intended to lead to the highest system reliability.

The V&V guidelines described in this report follow an explicit Cost-Benefit Risk Management Approach for V&V Systems with higher risk of negative consequences of failed or incorrect operation having correspondingly more powerful and effective recommended V&V methods. The increasing *costs* of applying these methods for higher integrity systems are intended to be offset by the presumed *benefits* of decreasing the chances of these risks being realized.

It is important to make very clear that these guidelines -- as carefully developed as they were -- are based in most instances on informed engineering judgment and constitute the best opinion of the authors. There is, unfortunately, very little empirical data to bring to bear on the issue of providing comprehensive cost-effective guidance for the V&V of AI and other software systems. Every aspect of this report should be interpreted as a thoughtful suggestion which needs to be examined and validated within the user's own context. The suggestion, for example, that there be three levels, or classes, of V&V, from least stringent (Class 3) to most stringent (Class 1) is only a reasoned proposal: users perhaps could identify a rationale which would make two or five levels of V&V more appropriate for their system development context. Similarly, personal experiences with certain V&V methods or special organizational capabilities for supporting certain techniques should quite definitely be considered when reviewing the recommendations given here, and substitutions should be freely made on these bases. The guideline packages recommending specific V&V methods as a function of Class of V&V, life-cycle phase, and system component to be tested should be considered as data points from which users may well want to extrapolate -- by adding, dropping, or modifying methods and method orders.

1.2 How the Guidelines Were Developed and Their Contents

Figure 1.2-1 shows the key steps in the development of the guidelines. The primary input came from the results of evaluating the strengths and weaknesses of alternative V&V methods as determined by surveys (Volumes 2 and 3 of this report), by their application (or consideration of application) to two nuclear AI systems (Section 5 of Volume 1), by a behavioral experiment evaluating effectiveness of techniques (Volume 4 of this report) and recommendation of an overall set of V&V methods for a variety of specific situations (Section 4 of Volume 1). These

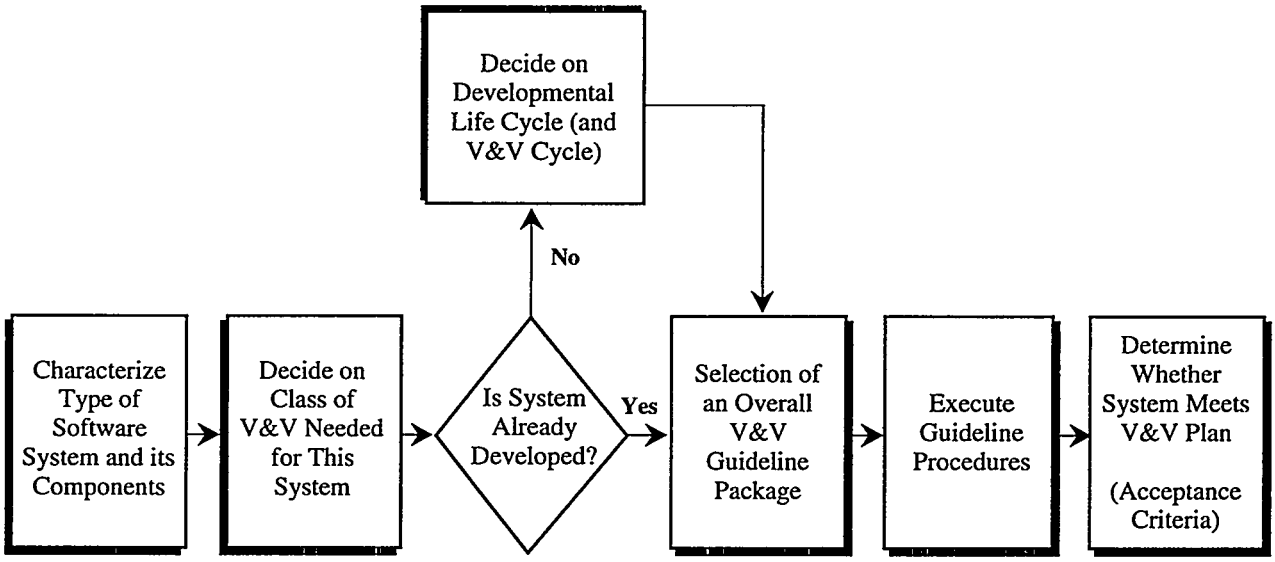


Figure 1.1-1: V&V Activities Covered by the Guidelines

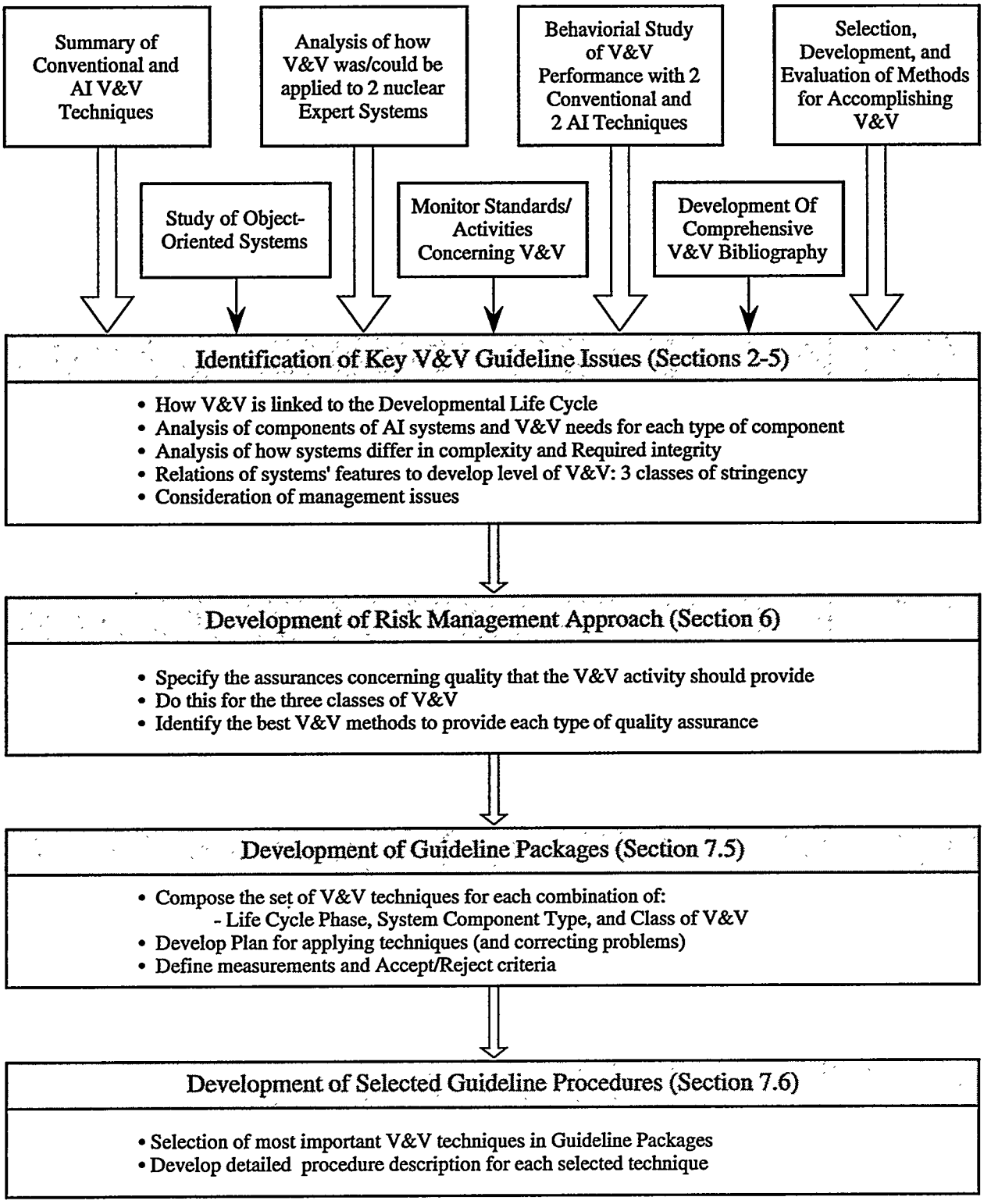


Figure 1.2-1 Input and Steps in the Development of the V&V Guidelines

results were supplemented by the findings of three smaller efforts (discussed in Section 7). All of this work resulted in the identification of a set of six key V&V issues:

- the role of V&V in the phases of the developmental life cycle,
- the recognition that AI systems can differ greatly, and that *complexity* and *required integrity* are primary bases for differences among systems,
- the relation of the above two factors to the need for more or less stringent V&V, leading to three V&V Classes,
- the quality assurances provided by the recommended V&V techniques for the three V&V Classes,
- an analysis of AI system components and their differing needs for V&V,
- and an evaluation of the merits of alternative V&V techniques taking into account these factors, especially the three V&V Classes.

Based on these issues, which will be discussed in detail in the following sections, a set of **guideline packages** were developed for the 27 combinations of three phases of the life cycle (Requirements, Design, and Implementation), three types of system components, and the three recommended V&V Classes. An additional package was developed for the system maintenance phase. There is a total of 16 V&V guideline packages since some of the different combinations can use the same guideline packages. These packages provide a high-level plan of V&V techniques to apply to the software system, and they specify the accept/reject criteria for determining whether the system has successfully met the V&V objectives. The guideline packages and plans are summary views of what needs to be accomplished, but do not include specific details. For a selected set of 11 V&V techniques considered especially important and not fully described in the literature, guideline procedures were prepared to provide the step-by-step instructions for accomplishing these techniques. It should be noted that the guideline packages, on average, contain a wide selection of V&V methods. This is because there are a large number of potential fault classes, and each V&V method addresses only a subset of these. A diversity of potential flaws requires a diversity of methods to address them. The general algorithm in selecting methods for inclusion in a package was as follows:

(1) methods highly rated in terms of cost-benefit and effectiveness in the review described in Volume 2 or the experiment described in Volume 4 are highly preferred;

(2) the more stringent the V&V class, the more powerful and numerous are the methods;

(3) powerful static-testing methods are preferred over and used extensively before dynamic ones;

(4) only the very best of the wide variety of dynamic testing methods were chosen;

(5) selection of both the static and dynamic methods was done according to a principle of maximizing the difference among methods in terms of faults covered and method of application;

(6) methods were selected to support decision support systems which form the bulk of AI systems; and

(7) methods were chosen appropriate for the particular component being testing.

NOTE: If there is a single most important suggestion above all else, it is to use computer-automated tools for every possible V&V activity, from checking the spelling, to generating test-cases and logging test results, to simulating the whole design. CASE tools, checkers, compilers, and editors all can provide effective reliable support making the task of V&V simpler and of higher quality.

1.3 Overview of Guideline Document

The next five sections, Sections 2-6, provide overviews and brief rationales for the key components of the V&V activities covered in the guidelines. In practice, these stand independent of the guidelines and need not be consulted for developing a V&V plan for a specific system. This is certainly true after several uses. Section 2 briefly considers the variety of applications specifically targeted for V&V activities. It then covers all of the AI types of software systems to which the V&V guidelines are applicable. Section 3 identifies three classes of stringency of V&V and briefly describes a process for determining the class for any particular system. Section 4 presents a conventional developmental life cycle and a recommended iterative alternative one. Section 5 presents the V&V methodology that drives the guidelines. It also shows how management practices, such as Quality Assurance and Configuration Management, affect V&V. Section 6 describes the risk management approach of the guidelines. The last section, Section 7, provides the description for the specific guidelines for developing and executing V&V plans.

2 APPLICATIONS AND SYSTEMS COVERED

2.1 Nuclear Applications

Expert systems and other AI systems may 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. AI systems could assist in any of these areas.

In **Operation**, AI systems could play important roles in aiding reactor operators during abnormal events or accidents as well as normal operations. Operations-oriented applications include emergency operating procedure identification, alarm signal processing and filtering, refueling outage management, and real-time monitoring of safety and/or economic plant performance parameters. In particular, model-based and qualitative reasoning AI systems are especially suited for such operational assistance.

As the average age of U.S. nuclear power plants continues to increase, **Maintenance** plays a larger role in their continued reliable, safe, and cost effective operation. 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 failure by assessing the meaning of changes in measured performance parameters, and they can assist in determining the impact on safety of proposed changes to plant hardware or processes.

That AI systems can play a major role in nuclear applications is evidenced by a survey which lists almost 300 such developed systems (cf. Bernard & Washio, 1990). 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 the anticipated need for its reliability and integrity are guideline considerations (see Section 3). For this reason, *the present guidelines could apply equally well to any AI system or conventional software application, within the nuclear industry or without.* 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.

2.2 Types of AI Systems Covered and their Components

The guidelines cover a variety of AI software approaches, as discussed below and as indicated in Table 2.2-1. *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.

2.2.1 AI Systems

The V&V guidelines were developed especially for a particular kind of AI Systems: **Expert Systems** or **Knowledge-Based Systems**. From a V&V point of view these two are considered equivalent, and the former term will be used. AI systems have been identified (Miller et al., 1993) as having four components: Knowledge Base, Inference Engine, Interfaces, and Tools and Utilities.

The last three components are almost always implemented using conventional procedural programming languages, and the guidelines cover the appropriate V&V methods for these based upon an extensive enumeration and evaluation of conventional V&V techniques. However, the greatest contribution of these guidelines for non-conventional systems is in the concern for the first component, the knowledge base. In fact, special analytical tools were designed and tested to insure effective quality assurance of this component.

Table 2.2-1 Applicability of the guidelines to various types of AI systems

Types of AI Systems	Applicability of Guidelines	
	Full	By Extension
Expert Systems	X	
Knowledge-Based Systems	X	
Rule-Based Systems	X	
Frame-Based Systems	X	
Model-Based Systems	X	
Case-Based Reasoning Systems	X	
Qualitative Casual Reasoning Systems	X	
Object-Oriented Programming Systems		X
Neural Networks		X
Genetic Algorithms		X

The knowledge base is the component which contains detailed information about the AI system subject matter. This component is frequently composed of IF-THEN **Rules**, and AI systems containing knowledge represented in this form are often referred to as **Rule-Based Systems.** As an alternative to rules, the knowledge base may include representations in the form of special declarative knowledge **Frames**, a structured collection of knowledge-attributes and their values. Systems based on frames, rather than rules, are commonly called **Frame-Based Systems.** The primary way of organizing a frame knowledge base is by means of a hierarchy with subclassification relations (e.g., a Vehicle might have three sub-types, Land-Vehicles, Air-Vehicles, and Water-Vehicles). Hierarchies are not only a way of organizing knowledge, but they also simplify the description of each knowledge element. The simplification occurs since sub-classes can carry down many of their descriptive features from their parent-class without having to copy the features over. This special copying procedure is called "inheritance". The guidelines also fully apply to these AI systems. Other types of AI systems identified as **Model-Based Systems, Case-Based Reasoning Systems,** or **Qualitative Causal Reasoning Systems** are types of AI systems, with rules, frames, or other types of declarative representation knowledge bases. The guidelines apply to all of these.

Concerning the other components of AI systems, the *Inference Engine* uses the knowledge base to make decisions or to take actions. *Interfaces* deal with the connection of AI systems to the databases, communication channels, users, etc. *Tools and Utilities* refers to general application programs which may be used in building the knowledge base or assisting in any other features of the AI system. As compared to AI systems, conventional software typically does not have a knowledge base or inference engine.[1]

2.2.2 Object-Oriented Systems

Another type of AI software system which is strongly addressed by these guidelines is **Object-Oriented Programming Systems** (OOPS). OOPS can be viewed as having five components: (1) *Objects,* (2) their associated *messages* and *methods,* (3) the hierarchically-organized *classes* of objects, (4) external *interfaces,* and (5) *tools* and *utilities. Objects* are code modules that contain both data and procedures. The *methods* are one type of object-procedure and are responsible for actions of computation, display, or communication with other objects. The communication act is accomplished by sending *messages.* Objects are described in terms of abstract *classes* (or types) very much like Frames. Specific objects are created as instances of an object class. The abstract object classes and their associated object-instances are organized into hierarchies, that is Class and Object, just as Frames are (cf. previous section). The same mechanism of inheritance is used to pass down information from parent-classes to their sub-classes.

2.2.3 Neural Networks and Genetic Algorithms

There are two final types of AI systems which should be mentioned: **(Artificial) Neural Nets** and **Genetic Algorithms.** 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. Neural Nets are most frequently developed in conventional procedural programming languages. The same is the case for genetic algorithms. Since the present guidelines cover recommendations for the conventional software components of larger AI systems, these two additional systems would indeed be covered with these guidelines. To the extent Neural Nets or

---

[1] The concept of *components* may be applied to systems implemented using conventional procedural programming languages. Doing so, conventional systems can be said to have three components: *Non-Interface Code, Interface Code,* and *Tools and Utilities.* Each of these three components can be further divided into three types of potential subcomponents: *Main Procedure, Embedded Procedure,* and *Function.* Additional sub-classifications can be derived from the features of the specific programming language (e.g., *declarations*). Conventional V&V techniques apply to all of these components, either as indicated by the present guidelines or otherwise.

Genetic Algorithms involve declarative representations of application knowledge, corresponding to an AI System knowledge base, these guidelines apply more directly.

2.3 Overall Categorization of System Components and Recommended V&V Philosophy

The components of all the varieties of AI systems, as well as the components of conventional procedural programming systems, can be categorized into three types of components *from the point of view of deciding on appropriate V&V approaches*. The three types and the recommended V&V philosophy for each are shown in Table 2.3-1. The guideline packages reflect these philosophies.

The first type, called **Declarative Knowledge Structures,** includes Rule Bases (from rule-based AI systems), Frames (from all types of frame-based systems), and Class Hierarchies (from object-oriented programming systems).[2] These components use a variety of syntax and naming conventions to explicitly represent the details of application knowledge. Thus, these components of AI systems are very amenable to a V&V philosophy of **formal verification,** that is, the use of formal mathematical theorem-proving techniques to prove a variety of properties about a component, such as redundancy, incompleteness, syntax violations, and inconsistencies. Although this approach is not yet mature, it is emphasized as the most effective V&V strategic approach for this type of component.

The second type of system element concerns **Highly Reusable Components,** components which can be reused over a wide variety of applications without needing any (or very little) customization to specific systems. These would include general tools and utilities, some generic system interfaces, and some general computational functions. For such components there is little sense in retesting them for each new application. Rather, a **certification** procedure should be developed which establishes the functional and performance characteristics of each such component, independent of the application. Wherever possible, this role should be performed by the commercial vendor of these components.

The third type of component is a catch-all category containing all the remaining types of system elements, including the overall integrated system itself, most of the interfaces, conventional databases, and all the other application-specific code. For this category, the traditional set of V&V methods should be applied to each new application.

---

[2] These structures richly characterize their internal information. Relative to them, conventional databases (e.g., flat, relational) are greatly impoverished, and the V&V philosophy recommended here does not apply.

Table 2.3-1 Recommended V&V philosophy for general categories of system components

Types of System Components	Recommended V&V Philosophy		
	Formal Verification	Certification	Traditional System V&V
Type 1 **Declarative Knowledge Structures:** • Rule Bases • Frames • Class Hierarchies • Objects Stored in Databases	X		
Type 2 **Highly Reusable Components:** • Tools & Utilities • Some Interfaces • Some Functions • All Commercial Software Packages used		X	
Type 3 **Mostly Non-reusable Components:** • Overall System • Most Interfaces • Conventional Databases • All Remaining Application-Specific Code			X

3 V&V METHODS

3.1 Source of V&V Methods

The V&V methods recommended in the guidelines are based on the results of previous tasks. The survey of existing V&V methods for conventional systems (Volume 2) identified 153 different types of methods, and these (or their extensions) provide the basis for the recommended conventional techniques. The survey of existing V&V methods for AI systems (Volume 3) provided insight into the state-of-the-art for AI system-specific methods which provided only four new techniques.

In an experiment with two nuclear AI systems (Volume 4) to evaluate their actual behavioral effectiveness, two conventional techniques and two of the new AI system techniques were used. The techniques used were *Requirements Tracing* (which includes Requirements Grouping), *Desk Checking, Knowledge-Base Syntax Checking*, and *Knowledge-Base Knowledge Checking.* Based on the positive outcomes, all four of these methods were chosen to be recommended in the guidelines. Finally, the consideration of deficiencies of previously considered V&V methods for testing AI system led to the proposal of new methods, as extensions to existing techniques, for inclusion in the guideline's arsenal of methods (in Section 4 of Volume 4).

3.2 Classes of V&V Techniques

In the conventional V&V survey, methods were 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. The quality assurances provided by the guidelines and the recommendations for specific methods are organized in terms of these three life-cycle phases.[3]

For the implementation phase, there are two main subclasses of methods. They are *Dynamic Testing* and *Static Analysis* techniques. The former involves actual execution of the code on an appropriate platform and operating system. Static Analysis involves no code-execution but rather various types of analyses, either manual or automated, of the source programs. In previous reports (for Volumes 2 and 4), Static Analysis methods were judged to be more effective and considerably less labor-intensive and costly than dynamic testing. The present guidelines therefore emphasize static analyses for the implementation phase.

3.3 V&V Evaluation and Ranking

Each of the 153 conventional V&V methods was characterized using 8 separate factors that permit an assessment of the methods' effectiveness for systems with varying levels of complexity and integrity. These factors are Broad Power, Hard Power, Formalizability, Human-Computer Interface Testability, Ease of Mastery, Ease of Setup, Ease of Running/Interpretation, and Usage. To determine the extent to which a V&V technique could detect different software defects, a taxonomy of 52 different types of conventional software defects was developed. Each of the 153 conventional V&V techniques covered anywhere from 2 to 52 of these defects. Each defect was covered by anywhere from 21 to 50 V&V techniques. In addition, the 8 factors were used to find the relative cost benefit of each method. The methods were then ranked by life-cycle phase and by V&V class based on this relative cost-benefit metric. Tables 6.3.3-1A through C, 6.3.3-3A through C of Volume 2 present this ranking.

---

[3] In the Volume 2 report, methods for Requirements and Design were grouped together. They are separate in these guidelines.

4 DEVELOPMENTAL LIFE CYCLE

The term *life cycle* refers to the "start-to-finish" phases of system development. The software development life cycle encompasses requirements specification, design, implementation, integration, field installation, and maintenance. The life cycle provides a systematic approach to the development and maintenance of a software system. Successful application of V&V techniques requires a well-defined and implemented software life cycle. There are two types of life cycles, the sequential and the iterative. The sequential life cycle is a once-through sequence of steps without providing formal feedback from later phases to prior phases. The iterative model involves repeated cycling through life cycle phases. A very detailed treatment of life cycles can be found in Section 4 of Volume 2 (including the rapid prototyping and iterative development so characteristic of expert systems).

For the purpose of the guidelines, a traditional waterfall life cycle based on NSAC-39 (Figure 4.0-1) has been assumed. It comprises the following activities:

1) Requirements
2) Design
3) Coding/Integration and Testing
4) Maintenance

Although maintenance is not specifically mentioned in NSAC-39, it is a vital life cycle activity and has been considered in the guidelines in Section 7. This report discusses only the V&V aspects of these activities. Section 4.2 of the NUREG/CR-6018 report provides a more detailed discussion of these phases for the interested reader. The NSAC-39 life cycle was chosen as the reference model because it is the simplest and most well-known within the nuclear industry. It should be noted, however, that this life cycle is not the most appropriate for AI systems development. An iterative model would serve that purpose better (cf. Volume 2).

4.1 Review

Requirements verification is used to determine if the specified requirements correctly and completely describe the system so that it satisfies its intended purpose. A quality Requirements Specification is crucial to the overall success of the development effort, just as a good set of plans and blueprints are crucial to successfully building a house. During requirements verification, each software requirement is uniquely identified and evaluated for software quality attributes including correctness, consistency, completeness, understandability, accuracy, feasibility, traceability, and testability.

4.2 Design Review

After requirements verification, Software Design Documents are produced and a Critical Design Review is held. During this phase, it is important to ensure that the software design correctly represents the requirements and to identify additional functions needed to implement them. The Software Design Document is also evaluated for: correctness, completeness, consistency, accuracy, and testability; compliance with applicable standards; interface consistency; and computer-user interface sufficiency.

4.3 Validation Test

During the coding and implementation phase of development, design is implemented into source code. It is important at this phase to ensure that the source code correctly represents the design. V&V activities during this phase include analyzing interfaces between source code modules for compatible data elements and types; tracing source code to design elements and evaluating for completeness, consistency, correctness, and accuracy; preparing an

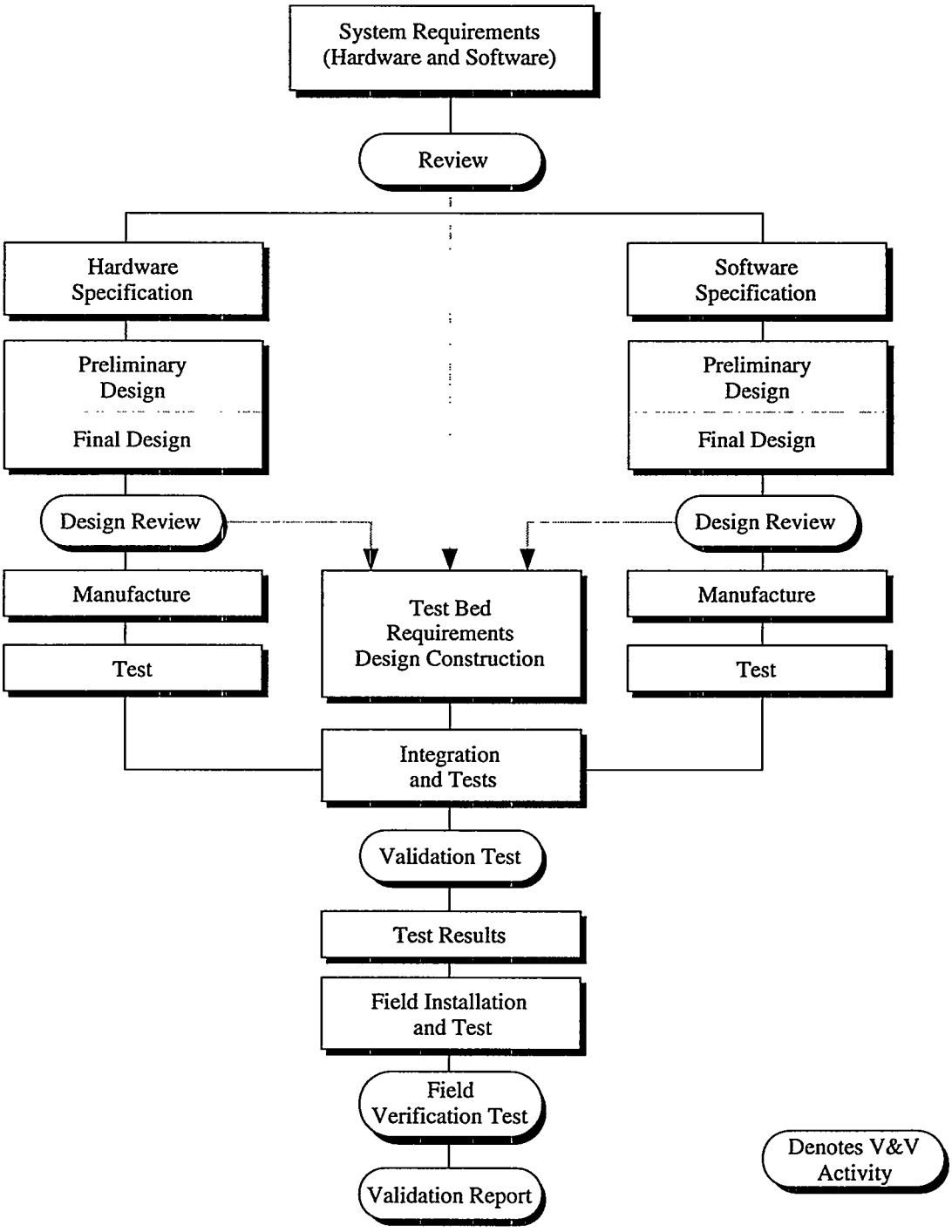


Figure 4.0-1 Relationship of V&V Activities to Generic Project Activities, From NSAC-39 (1981)

Acceptance Test Plan; and reviewing documentation such as programmer and user's manuals for completeness, consistency, correctness, and accuracy. A Verification Readiness Review is then held to determine the system's readiness for integrated testing. After the coding and implementation phase, the system as a whole is evaluated against the original Requirements Specification. The Acceptance Test Plan is used to perform this validation. Testing and analysis, preferably performed by an independent third party, are the keystones of this activity. Test results are evaluated against the criteria specified in the test procedures. Test results are verified to ensure that the correct test inputs are used, outputs are correctly reported, and all test cases were correctly executed in the appropriate environment.

In rapid-prototyping development efforts utilizing an iterative life cycle, maintaining a regression test case set is especially important. These test cases are indexed to the requirement(s), design element(s) and code module(s) they test. If the requirements, design, or code modules change, the test cases then need to be updated.

4.4 Maintenance Test

More likely than not, the software or its operational environment will be modified after installation and operation. To maintain the verified and validated status of the software, an ongoing V&V program must be established. The V&V methodology used during the development effort should be revised to reflect any operational constraints. It is very important to ensure that configuration control of the software is maintained.

5 MANAGEMENT ASPECTS OF VERIFICATION AND VALIDATION

Software must be managed, be it conventional software systems or AI systems (which are just another form of software systems). Management of software systems falls into many categories. These include managing the software development life cycle, developing a software system to meet exacting industry standards, maintaining configuration control of the developing or developed software system, ensuring that proven, standardized software engineering practices are used to develop the software, and managing the quality assurance activities for the software. Each of these management activities warrants further consideration, because they all influence, directly or indirectly, the quality of V&V that can be obtained.

5.1 Management of the Software Life Cycle for V&V

Just as one should not even contemplate sitting down and writing code when first handed a requirements document, one should not wait until a system has been coded to perform V&V. However, traditionally this is what happens. Verification and validation is considered to be synonymous with testing; management waits until a system is developed to test it, instead of performing V&V during the most crucial early phases of the life cycle.

V&V should take place throughout the life cycle, not only at the end of it. Recovery from defects discovered during the requirements or design phases is relatively inexpensive. Once the system has been fully implemented, re-design and complete code rewrites, which are labor intensive and expensive, are often needed. A variety of evidence shows that the relative cost of fixing an error increases drastically as a function of the life cycle phase in which it was detected (Boehm, 1981; Humphrey, 1990). As a result, good V&V performed early in the life cycle can provide tremendous cost savings to the project. The cost savings can more than justify the expenditure for the V&V itself.

5.2 Standards

A detailed discussion of the eight key standards and guideline documents which are currently the primary source of V&V guidance in the nuclear industry (shown in Table 5.2-1) can be found in Section 2.3 of Volume 2. These standards discuss a range of topics, including how to structure a V&V program for a Safety Parameter Display System (SPDS), software quality assurance requirements for thermal-hydraulic safety analysis software, mapping of recommended software quality assurance practices against the 10 CFR 50 criteria for a complete nuclear quality assurance program, etc. The key lessons to be learned from all of these standards are the following:

1. How and why to establish a software development life cycle,
2. The Software Quality Assurance (SQA) practices and procedures that should be followed at each step,
3. The documentation to be produced/revised at each step,
4. Criteria for testing and validating nuclear power software applications, and
5. Testing techniques.

5.3 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.

Table 5.2-1 Key standards and regulations related to V&V of conventional software systems

Document		Document ID	Description
Key Documents	**SPDS**	NSAC-39	Verification and Validation for Safety Parameter Display Systems, December 1981
	QA, Design and Analysis Codes	NUREG-0653	Report on Nuclear Industry Quality Assurance Procedures for Safety Analysis Computer Code Development and Use, August 1980
	QA	NUREG/CR-4640	PNL-5784, Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry, August 1987
		ASME NQA-2a-1990 Part 2.7	Quality Assurance Requirements of Computer Software for Nuclear Facility Applications, 1990
	Class 1E Real Time Systems	ANSI/IEEE ANS-7-4.3.2-1982	Application Criteria for Programmable Digital Computer Systems of Nuclear Power Generating Stations, July 6, 1982
		Reg. Guide 1.52	(Task IC 127-5) Criteria for Programmable Digital Computer System Software in Safety-Related System of Nuclear Power Plants, November 1985
	V&V	ANSI/IEEE 1012-1986	Software Verification and Validation Plans, November 14, 1986
		IEC 880	Software for Computers in the Safety Systems of Nuclear Power Stations, 1986

Configuration management controls the manner in which a system's components are modified, improved, and integrated. The four major Configuration Management (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 1042-1987.

Table 5.3-1 outlines the activities and responsibilities of the V&V team at each phase of the life cycle as compared to the QA and CM activities. Note that the items in each column are independent of each other.

5.4 Software Engineering Practices

It is essential that proven, standardized software engineering techniques 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 important, established practice, as is careful code documentation. These and other good software engineering principles are described in a number of sources (see Huffman, 1988). The software engineering techniques that are planned for use should be documented in the Software Development Plan. Then QA and/or V&V should monitor activity, by means of reviews and status reports, to ensure that the plan is being followed.

Equally as important as setting up the initial strategy for using software engineering techniques is continuously monitoring the software development process, specifically looking for areas which can be improved. These areas then need to be aggressively pursued (see suggestion by Humphrey, 1990).

5.5 Metric-Based Management

V&V should be performed within a framework that will assist management in making well-informed, risk-reducing decisions. Some commonly asked questions during software development such as "Are we ready to write code?", "Are we ready to test?", and "How do we know if we are ready to test?" can be answered through the use of software metrics.

One way to use metrics to manage a development effort is to determine software maturity at strategic points within the development life cycle, e.g., readiness for Critical Design Review, for Formal Qualification Test, for Technical Test, or for Operational Test. Categories such as stability of requirements/design/code, test coverage, fault profile analysis, etc., can be established and possibly further decomposed to assess software maturity. The Software Quality Engineering (SQE) Branch of the Product Assurance and Test Directorate (PA&TD) of the U.S. Army's Armament Research and Development Engineering Center (ARDEC) has developed such a framework called the Readiness Growth Model (Janusz, 1992).

The Readiness Growth Model (RGM) is based on the waterfall, or sequential, life cycle and was designed to measure software maturity at various decision points during the life cycle. The RGM is cumulative and makes use of: 1) requirements traceability, 2) stability of requirements/design/code, 3) test coverage, 4) testing success, 5) fault profile analyses, and 6) "other" indicators to determine software maturity. These six major areas are subdivided into 26 quantitatively assessed categories, which are weighted from 1 to 10. The 26 sub-categories are then summed to obtain a total RGM score, with 100 being the highest possible score. Points were allocated so that if the software warranted a "perfect" rating for each characteristic, the RGM score for Operational Test readiness would equal 100.

It should be noted that a project has total flexibility in determining what metrics to use to monitor software maturity. The only critical aspect is to decide on some set of quantitative procedures to guide passing from one state of development to another. Each guideline in Section 7 mentions several metrics that result from performing a specific V&V method. The user should simply select one or more of these measures to plug into a spreadsheet embodying the RGM (Janusz, 1992).

Table 5.3-1 Life-Cycle comparison of activities associated with V&V, Quality Assurance (QA), and Configuration Management (CM)

V&V	QA	CM
REQUIREMENTS o Evaluates, reviews and comments on Requirement Specification o Traces requirements to their sources o Initiates requirement tracing method to trace and verify requirements and links them to detailed design o Ensures accurate translation between Customer's Specification and vendor's Functional Specification o Inspects hardware configuration for compliance to specifications and contract o Attends reviews o Writes Discrepancy Reports (DRs) o Develops V&V Plan	**REQUIREMENTS** o Reviews specifications for obvious errors and signs off o Inspects all items received from vendors o Performs on-site factory acceptance of major vendor items o Participates in reviews o Checks for adherence to standards o Monitors vendor performance and quality of documentation o Reviews design specifications for obvious errors and signs off	**REQUIREMENTS** o Maintains originals of all documents o Maintains all source code master files o Receives and logs all documentation received from outside sources o Maintains all software and documentation libraries o Maintains archives o Maintains total hardware configuration tracking and accounting o Maintains control of originals on all documentation
DESIGN o Traces each requirement into design and adds references to requirement tracing method to show linkages o Analyzes detailed design o Analyzes design document for correctness, feasibility, consistency, testability, operational integrity, etc. o Performs/analyzes timing and sizing to ensure adequate hardware resources o Analyzes operating sequences, data flow, task interaction, and mode switching o Looks for unnecessary redundancy, unidentified design element, etc. o Emphasizes analysis of high-risk, high-priority items o Analyzes failover and device switching o Audits documentation for completeness against the delivered CM library listings - ensures 100% match between requirements document and design o Writes Discrepancy Reports	**DESIGN** o Attends design reviews o Performs analysis of design changes vs. test procedures o Checks for adherence to standards o Monitors CM actions for completeness and adherence to procedure	**DESIGN** o Manages transfers in design aspects between final CM master files and user directories o Builds system as required o Backs up system as required o Runs difference program upon request from QA or V&V to verify changes to design o Maintains informal change control o Maintains master file on all Trouble and Discrepancy Reports

Table 5.3-1 (Continued)

V&V	QA	CM
IMPLEMENTATION, TESTING AND INTEGRATION o Reviews test procedures for adequacy o Adds test references to requirement tracing method o Monitors developer's test program o Analyzes test results independently o Writes DRs on any deficiencies not written up by QA o Performs independent testing if needed o Verifies operator/user manual during testing o Performs audit of code included in final build o Looks for unnecessary, unidentified code, etc. o Audits design documentation for completeness against delivered code listings, ensuring 100% match	IMPLEMENTATION, TESTING AND INTEGRATION o Writes majority of DRs on test failures o Follows test procedure to the letter o Actually performs File Allocation Table o Participates in dry run and runs for record	IMPLEMENTATION, TESTING AND INTEGRATION o Maintains test result records and test data o Inspects and inventories systems prior to shipping o Monitors all cleaning, packing and loading of system for shipment o Participates in Site Acceptance Tests o Maintains all source code master files o Certifies final CM build o Maintains all as-built documentation o Initializes and manages all changes to documents and/or code o Runs difference programs on request from QA and V&V to verify changes to code

6 RISK MANAGEMENT APPROACH OF THESE GUIDELINES

Any time a system is developed to do useful work there are risks. Two types can be identified, those associated with developing the system and insuring that it is deployed and used, and those associated with a failure or mis-operation of a deployed system. The former are called development risks; the latter, failure risks. Development risks cover everything that could go wrong in getting the system built and installed, such as running out of time, money, or management support; trying to do something that's too hard for the development team; insufficient or poor staffing or tool support; inadequate supervision and practices; failure of external agents to deliver on time or the appropriate product; etc. These development risks are not addressed by these guidelines. They are avoidable, in great part, by proper attention to the management issues discussed in Section 5.

Only failure risks are further considered. In subsection 6.1, the relations among system complexity, flaws, failures, perceived risk and degree of required integrity are established. In 6.2, system flaws are identified for three phases of the developmental life cycle. Subsection 6.3 discusses three levels or classes of required V&V for a system based on the system's complexity and its degree of required integrity. Subsection 6.4 identifies the specific quality assurances that are provided for the three V&V classes and three life cycle phase, and the recommended V&V methods are identified. Finally, in 6.5, the cost-benefit aspects of this Risk Management approach are summarized.

6.1 Relationships Among Complexity, Failures, Faults, Risk, and Required Integrity

System *failures* (including mis-operations) and their potential severity are postulated to be a monotonically increasing function of system *complexity* (defined in detail in Section 6.3). As the complexity of a system increases, so, it is proposed, does the probability that failures will occur and that the failures will be increasingly severe. This conjecture is consistent with the software testing literature and is exemplified in Figure 6.1-1 (see page 27).

For a failure to actually occur, there must be a *fault* in the code. At the same time the proper operating conditions to expose the fault must exist. A *fault* is defined as an error which could, in an implemented system, lead to a failure (or mis-operation) of that system. The fault may have been introduced during implementation, during design or during the requirements-analysis phase of the developmental life cycle. There is a wide variety of possible faults for each life cycle phase, as discussed in the next section, and most can lead to an operational failure under the right conditions. The effects of complexity increases the number and type of faults. Figure 6.1-1 shows the relationship of complexity to faults. While a failure always entails a fault in the code, the reverse is not true. System failures can have all kinds of consequences. The concept of *risk* involves assessing the negative "value" of the possible *consequences* of system failures (weighted by the judged probability that these will occur). These consequences include not only the immediate action (or inaction) resulting from a failure or mis-operation, but also the longer-term consequences. The concept of risk also includes the effects and costs of finding the problem and restoring the system. Consequences include the following potential negatively-valued outcomes of system failures:

- 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 system elements
- discomfort to people or animals
- interruption of service and/or disruption of system mission
- inconvenience to people
- direct financial loss

- resource cost (people, equipment) to restore the system
- loss of information
- loss of opportunity
- loss of business
- loss of privilege, authority, or rights
- loss of good will or reputation
- impact on the availability or operation of other systems
- impact on an organization's capability to perform

It should be noted that there is no necessary relation between *System Complexity* and these potential negative consequences. Systems of the same complexity level, and the same failure-producing capacity, can have radically different consequences upon failure. It all depends on the particular application and the context of its installation.

The relations between financial cost and the above-listed negative consequences is mostly indirect. Computing those relations is a very complicated matter. Also, for many, the financial cost will only be one measure of these negative consequences; they also impact professional integrity, personal code of ethics, sense of civic responsibility, and so forth. Thus, the determination of the "costs" of these consequences of failure must take in many other factors besides immediate financial impact. This accounting will vary with each individual and organization and is therefore essentially subjective.

While the association of consequences of system failure to "costs" cannot be specified, that to "perceived risk" can be hypothesized. *A system will be perceived as being "high risk" to the extent that system failures could realistically cause highly undesirable consequences.* This conjecture is illustrated in Figure 6.1-2, which shows Perceived Risk growing rapidly as the belief in negative consequences grows. The shape of this curve is of minor importance. The point is that it is rational to view a system as being of higher risk to the extent it can produce a higher quantity of negative failure consequences. This point is consistent with all the literature on strategies underlying risk management and quality assurance of software (e.g., Charette, 1992; ANSI/IEEE ANS-7-4.3.2-1982, 1982; Humphrey, 1990; Boehm, 1990; Redmill, 1989).

The phrase *Degree of Required Integrity* refers to the customer's perceived need to keep the system operating without any failures, and therefore without any of the types of negative consequences listed earlier. It is reasonable to postulate that the Degree of Required Integrity will increase as the Perceived Risk of a system increases. This is shown as a linear function in Figure 6.1-3. The system is believed to be risky in terms of its potential to have "costly" failures, it is very desirable to have that system fail only rarely.

In summary, it has been proposed that the factor of System Complexity underlies system failures, by leading to the presence of system faults. Failures can have a variety of consequences. When potential failures are perceived as being highly negative, then Perceived Risk of the system to the customer/user increases. When Perceived Risk increases, the Degree of Required Integrity will increase. Thus, Complexity is the factor that determines the expected frequency of system failures, and Degree of Required Integrity is the factor that determines the level of need to avoid failures. When both these factors are high, an extensive amount of effort must be required by V&V and other activities to achieve the goal of rare failures. On the other hand, when both these factors are low, a much lower amount of V&V effort would be expected.

Section 6.3 focuses on the relationship of these two factors to the level of required V&V activities. In the next section, the specific types of faults that can lead to failures are discussed.

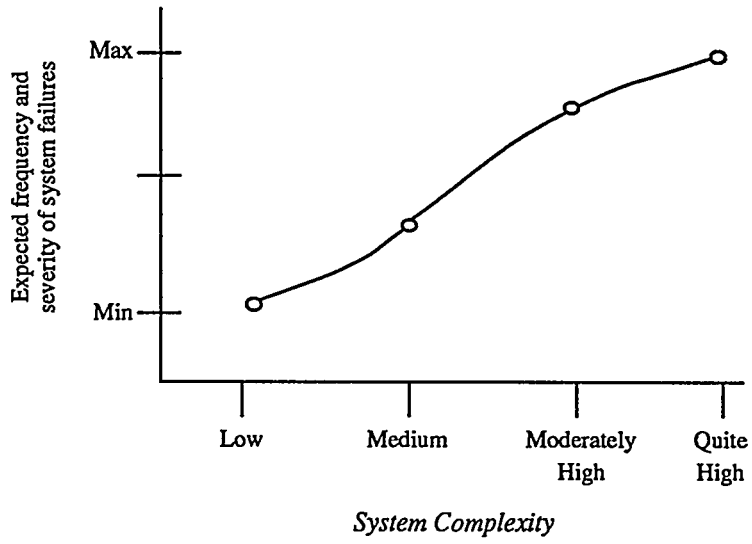


Figure 6.1-1. Hypothesized relation between system complexity and the frequency and severity of failures

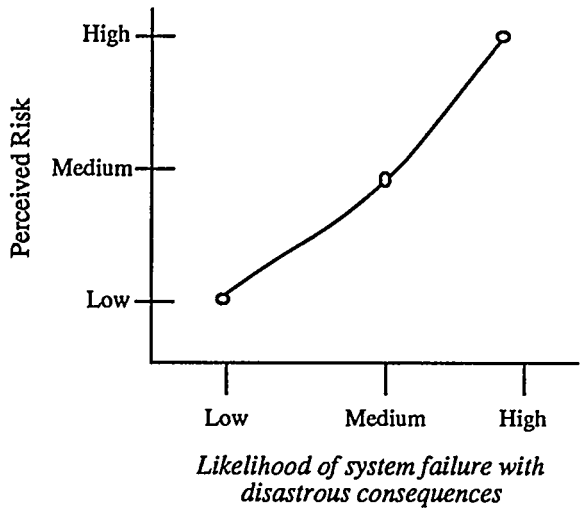


Figure 6.1-2. Hypothesized relation between perceived risk and the belief that serious errors could occur

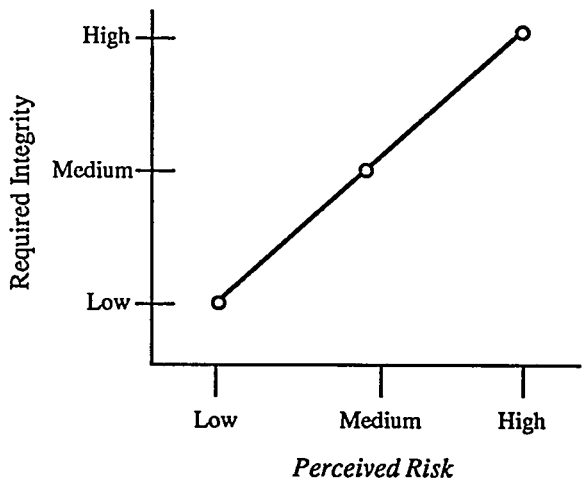


Figure 6.1-3. Hypothesized relation between Degree of Required Integrity and Perceived Risk

Figures 6.1-1, 6.1-2 and 6.1-3

6.2 System Faults as a Function of Life-Cycle Phase

Faults in the system can, in the appropriate operational context, lead to system failures. Faults increase with System Complexity, and the desirability of removing them increases with Degree of Required Integrity. The manner in which they should be removed, as well as prevented from occurring in the first place, will be determined by the specific nature of the faults. Therefore it is important to understand what types of faults they are. A major source of faults was the taxonomy of life cycle defects developed for the earlier review of conventional V&V techniques (Table 6.1.1-1, pp 90-93, Volume 2). The remainder were generated from reading of the AI literature, from review of recent software literature, and from professional experience with V&V of software systems.

The faults for the requirements stage are organized into nine main types. These are incompleteness, incorrectness, lack of clarity, lack of explicitness, lack of consistency, lack of testability, inability to be separately identifiable, inadequate human-computer interface, and inadequate operational concept. These main requirement faults are listed and described in Table 6.2-1. Each fault has a variety of subtypes, and these are listed in Table 6.2-2.

Eight or nine major types of requirements faults also carry over into the following design phase. In addition, there are seven new types of faults; these are all listed in Table 6.2-3. Table 6.2-4 provides the subtypes of some of these design flaws. Finally, the implementation faults are listed in Table 6.2-5. Five of the nine main types of faults carry over with nine new types, for a total of eighteen types. Several of these types of implementation faults, especially *knowledge-base faults*, have quite a number of subtypes, and these are shown in Table 6.2-6.

6.3 Three Classes of V&V

In Section 6.1, two factors were identified as the major determinants of the amount of V&V that might be desirable for a system. These are Complexity and Degree of Required Integrity. Complexity is the factor which, as it increases, increases the potential system failures, which can occur due to the faults just discussed. There are six contributors to System Complexity, and these are described in Table 6.3-1. Note that size, per se, is not a complexity factor.[4] Degree of Required Integrity determines how strongly the customer/user is interested in avoiding having such failures occur. This is an individual decision to be based on a wide variety of factors but primarily the types of negative consequences that can occur with system failures, as discussed in Section 6.1. High values of both of these factors would produce a high likelihood of failures at a high level of desired avoidance, and a very substantial amount of V&V would be warranted. Similarly, lesser amounts of V&V would be indicated for lower levels of these factors.

The determination of level of indicated V&V as a function of these two factors is shown in Figure 6.3-1. The three levels of complexity were chosen in terms of the six elements described in Table 6.3-1 to roughly cover equal intervals of the expected number of contained system failures as suggested by Figure 6.1-1. These three levels are low complexity, moderately high complexity, and quite high complexity. The key difference between the highest level of complexity and the next highest is the presence of direct control functions in the quite-high case. Because of the subjectivity of the Required Integrity factor, three equally-spaced levels of Low, Medium, and High were simply chosen to cover this factor's range. The cells of the resulting 3 x 3 table are numbered, and specific AI nuclear application systems are given in the bodies of the cells as examples.

---

[4] Large as well as small programs are described by the six factors. Program size primarily affects the time and resources needed to develop the application, with very large programs requiring much more extensive interim reviews, documentation, and configuration management.

Table 6.2-1 Types of Faults That Can Occur During the Requirements Stage[5]

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.
Inability To Be 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.

[5]Note: This and the following five tables are replicated in the overview, Volume 1, as Tables 4.1.4.2-1 through 4.1.4.2-6.

Table 6.2-2 Decomposition of Identified Faults for the Requirements Stage

Major Fault	Subfaults
Incompleteness	• 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	• Failure to specify initial system state when not equal to zero • Requirements do not express customers' needs -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	• 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	• Something is stylistically implied by the sentence construction but not explicitly stated • Something is logically implied but not stated
Lack of Consistency	• 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	• 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 6.2-2 (Continued)

Major Fault	Subfaults
Inability To Be Separately Identifiable	• 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)	• 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 Operational Concept	• System causes extra steps to be taken to perform a previously manual task • Operational concept (CONOPS) not supported by the system or key steps have been omitted • CONOPS violates standards or guidelines • CONOPS violates ease-of-use precepts

Table 6.2-3 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 not 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 6.2-4 Decomposition of Identified Faults for the Design Stage

Major Fault	SubFault
Incompleteness	• 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	• Design elements that cannot be traced to a requirement or specification have been included
Incorrectness	• 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[6]	• 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[6]	• Something is stylistically implied by the sentence construction but not explicitly stated • Something is logically implied but not stated
Lack of Consistency	• 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

[6] The subfaults are identical to those of the Requirements Stage

Table 6.2-4 (Continued)

Major Fault	SubFault
Inadequate Operational Concept[6]	• 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 • CONOPS violates standards or guidelines • CONOPS violates ease-of-use precepts
Inadequate Human Computer Interface	• Human Computer Interface difficult to learn/user, see/read, etc. • 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 labelled
Inadequate System Architecture	• 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	• 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
Inadequate Knowledge Structure Design	• 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	• 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	• 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	• 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	• Frequent error traps have not been provided or do not have recovery mechanism • Design possesses weak modularity • Design possesses excessively rigid control structure

Table 6.2-5 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 Flaws	Logically incorrect implementations of the design exist or control structures incorrectly implement the design.
Data Operations and Computations Flaws	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.
Lack of 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 6.2-6 Decomposition of Identified Faults for the Implementation Stage

Major Fault	Subfaults
Incompleteness	• 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	• "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 Fault	• 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 Fault	• 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
Lack of Consistency	• 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

Table 6.2-6 (Continued)

Major Fault	Subfaults
Inadequate Operational Concept	• 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	• 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	• 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 ifrelational
Safety Problem/Hazard Conditions	• 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	• 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	• Frequent error traps have not been provided or do not have recovery mechanism • Design possesses weak modularity • Design possesses excessively rigid control structure
Inadequate System Documentation	• 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 6.2-6 (Continued)

Major Fault	Subfaults
Knowledge Base Faults	<u>Syntax Faults</u> • 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 <u>Knowledge Faults</u> • Ambivalent rules • Missing rules • Semantic inconsistency defects • Logical completeness defects • Lower/upper numeric range defects • Set of legal values defects • Minimum/maximum cardinality of roles defects • Incompatibility of roles values defects • Disagreement with argument data type specifications defects • Illegal subrelations defects • Incompatible variable and reference combination in rule • Keyword not recognized • Variable not recognized • Comparison not recognized • Reference not recognized

Table 6.2-6 (Continued)

Major Fault	Subfaults
Knowledge Base Faults	• Semantic type or class violation • Necessary semantic knowledge is missing • Rule condition redundant with present state-path <u>Structure Faults</u> • 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 <u>Object-Oriented Programming Faults</u> • 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 • 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 6.3-1 Six Factors of Software System Complexity[7]

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

[7]This table is replicated in the Overview, Volume 1, as Table 2.1-1.

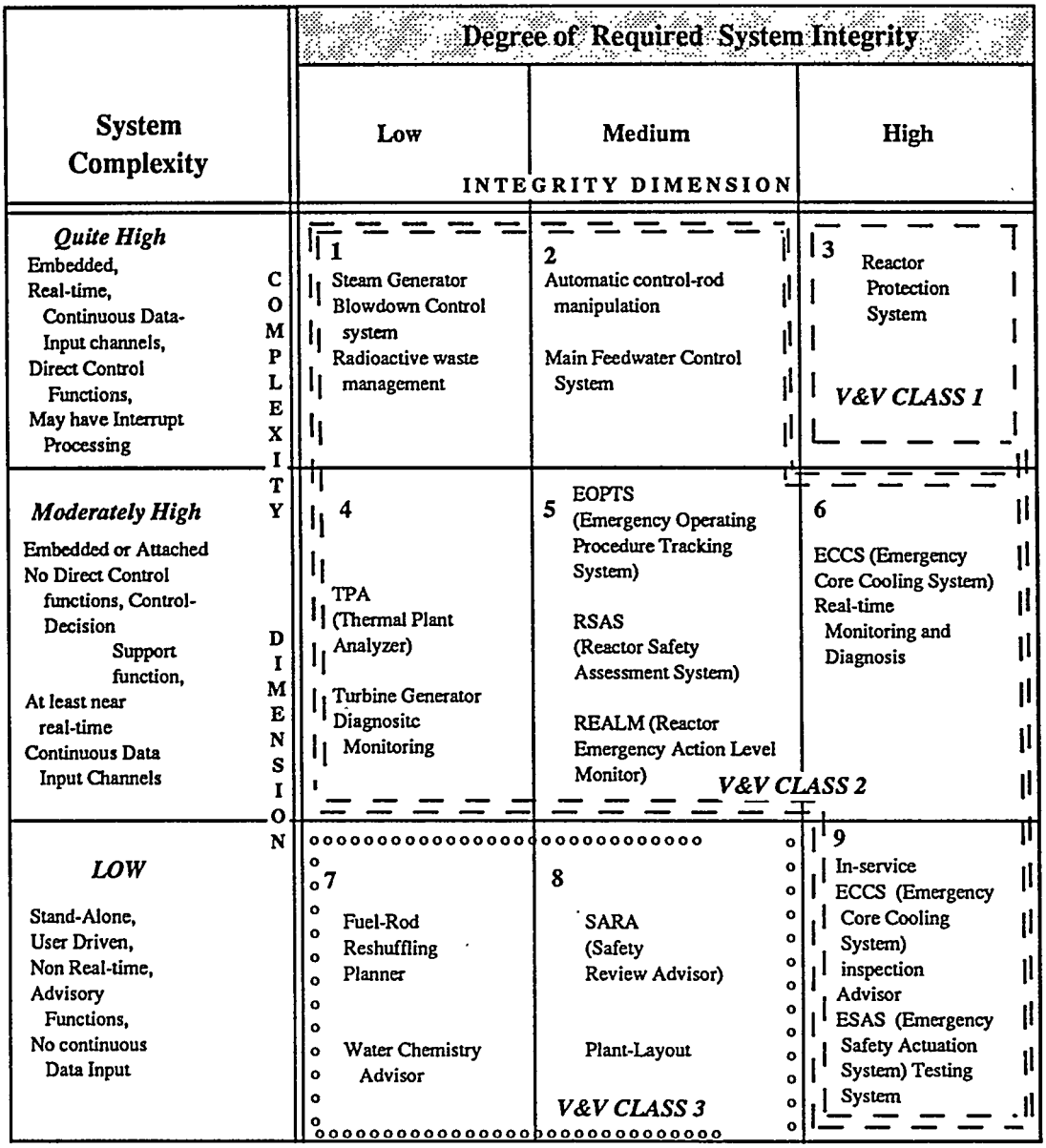


Figure 6.3-1 Three Classes of Stringency of Recommended V&V with Examples of Expert Systems for the Nuclear Power Industry

The intersection of the highest complexity and integrity levels (cell 3) defines the situation requiring the most aggressive V&V approach, and this is called Class 1. There is only one cell in this class. The lowest level of required V&V, Class 3, is for low complexity and low-to-medium integrity and includes two cells, 7 and 8. The remaining six cells (cells 1, 2, 4-6, and 9) are in the intermediate V&V class, Class 2.[8]

6.4 Quality Assurances for each V&V Class, as a Function of Life-Cycle Phase

The three classes of V&V have been defined in terms of the faults against which they offer assurances (cf. Section 6.2). Positive assurances are named here such that the particular fault is absent (e.g., "completeness" becomes the assurance that there are no incompleteness faults, "consistency" becomes the assurance that there are no inconsistency faults, and so forth). *The identification of the specific quality assurances provided by specific levels of V&V defined for well characterized systems is a major, and novel, contribution of these guidelines.*

The lowest level of required V&V, Class 3, reasonably will offer the fewest and simplest assurances. As the stringency of the class increases, additional and more difficult assurances are needed. At the highest level of required V&V, where the number of expected failures is largest and the need for zero failures is the greatest, the greatest number of quality assurances must be provided by the V&V activities.

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 6.4-1 through 6.4-3. Having identified the quality assurances to be provided, it is incumbent to specify *how* they are to be achieved. This is given by the third column, which lists the most appropriate V&V methods, as determined by the NUREG/CR-6018 report in this contract, to accomplish those assurances.

The major aspect missing from this Risk Management approach to quality assurance via V&V Classes is the quantification of the risk remaining after successful application of the indicated V&V methods. Unfortunately, the data concerning fault and failure probabilities and the V&V methods' "power" to detect faults, both fundamental to developing a quantitative estimation model, is almost completely lacking. For the present time, this *qualitative approach to Risk Management* must suffice.[9]

It is important to note that various aspects of the V&V methods -- e.g., practicality, cost-benefit, etc. -- have been previously addressed in the comprehensive review of V&V methods (see Volume 2). All of the recommended V&V methods have been rated on eight separate factors: four concerning the power of the method to detect various kinds of faults, and four concerning aspects of the ease-of-use of the methods. These eight factors were combined for each method to derive composite cost-benefit and effectiveness measures. These ratings should be consulted as a guide for assessing, say, the cost-benefit of a particular set of methods within a guideline package, or for guidance for substitution if particular suggested methods are believed to be impractical. One of the key principles used in selecting methods for guideline packages, however, was to recommend the more costly ones (to learn, use, run, or interpret) only for the more stringent V&V classes.

---

[8] The partitioning of these complexity-integrity cells into the three V&V classes was reviewed by both the USNRC and EPRI organizations. In addition, the classification scheme was presented for comment at several conferences and symposia, and there was no sustained concern about the present assignments.

[9] By being very specific about the assurances provided by each recommended V&V method, it becomes much easier to design experiments and collect data concerning the frequency of each of the assurance-flaws, and the efficacy of the various methods to detect them.

Table 6.4-1: Types of Assurances Provided by Each V&V Class and Appropriate V&V Methods: At the Requirement Stage[1]

V&V Class	Assurances Provided	Appropriate V&V Methods
Class 3	Limited assurance that the requirements meet the seven key criteria[2]; i.e., the requirements must be:	
	1) *complete* -- 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).	Formal Requirements Review (FRR) (Subject Matter Experts (SMEs) and users, editorial) Requirements Analysis
	2) *correct* -- their specifications accurately capture the client's intent and understanding, and they agree with known facts, where relevant	FRR (More SMEs than users) Requirements Analysis
	3) *clear* -- 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	FRR (editorial) Requirements Analysis
	4) *explicit* -- 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	FRR (requirements analysis specialist, recommend automating with a CASE tool) Requirements Analysis
	5) *consistent* -- none of the requirements logically conflict with any other requirement	FRR (requirements analysis specialist, recommend automating with a CASE tool) Requirements Analysis
	6) *testable* -- the means for determining whether the requirement has been satisfied is either explicitly stated or can be readily, and uncontroversially, determined	FRR (requirements analysis specialist) Requirements Analysis
	7) *separately identifiable* -- each separate requirement is allotted to a single specification and is prioritized; specifications do not contain compound requirements	FRR (editorial followed by requirements analysis specialist performing requirements "break up") Requirements Analysis

[1]This and the following twoTables are replicated in the Overview, Volume 1, as Tables 4.1.4.4 through 3.

Table 6.4-1 (Conintued)

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

[2] These are considered a minimum necessary set of the criteria concerning requirements put forth by a variety of key sources (e.g. DoD-2167A; NSAC-39; IEEE 7-4.3.2; Davis, 1990.)

Table 6.4-2: 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
Class 3	Limited assurance that the design meets the seven key criteria; i.e., the requirements must be:	
	1) *complete* -- identification of elements of the design purporting to implement each specific requirement.	Requirements Tracing
	2) *correct* -- limited evaluation that each design element, corresponding to a required functionality, is reasonable and would work if implemented.	Formal Design Review
	3) *clear* -- design specifications are expressed unambiguously, using language (or a formalism) that is familiar and understandable.	Formal Design Review
	4) *explicit* -- 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) *consistent* -- none of the design elements logically conflict with any other design element (e.g., units used, significance, precision, etc.).	Formal Design Review
	6) *interfaces consistent* -- 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) *database design sufficient* -- 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 Form

Table 6.4-2: (Continued)

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

Table 6.4-3 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
Class 3	Limited assurance that the implemented system meets these criteria; i.e., the system must be:	
	1) *complete* -- identification of system elements purporting to implement each specific requirement.	Requirements Tracing
	2) *unfounded functionality* -- identification of all system elements not traceable to a requirement.	Requirements Tracing
	3) *logic and control correct* -- limited evaluation of logic and control.	Formal Customer Review Functional Testing
	4) *data operations and computations correct* -- limited evaluation of data operations and computations.	Formal Customer Review Functional Testing
	5) *consistent* -- 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. Internal: no conflicts exist in passing data (variables) between modules of the system. External: no conflicts exist in passing data between modules and interfacing modules of external systems.	Formal Customer Review Functional Testing Scenario Testing
	6) limited evaluation of the adequacy of the Operational Concept.	Operational Concept Analysis Formal Customer Review Functional Testing
	7) limited evaluation of the adequacy of the Human-Computer Interface.	User Interface Inspection Formal Customer Review Functional Testing
	8) *database sufficient* -- the implemented database structure capably handles data storage/retrieval and manipulation.	Functional Testing

Table 6.4-3: (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods
Class 3	Limited assurance that system will perform correctly under random operating conditions.	Functional Testing
	Limited assurance that good software engineering practices are in use.	Random Testing
	Limited evaluation of the adequacy of the system's documentation (in-line, user, source code).	Formal Customer Review
	Limited assurance that the knowledge base is correct and sufficient.	Process Oriented Audits
	Limited assurance that the system complies with all levied standards.	Formal Customer Review

Table 6.4-3: (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods
Class 2	All Class 3 assurances.	All Class 3 methods
	Moderate 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.	Formal Customer Review Process Oriented Audits
	Moderate evaluation of the adequacy of the system's documentation.	Desk Checking
	Moderate assurance that the knowledge base is correct and sufficient.	Desk Checking

Table 6.4-3: (Continued)

V&V Class	Assurances Provided	Appropriate V&V Methods
Class 1	All Class 3 and Class 2 assurances.	All Class 3 and 2 methods
	Extended 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

6.5 Summary of the Cost-Benefit Aspects of this Risk Management Approach

The cost-benefit nature of this V&V approach is demonstrated by the simple relation diagrammed below:

	For A Particular Class of V&V	
	!	
	!	
The COST of Experiencing the negative consequences of predicted system failures, C1	---<<is to be>>--- balanced against	The COST of doing the V&V, C2, PLUS all the BENEFITS of experiencing few system failures, B1

Three values have to be estimated in the above relation, that is, C1, C2, and B1. These three values need to be formulated in comparable units, such as dollars or labor-hours. It should be noted that the benefit, B1, is not simply a function of C1. In addition to avoiding the cost of negative consequences, there are unique positive consequences which accrue to highly reliable systems and have definite if difficult-to-calculate value, e.g., being able to publicize a long period of operation without failures, gaining the reputation of having "trusted systems", or being able to attract good people and investment dollars.

However the values are estimated, the particular class of recommended V&V activity should be undertaken when the following inequality is true:

$$C1 \geq C2 + B1 \qquad \text{(Equation 6.5-1)}$$

This relation takes into account the fact that V&V costs are never insignificant. For Class 1 V&V, they can be very expensive. If C1 is less than the sum of the other factors, assuming the cost-benefit estimations are reasonable, then the desired inequality (of Eq. 6.5-1) can be obtained by redefining the system such that the Class of recommended V&V is reduced. Such a reduction can be achieved by two means:

(1) Changing the functionality or the operational context of the system such that the judged degree of Required System Integrity is decreased; or

(2) Reducing the complexity of the system.

7 GUIDELINES

This section contains the materials needed to actually accomplish the V&V of AI systems. The previous sections defined the context and issues for V&V. This section gives the actual description guidance.

The "roadmap" for accomplishing V&V is shown in Figure 7.0-1. One enters with the decision to perform V&V on a system. Such a decision is most effective when it is made at the very outset of a system-development project and can influence the requirements and design, thereby addressing major sources of error for relatively small costs as compared to the expense of error-detection and repairs made on an implemented system.

To learn about the V&V issues, Sections 1.0 through 6.0 of this report should be read. Otherwise, the first step to actual use of the V&V guidelines is to determine the developmental phase the system is in, as directed in the following section, 7.1. The next concern is what aspect of the system to test first. This is covered in 7.2. The third major decision is to determine how much V&V is appropriate for the particular system. The detailed questionnaire which will lead one to a reasonable choice is given in Section 7.3.

The results of the three decisions will permit one to select an appropriate "guideline package". The guideline package indicates 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 Section 7.4. The actual guideline packages, 16 in total, are provided, in the User's Manual, Volume 7.

The number of V&V methods chosen for use in a package varies as a result of the phase of development (increasing up to the implementation phase) and the V&V Class (more for systems requiring more stringent V&V). The number of methods varies little as a function of the aspect of the system to be tested. The reason that several V&V methods are needed instead of just one or two is because of the very large number of faults systems can contain and the fact that any particular V&V method is effective at finding only a subset of these faults. By using a number of methods, there is a pretty good chance of detecting most of the faults. Of course, the more V&V methods that are used, the more expensive and time-consuming is the V&V process. Every attempt was made, therefore, to put packages together so that increases in the cost and time of applying more methods are offset by increases in higher quality by greater assurance of detection of faults. The actual amount of V&V performed must be determined by weighing its cost against the benefits obtained. All of the methods recommended are among the most effective and cost-beneficial of the 153 methods evaluated early in the project (cf. Volume 2).

The guideline packages discussed in Section 7.5 specify what V&V methods should be used for a particular situation, what assurances they provide, the teams needed to accomplish the various methods, the quality measures to be obtained for each method, and the decision criteria for accept/reject of the system based on these measures. The packages do not instruct how to perform the specific V&V methods; a literature reference where the user can find such information is provided. There are some methods whose value is so great, and for which the references are really not adequate, that actual step-by-step procedures are provided. Eleven such detailed V&V procedures were provided in the User's Manual, Volume 7, and are discussed briefly in Section 7.6.

7.1 Determining the Development Phase

Four phases in system development are identified in the simplified view taken here. These were discussed in Section 4. They are Requirements, Design, Implementation, and Maintenance. The characteristics of these phases, and the object of V&V, are shown in Table 7.1-1. If there is any doubt what phase of development the system is in, this table should be consulted to determine the phase. This is one of the three decisions necessary to select the appropriate V&V guideline package.

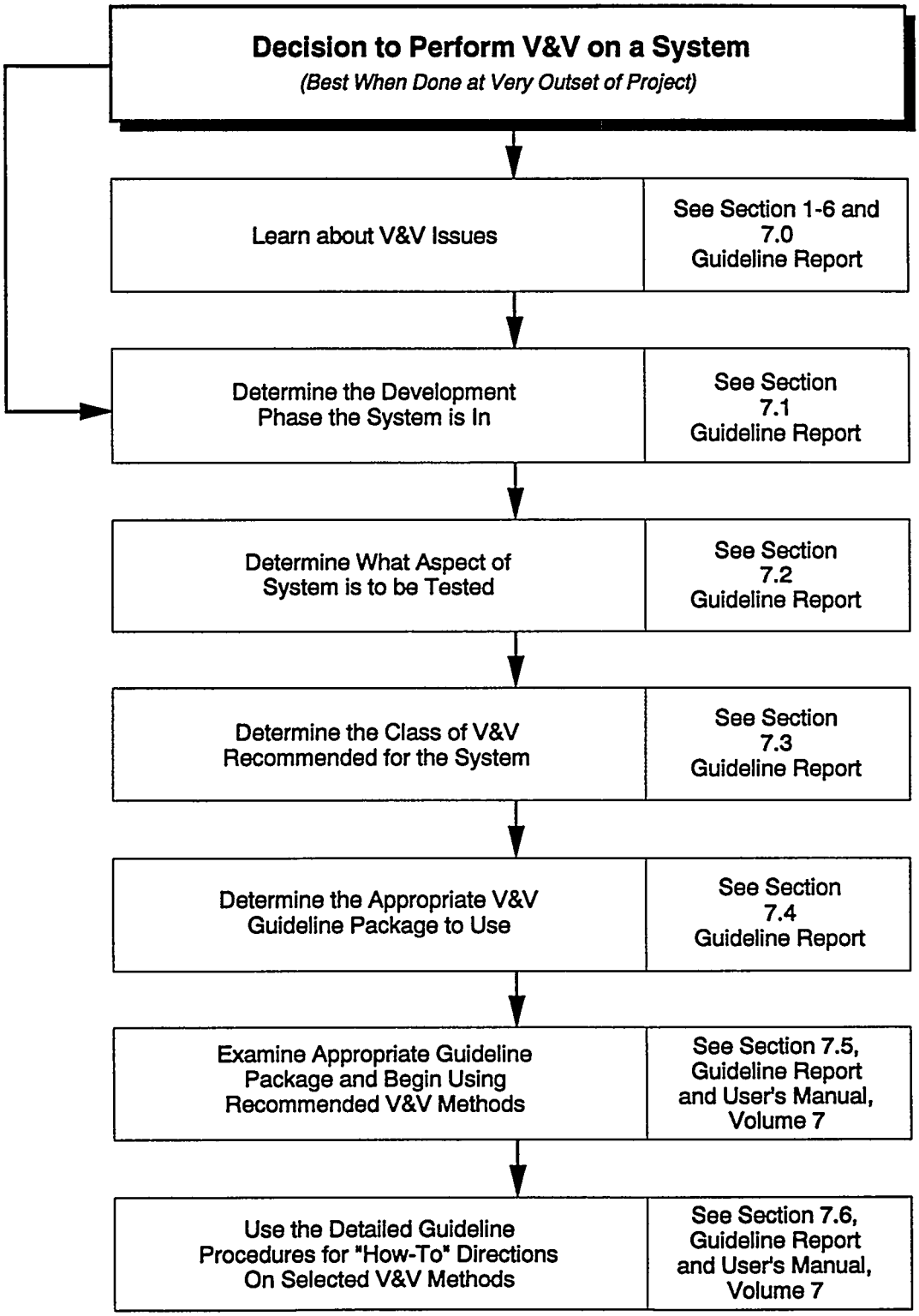


Figure 7.0-1: Roadmap for Using the V&V Guidelines

Table 7.1-1 Description of developmental phases and the proper object of V&V in each

Development Phase	Description	Object of V&V
Requirements Generation	The system is in the earliest stage of development. Exactly what the system is supposed to do is being formulated. A careful writeup of these desired features will compose the Requirements Document. Design has not yet begun.	The Requirements Document
Design	A Requirements Document (or "understanding") exists. Development of an architecture and set of hardware and software processes to accomplish these requirements is under way. The completed design will be described in the Design Document. Implementation has not yet begun.	The DesignDocument *(and Requirements Document)*
Implementation	A system design exists in some form. Coding of the design in some language (or using some code-generation tool) has begun. The result of this phase is a working system on some computer platform. The system has not yet been installed for the customer.	The System Source Code *(and Design Document and Requirements Document)*
Maintenance	The system has been installed at the customer site and has begun to be used. Some problems or deficiencies have been noted, or ideas for extension of the system function have been proposed. Approval has been granted for these changes. They have been described in some form (e.g. Modification Request), and design for implementing them was developed (and mini-V&V actions were taken for both steps). The changes have been implemented and the modified source code is ready for maintenance V&V.	The Modified System Source Code *(and Modification Request)*

7.2 Determining the System Aspect To Be Tested

Sections 2.2 and 2.3 of this report discuss the various types of AI systems and their components. The total set of system components (including the overall system) were sorted into three classes, with the V&V approach appropriate for each. The result, previously shown in Table 2.3-1, is repeated here as Table 7.2-1 for completeness.

The Type 1 component comprises the various forms of declarative knowledge structures in AI systems, the so-called *knowledge base.* 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, or all associated tools and utilities used with the system in its development, which are very generic and highly reusable. These elements can be easily used on large numbers of different systems with no or minimal changes to their code. The type 3 component includes everything else, including the total overall system. The differing V&V philosophies lead to different sets of specific V&V methods making up the guideline packages, but there is not a great deal of difference in the actual numbers of methods recommended as a function of component type.

The decision to be made at this point is which type of system component is to be selected for V&V review at this point. This is the second of the three decisions needed to select a V&V guideline package.

7.3 Determining the Appropriate Class of V&V

The third decision of determining the appropriate V&V class is guided by a three-part questionnaire shown in Figure 7.3-1. This questionnaire provides a practical means of assessing system complexity and required integrity. These were the two factors identified in the previous section of this report (section 6.3) as being primary determinants of the stringency of V&V needed for a system. The questionnaire provides a way to assess one's own system on these factors. Part 1 of the questionnaire leads to a determination of complexity, Part 2 leads to an assessment of required integrity, and Part 3 provides a table to indicate which of three levels of V&V is recommended based on the first two parts.

Whenever a class of V&V is decided upon, the user should evaluate the definite costs associated with using the set of V&V methods appropriate for that class (and the particular system and life cycle phase), as discussed previously (see Section 6.5). As was suggested, a system for which the V&V appears too costly can either not be built or, more practically, have its requirements changed so that its complexity or required integrity are decreased, thus decreasing the needed stringency of V&V.

7.4 Determining the Appropriate V&V Guideline Package

The previous three subsections provide the information necessary to select the recommended set of V&V methods for this situation, using Table 7.4-1. The results of Section 7.1 identify the phase of system development. Systems in the Requirements phase should use part (a) of Table 7.4-1; those in the Design phase should use part (b); and those in Implementation should use part (c). A system in the Maintenance phase should use Guideline Package P.

The results of the decision of the component type next to be tested, provided in Section 7.2, indicate which row should be chosen. The results of the questionnaire (Section 7.3) determine the appropriate column. Within the chosen cell is a package identifier (e.g., the V&V of a Class 2 implementation of a highly reusable component should use Guideline Package K). The full set of Guideline Packages are found in the User's Manual, Volume 7.

Table 7.2-1 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 **Declarative Knowledge Structures:** • Rule Bases • Frames • Class Hierarchies • Objects Stored in Databases	X		
Type 2 **Highly Reusable Components:** • Tools & Utilities • Some Interfaces • Some Functions • All Commercial Software Packages used		X	
Type 3 **Mostly Non-reusable Components:** • Overall System • Most Interfaces • Conventional Databases • All Remaining Application-Specific Code			X

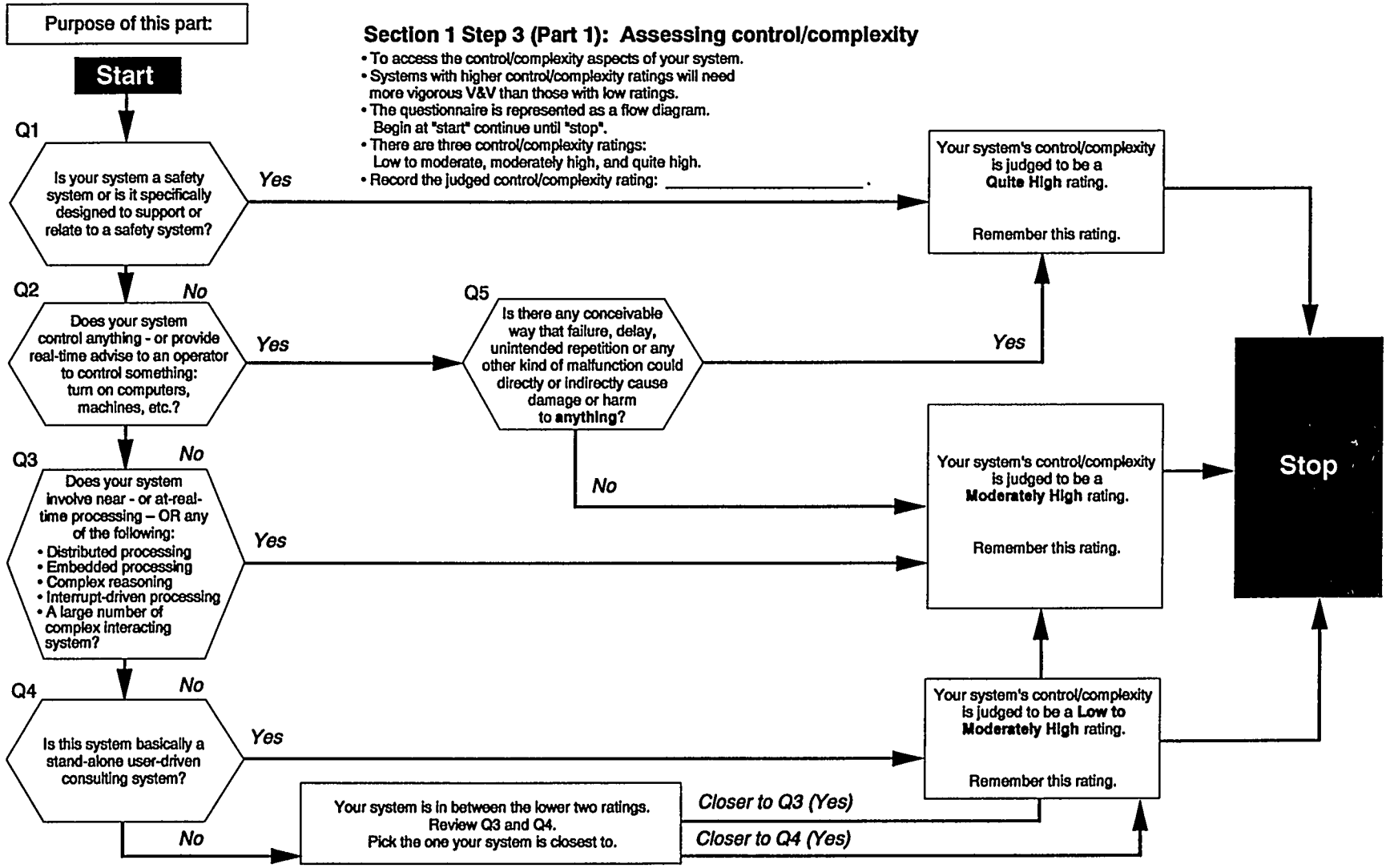


Figure 7.3-1 Three-part questionnaire to determine recommended V&V Class.

Figure 7.3-1 Three-part questionnaire to determine recommended V&V Class (cont'd.)

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

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 7.3-1 Three-part questionnaire to determine recommended V&V Class (cont'd.)

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.

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

Table 7.4-1: Identification of the 15 Guideline Packages Assigned to the Three Types of System Components and V&V Class for the Requirements Phase (a), the Design Phase (b), and the Implementation Phase (c).

(a) Requirements Phase

Type of System Component	V&V CLASS		
	Class 3	Class 2	Class 1
Knowledge Structure	Package A	Package B	Package C
Highly Reusable	Package A	Package B	Package C
Other and System	Package A	Package B	Package C

(b) Design Phase

Type of System Component	V&V CLASS		
	Class 3	**Class 2**	**Class 1**
Knowledge Structure	Package D	Package E	Package F
Highly Reusable	Package D	Package E	Package F
Other and System	Package D	Package E	Package F

(c) Implementation Phase

Type of System Component	V&V CLASS		
	Class 3	**Class 2**	**Class 1**
Knowledge Structure	Package G	Package H	Package I
Highly Reusable	Package J	Package K	Package L
Other and System	Package M	Package N	Package O

Maintenance phase systems should use Guideline Package P.

7.5 The V&V Guideline Packages

The 16 V&V guideline packages, A-P, referred to in Table 7.4-1, are actually provided in the User's Manual, Volume 7. Refer to the first of these, A, found in Appendix A, for the following explanation of the columns of information about the listed V&V methods.

The user should look here through the whole guideline package and note that it has identifiers at the top telling the name of the package, its life cycle phase, V&V Class, and Component Type. The package is in the form of two tables. The first table lists all the V&V methods that are recommended for this situation. The second table, on the last page of the guideline package, tells how the user should plan on applying these methods (this is a very simple plan in the case of Guideline Package A).

In the guideline package there are eight columns. The first row of the guideline package contains notes about the contents of the columns in the guideline. The first column gives an ID number to the V&V method described in that row. This ID number is referred to in the guideline plan table. The column labeled "Method" lists the names of the recommended V&V methods. The third column, "Type", identifies the method as either Static, which requires no actual execution of the code, or Dynamic. The fourth column labeled "Assurances" lists all of the assurances of quality afforded by application of this method. The fifth column, "Agents" describes the qualifications of personnel for performing the various methods. The "Intensity" column, column six, describes the amount of effort required in applying the various V&V methods, usually in one of three (often qualified) terms: Limited, Moderate, or Extended. The legend, "Measures" describes the quantitative measures or judgments that have to be taken or made during application of the method. Most methods use the same measures. The measures are typically in terms of the number of some type of problem such as the number of "show stoppers" discovered in applying the method. The seventh column, "Criteria/Action", establishes what the thresholds are for the defined measures for three possible outcomes. These outcomes are (unconditional) Accept, Conditional Acceptance, and Reject. In the case of the first or third of these outcomes the V&V stops for the method being applied. If the outcome is "Reject", then a decision has to be made by the development team whether V&V should continue or whether the system should be redesigned or re-implemented. If the decision is conditional acceptance, then the problems found are to be fixed and the method applied again. The last column "Ref" contains the reference for the recommended methods. If the entry begins with an "R", then it is to be found at the end of the recommended V&V methods and before the guideline plan (and also in the numbered references in Section 8). If the entry begins with a "P", then it is a Guideline Procedure (cf. Section 7.6).

In the case of Guideline Package A, only two V&V methods are recommended (sometimes there are as many as 19). The two methods are Requirements Analysis and Formal Requirements Review. According to the plan, they are to be performed in the order they are given. For Requirements Analysis, which is suggested to be performed by a team of a V&V person, a requirements analyst and an editorial expert, the system component is to be accepted as being of sufficiently high quality, passing the V&V qualification, if (1) there are no deficiencies in critical requirements, (2) there are four or fewer deficiencies in the non-critical requirements, and (3) the percentage of non-critical requirement deficiencies is equal to or less than 8%.

7.5.1 Standard Decision Thresholds

The thresholds used for all of the V&V methods fall into two problem classes: (1) those concerning requirements, and (2) the rest, which concern other aspects of system quality. A standard set of decision values has been used for the guideline packages, and these are shown in Table 7.5.1-1. The values are given for the thresholds for the two problem classes, for the three types of decisions, and for each of the V&V Classes. Four measurement thresholds are given for the problem class concerning requirements, while three measurement thresholds are given for the problem class concerning system quality.

As stated above the thresholds are based only on engineering judgment. The values in Table 7.5.1-1 are derived from the following kinds of considerations given below. Three thresholds are intended to provide reasonable guidance and should not be viewed as absolutes. The rationale for these thresholds included the following:

- The thresholds should decrease as the stringency of V&V increases (from Class 3 to 1).
- Not even a single Show Stopper (or Critical Requirement deficiency) should be accepted in any software system.
- The Conditional Accept decision is the middle ground where more errors are acceptable since they will be fixed and V&V performed again.
- Too many detected errors, however, imply fault-prone programming practices, so that, at some point, one should reject under the assumption of pervasive impoverishment.

An example is the rationale for 4 as the threshold for conditional acceptance for Class 3, for the Critical Requirements and major problems metrics, is as follows:

- 1 or 2 of these significant errors might be attributable to careless developmental testing (which presumably precedes V&V), particularly since Class 3 systems are probably the lowest budget systems and practices will often be lesser.
- For the limited developmental capability expected to be associated with Class 3 systems one can envision that some major errors are correlated, in that a deficiency in some high level design or conceptualization process could give rise to, say, a couple of specific conceptually- related errors. This could account for up to 4 such problems.
- There being more than 4 major errors detected, remembering that V&V is in addition to all normal developmental testing, implies such poor quality procedures that the whole system becomes suspect.

- The near-exponential reduction of the allowable major errors for both requirements and quality thresholds for Class 3 to Class 2 and again to Class 1 is a reflection of the presumed exponential growth in complexity and required integrity.

7.5.2 Cost-Benefit Tradeoffs

The guidelines for Class 3 systems presume that exhaustive V&V examination is neither warranted nor affordable. Yes, the economy-minded regimen recommended here might not detect all system-crashing problems. But the impact of system malfunction is so limited that it is believed to be a justified risk. Class 2 systems, on the other hand, can produce an expensive and highly undesirable impact upon failure, and a considerable stricter and more

Table 7.5.1-1 Standardized decision thresholds (based on engineering judgement) for Guideline Packages for the two sets of measures (concerning Requirements and concerning System Quality), for the three Classes of V&V (Class 3, Class 2, and Class 1), and for the three decisions (Accept, Conditional Accept, and Reject)[1]

V&V Class	Decision	Logic[2]	Concerning Requirements (Req.s)				Concerning System Quality		
			# Critical Req.s Deficient (CRD)	# Non-critical Req.s Deficient (NRD)	% NRD	% Unintended Functions (%UF)	# Show-Stoppers (SS)	# Major Problems (Maj)	# Minor Problems (Min)
Class 3	Accept	And	= 0	≤ 4	≤ 8	≤ 6	= 0	= 0	≤ 4
	Conditional Accept	And	≤ 4	≤ 6	≤ 12	≤ 10	≤ 1	≤ 4	≤ 12
	Reject	Or	> 4	> 6	> 12	> 10	> 1	> 4	> 12
Class 2	Accept	And	= 0	≤ 2	≤ 4	≤ 4	= 0	= 0	≤ 3
	Conditional Accept	And	≤ 2	≤ 4	≤ 8	≤ 5	≤ 1	≤ 2	≤ 8
	Reject	Or	> 2	> 4	> 8	> 5	> 1	> 2	> 8
Class 1	Accept	And	= 0	≤ 1	≤ 1	≤ 1	= 0	= 0	≤ 1
	Conditional Accept	And	≤ 1	≤ 2	≤ 4	≤ 3	= 0	≤ 1	≤ 4
	Reject	Or	> 1	> 2	> 4	> 3	> 0	> 1	> 4

[1] For example, the conditional accept threshold concerning Requirements for Class 2 is: CRD ≤ 2 and NRD ≤ 4 and %NRD ≤ 8 and %UF ≤ 5. The reject threshold for Class 1 is: CRD > 1 or NRD > 2 or %NRD > 4 or %UF > 3.

[2] When logic is "and" all of the metrics must be true to take that decision; when logic is "or" any of the metrics being true justifies the decision.

expensive course of V&V is indicated. For the rare Class 1 system whose failure can have truly drastic consequences, the exceptionally careful and thorough V&V approach recommended provides a most comprehensive and multi-faceted course, admittedly at great expense. Such costs, however, are believed to be justified by the much higher assurance that critical failure-inducing faults will be hunted down and eliminated.

As has been stated previously, this report only provides reasonable guidance, and the users should modify the recommendations based on their own particular situations.

7.5.3 Procedures for Determining Measures

The measures which relate to requirements make a distinction between "critical" and "non-critical" requirements. The system-quality measures distinguish among "show-stoppers", "major" problems, and "minor" problems. This section provides suggested procedures to follow for arriving at a reliable understanding of the way these relative terms should be interpreted in the context of a particular system.

Concerning the criticality of requirements measures, it is assumed that a requirement document is available, and it has not yet been classified into critical and non-critical requirements. First, in most Federal Government requirements documents (or Statements of Work), it is possible to separate true requirements from the "goals" by examining sentences for the presence of "shall" and "will": sentences with "shall" are real requirements; sentences containing "will" are only objectives to strive for and should be ignored for the purposes of the ratings of requirements as critical and non-critical. Each true requirement should be examined carefully, asking the following question: "On a 5-point scale, with 1 being lowest and 5 highest, what is the true necessity of this requirement?" In asking the question, imagine that only half the budget and half the time is available for system development but ***some*** system capability must be delivered. What could be sacrificed so that there is still an acceptable base-system capability? After rating all the requirements, put them in order of their rating from 5 down. Read down the list until an area of requirements are found which are quite reasonably in the "can-live-without" category, perhaps at scale-value 2-3. Shuffle the requirements around on the ordered list around this change-over point until there appears a partition point above which the requirements really all seem to share a strong necessity, and below which they seem less strong. This is then a good decision point for defining "critical" and "non-critical."

Unintended functions are identifiable as any elements in the design or code whose function cannot be justified on the basis of the requirements. Each separate occurrence of unintended function should be noted.

Concerning the quality measures of "show-stoppers", "majors", and "minors", a different procedure must be used, although it is useful to have first ranked the requirements as above. In this case the focus is on the consequences of a discovered problem, that is, how much rework is necessary to correct the problem. A "show-stopper" is something which implies something dreadfully wrong about an extremely important system feature (or requirement) which pervades through the whole design or implementation, such as a fundamental decision to use a faulted architectural approach to achieve functional diversity in a safety-critical control system.

Show-stoppers can only occur as a broad deficiency, such as in the overall architecture, in a repeatedly-used algorithm, in the choice of a complicated data-structure that strongly influences how many other features are developed, etc. Further, their correction must involve a great deal of work. Finally, the strongest show-stoppers are those whose presence makes one lose confidence in the rest of the system: "If the developers made this kind of error, they could have done any number of additional shoddy things." Show-stoppers can be discovered by virtually any method, but they are much more likely to emerge with reviews and general inspections. Whatever methods are used, each discovered error should be considered carefully, asking the questions, "Is this error possibly evidence of a general

pervasive conceptual deficiency?" and "Does this error, together with other errors, suggest the emerging outline of a show-stopper?"

In contrast to show-stoppers, "major" errors are much more constrained in effect and needed correction. They will require considerable re-work to correct, but nothing like that for show-stoppers. There is seldom the strong loss of confidence that accompanies detection of show-stoppers. Finally, "minors" are exactly that, problems ranging from nits to clear errors but all needing only a moderate effort to correct.

In preparation for the classification of the problems found by the various methods, it is quite useful to generate and keep in mind potential examples of each of the three classes, particularly the show-stoppers and majors. Top requirements from a requirements ranking might suggest what some truly disastrous problems might be, but it is always safe to imagine how the application architecture could be designed so as to provide a minimally effective solution.

7.5.4 Selection of Methods for Inclusion in the Guideline Packages

Of the 153 V&V methods reviewed and evaluated in this project (see Volume 2), 32 were chosen to make up the 15 Guideline Packages A through O.[14] A major factor guiding the selection were the cost-benefit and effectiveness ratings of the methods determined in that review for each phase of the life cycle as a function of the Class of V&V required. The highest rated appropriate methods were chosen.

A second major factor in the selection process was the assumption that most of the systems for which these guidelines would be used can be characterized as *Decision Support Systems.* Such systems involve a great deal of interactivity between the user and the software: the system does a lot of sensing, measuring, data storage, data retrieval, and algorithm execution, and then presents interim information to the user. The user makes decisions based upon the information provided, usually complicated ones requiring reasoning based on experience. In such systems the allocation of decision and processing elements between the user and system, and the actual style of presentation and interaction is crucial to system effectiveness. Accordingly, methods which assess the quality of the user interface and the style of interaction -- the *concept of operations* -- are crucial to demonstrating the quality of these systems.

A third major selection factor was the type of component: knowledge bases require special knowledge/syntax checking procedures while reusable components need certification-type methods. The complete overall system needs testing from a wide variety of perspectives.

Table 7.5.4-1 shows the use of the 32 chosen methods in the 15 Guideline Packages A-O. Of the nine methods used for a third of the packages (a total of 5 or more times), as shown by the row total in the last column, *Operational Concept Analysis* and *User Interface Inspection* both are justified in terms of the second selection factor described above. The remaining six most frequently used methods were selected on the basis of the first principle, their evaluated ranking: *Requirements Tracing, Formal Customer Review, Functional Testing, Random Testing, Robustness Testing*, and *Regression Testing*. The remaining 24 methods are justified in terms of the third factor, the component being tested, with the effectiveness ranking being used to select among equally appropriate candidates.

According to the assignments of Table 7.4-1, every third Guideline Package is in the same V&V Class; that is, **A, D,** and **G** are for Class 3; **B, E,** and **H** are for Class 2; and **C, F,** and **I** are for Class 1, etc. From the column totals in Table 7.5.4-1 (given by the last row), one can see that, except for the Requirements phase, the difference in

[14] The sixteenth package, **P**, is for the maintenance situation and provides guidance for selecting one or more of the other packages **A-O**.

Table 7.5.4-1 Occurrence of the 32 recommended V and V methods within the V and V Guideline Packages A-O

V&V Method	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	Total
Requirement Analysis	x	x	x													3
Formal Requirements Review	x	x	x													3
Formal Verification			x													1
Requirements. Modeling/animation			x													1
Requirements Tracing				x	x	x	x	x	x	x	x	x	x	x	x	12
Formal Design Review				x	x	x										3
Design Modeling/Animation						x										1
Desk Checking								x	x							2
Operational Concept Analysis		x	x	x	x	x					x	x	x	x	x	10
Data Interface Inspection												x		x	x	3
Database Analysis					x	x										4
User Interface Inspection		x	x	x	x	x										10
Algorithm Analysis																1
Process Trigger/Timing Analysis																1
Failure Mode/Effects Analysis						x			x							3
Knowledge Engineering Analysis						x		x	x							4
Automated Anomaly Testing																2
Automated KB Syntax Checking							x	x	x							3
Automated KB Semantic Checking Process							x	x	x							3
Process Oriented Audits																2
Software Practices Review					x	x										6
System Engineering Review						x										2
Formal Customer Review							x	x								6
Functional Testing							x	x								9
Structural Testing								x								1
Benchmarking																2
Beta Testing								x								3
Boundary Testing								x								4
Random Testing							x	x								9
Robustness Testing																5
Validation Scenario Testing																2
Regression Testing								x								6
Total	**2**	**4**	**6**	**4**	**7**	**10**	**6**	**10**	**12**	**6**	**10**	**11**	**6**	**14**	**19**	**127**

number of methods recommended goes sharply up from Class 3 to Class 2, with a smaller increase from Class 2 to Class 1. This is because the Class 2 systems, by their rating of moderately high complexity and medium required integrity will be considerably more in need of V&V than the simple, low-complexity and low-integrity, Class 3 systems. Class 1 systems are a smaller increase over Class 2 in the complexity factor.

Examination of the methods recommended for V&V of the knowledge base (KB) -- Packages **G, H,** and **I** -- reveals that dynamic methods are included in addition to the static ones. Static methods test many aspects of the KB, and their great advantage is that they don't need the system to be running; several can actually *prove* that certain faults are or are not present, thus obviating the need for relentless structural testing. However, static methods cannot determine whether there are flaws due to the interaction among the KB, the inference engine, and other components. In any event, no system should be accepted without some kind of random or stress-testing using dynamic methods.

One last aspect of Table 7.5.4-1 may need explanation; that is the quite substantial number of methods needed for Packages **N** and **O** (testing the overall system for Class 2 and Class 1 respectively). In Guideline Package **N**, 14 methods are recommended for testing Class 2 overall systems, 8 static analysis techniques and 6 dynamic testing ones. In Package **O**, there are 19 methods, 13 static analysis techniques and the same 6 dynamic testing ones as in Package N. In justification, it is first of all important to realize that in all of the packages the methods are ordered in a manner to minimize the amount of overall testing. In particular, need for the highly labor-intensive and costly dynamic methods is reduced by employing the various static methods first. Even the order of static methods is designed to reduce the amount of testing -- and timely repairs -- by putting the specialized analysis methods before the more substantial reviews. Thus, the whole is rather less than the sum of the parts.

A second aspect to consider is the long-term cost-benefits of following the full recommended set of methods. If all the methods are completely and carefully employed, in the order indicated -- with problems fixed and regression testing invoked after every method -- the system will almost certainly have as high a quality level as is possible given today's tools and techniques, certainly a level of quality well beyond most initially-developed systems. One must remember that the reason there are so many V&V techniques is because there are so many different ways software can have problems. The bulk of the methods recommended here are highly effective and specialized techniques, with high capability for detecting specific kinds of important errors. For software with an expected life of at least a year, if not 10, the old adage "pay now or pay later" is absolutely true, except that it will cost two to five times as much (or more) to fix a system in the field as it will when it still is under development. And that is not counting the errors one typically introduces in field repairs that have to be fixed later. Therefore, effective V&V testing throughout development is worth nickels (of maintenance) on the penny (of development).

A third point to be taken into account concerning the high number of system testing methods for Class 2 and Class 1, is that for systems which are marginally just into Class 2 or Class 1, it would be quite reasonable to decrease somewhat the number of methods -- since the set is designed for systems which are solidly in the middle of these classifications. For example, systems which have very little database activity and are marginal could drop the Database Analysis method; by similar reasoning, other methods could be eliminated.

Please note that the guidelines and procedures themselves are presented in the User's Manual, Volume 7.

7.6 Selected Guideline Procedures

As discussed above, 32 methods were used in the Guideline Packages. Eleven of the more important of these were chosen, and very detailed step-by-step procedures were written to provide specific guidance in the use of these procedures (technical references are given for the remainder). The sections of these procedures are described below. A sample template for a guideline procedure, labeled "P0.0" is given in Appendix B; the discussion below will reference this sample template. Also given in Appendix B is a full example for one of the shortest procedures, "Random Testing." The full presentation of all 11 procedures is provided in the User's Manual, Volume 7.

Referring to the sample template procedure (P0.0), the sections are discussed from top to bottom, staring with "When to use..." at the very top.

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 the Program Management approval is recommended prior to commencement.

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

EXECUTION - For shorter guideline procedures (such as P6.0 Random Testing, found in Appendix B) the execution part of the procedure lists the (numbered) steps needed to accomplish the procedure. For guideline procedures that are very lengthy (such as P1.0, Requirements Tracing), the execution part will be broken down into subparts. At the highest level, there will be a flow diagram shown consisting of the rectangles shown at the bottom of the P0.0 sample template in Appendix B. Single-line rectangles contain all of the information about an aspect of the procedure, while rectangles with the double-lines on the left and right indicate that there is much more information on a continuation page.

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 endnotes which are located at the end of the entire Procedure (e.g. after P1.8 for Procedure P1.0).

INTERRUPTS: It is possible that during execution of a particular step of the V&V method this activity should 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 [], since the lower levels are textual, not graphical.

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.

ACCEPT/REJECT CRITERIA - for top-level guideline procedures there are three possible results: ACCEPT, CONDITIONALLY ACCEPT, and REJECT. If ACCEPT, fix any problems, and go to the next action (if this is the last method, stop). If CONDITIONAL ACCEPT, then fix the problems, repeat the action, and ensure that the problem did not recur. Finally, if the decision is REJECT, stop all activities and discuss problems with the program manager.

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 of 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.

8 CONCLUSIONS

This document presents the rationale for and description of guidelines for the verification and validation (V&V) of artificial intelligence (AI) software systems. These guidelines, presented in full in the Volume 7 User's Manual, form the final product of this project, which is jointly funded by the U.S. Nuclear Regulatory Commission (USNRC) and the Electric Power Research Institute (EPRI), and reflect the integration of all previous activities. Since AI conventional software components are also addressed by these guidelines, the guidelines are directly applicable to the V&V of any conventional software.

A Cost-Benefit Risk Management Approach is the underlying philosophy used in developing these guidelines. Systems with a higher risk of negative consequences from failed or incorrect operation have correspondingly more powerful and effective recommended V&V methods, which are also necessarily more expensive.

These V&V guidelines were developed by a variety of reviews, research, analysis, development, and presentation activities involving both the nuclear industry and the software testing field. During several activities, two actual nuclear power AI systems, provided by the USNRC and EPRI, were used as a test bed to evaluate V&V techniques. Along with the extensive industry surveys and evaluation of available V&V techniques, new innovative techniques were developed and tested in a first-of-a-kind behavioral experiment with 20 nuclear and software industry participants. It should also be noted that relevant U.S. and international standards and regulations and their associated professional society activities were monitored and included in this project.

It should be noted that the guidelines have not yet been validated by field experience. They are recommendations based upon many careful analyses, the best available empirical data concerning V&V methods (scanty), and considered judgement. Every attempt was made to factor into the guidelines a wide variety of relevant considerations. However, users should take into account the costs and the benefits of the recommended procedures as they relate to the particular systems under test and modify or adapt them according to their best judgement.

During the course of this project, the following six important AI V&V issues were identified and incorporated into the development of these V&V guidelines:

1. V&V needs to be incorporated into all phases of the software developmental life cycle,

2. Complexity and required integrity were identified as the two key differentiating factors among AI systems,

3. Software was classified into three V&V classes of increasing stringency based on the above two factors,

4. More stringent V&V classes should provide greater assurance of the absence of faults,

5. Each AI system component may have different V&V requirements, and

6. The merits of each V&V technique needs to be evaluated within the context of the three V&V classes, the life cycle phase, and the type of component.

The guideline packages were developed on the basis of best engineering judgement for 27 combinations of the three determinants for V&V: life-cycle phase (Requirements, Design, or Implementation), type of system

component (Declarative Knowledge Structures, Highly Reusable Components, or Mostly Nonreusable Components), and the three V&V Classes (based on a matrix of software complexity and required integrity). A separate package was developed for software system maintenance. Due to multiple applications, there are a total of 16 V&V guideline packages. For 11 of the most important V&V techniques delineated in these packages, supplementary detailed guideline step-by-step procedures are also provided.

The guideline packages emphasize extensive early powerful static testing techniques followed by fewer dynamic testing methods. Those techniques which have been determined to have the highest cost-benefit and effectiveness in identifying errors were chosen for these guidelines. In addition, methods which were proven to be extremely effective in the earlier experiment were selected for inclusion. A mix of static and dynamic techniques were incorporated so as to offer the maximum possible coverage of different types of software system faults. Finally, the importance and power of computer-automated tools (e.g., CASE tools) for V&V is highlighted as a critical component in performing V&V.

The process for selecting V&V methods in these guidelines starts with the user identifying the life cycle phase and system component type which is intended to be subject to V&V. This is accomplished with the aid of specific descriptions of each phase and component type. Then, using a four page questionnaire, the user determines the V&V Class based on the assessment of system complexity and required integrity. Finally, a table is provided which correlates life cycle phase, system component type, and V&V Class to a specific V&V package which is identified by an alphabetic letter.

Each V&V guideline package delineates specific methods, the level of assurance which they provide, the size and composition of the V&V team to be used with that method, the intensity of the V&V activity, the performance measures to be taken, and the best judgment as to recommended thresholds for making the decisions of accept, conditionally accept, and reject.

As previously discussed, the V&V guideline packages may refer to one or more of 11 specific procedures which are included in the User's Manual (Volume 7). These procedures present detailed step-by-step instructions, in a general flow chart and block diagram format, of how to perform a specific V&V method. These procedures include such aspects as planning, program manager approval, setup, execution, analysis, accept/reject criteria, reporting, review, and output/results.

The V&V guidelines, applicable to both AI and conventional software systems, represent the culmination of a three year project which gathered extensive V&V information, created and tested both established and new techniques using real nuclear industry AI systems, and developed innovative methods for system V&V classification. These guidelines offer a state-of-the-art, cost-effective method for performing V&V with the broadest possible spectrum of software that may be encountered throughout the nuclear industry.

9 REFERENCES

Alford, M., *SREM At The Age of Eight: The Distributed Computing Design System,* Computer, 18 (4), pp. 36-46, 1985.

ANSI/IEEE ANS-7-4.3.2-1982, *Application Criteria for Programmable Digital Computer Systems of Nuclear Power Generating Stations,* American Nuclear Society, 555 North Kensington Ave., La Grange Park, Illinois, 60525, July 6, 1982.

ANSI/IEEE 1012-1986, *Software Verification and Validation Plans,*, American Nuclear Society, 555 North Kensington Ave., La Grange Park, Illinois 60525, November 14, 1986.

ANSI/IEEE 1042-1987, *Guide to Software Configuration Management,* American Nuclear Society, 555 North Kensington Ave., La Grange Park, Illinois, 60525, September 12, 1988.

ASME NQA-2a-1990 Part 2.7, *Quality Assurance Requirements of Computer software for Nuclear Facility Application,* The National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, 1990.

Barnes, M., P. Bishop, B. Bjarland, G. Dahll, D. Esp., J. Lahti, H. Valisuo, and P. Humphreys, *Software Testing and Evaluation Methods (The STEM Project),* Technical Report, OECD Halden Reactor Project, HWR-210, The Institutt for Energiteknikk, Halden, Norway, May 1987.

Beizer, B., *Software Testing Techniques,* Van Nostrand Reinhold, New York, New York, 1990.

Bernard, J.A., and T. Washio, *Expert System Applications Within the Nuclear Industry,* American Nuclear Society, La Grange Park, Illinois 60525, 1989.

Blanchard, B.S., and W.J. Fabrycky, *Systems Engineering and Analysis,* Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

Boehm, B.W., *Software Engineering Economics,* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

Boehm, B.W., *Software Risk Management,* Academic Press, New York, New York, 1990.

Boehm, B.W., *A Spiral Model of Software Development and Enhancement,* IEEE Computer, pp. 61-72, May 1988.

Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

Charette, R.N., *IV&V and Risk Management,* Presentation Made to the ANSI/IEEE Standard 1012 Working Group at the National Institute of Standards and Technology, The American Nuclear Society, 555 North Kensington Avenue, La Grange Park, Illinois 60525, November 12, 1992.

Chisholm, G.H., B.T. Smith, and A.S. Wojcik, *Formal System Specifications - A Case Study of Three Diverse Representations*, ANL-90/43, The Mathematics and Computer Science Division and The Reactor Analysis Division, Argonne National Laboratory, Argonne, Illinois, 60439, December 1990.

Culbert, C., G. Riley, and R.T. Savely, *Approaches to the Verification of Rule-Based Expert Systems*, SOAR '87 First Annual Workshop on Space Operations Automation and Robotics, SCAMC, Inc., August 1987.

Davis, A.M., *Software Requirements: Analysis and Specification*, Prentice-Hall, Inc., New York, New York, 1990.

Department of Defense, *Military Standard 2167A, Defense System Software Development*, Department of Defense, Washington, D.C. 20362, February 29, 1988.

Deutsch, M., *Software Verification and Validation: Realistic Project Approaches*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Dunn, R., *Software Defect Removal*, McGraw-Hill, New York, New York, 1984.

Fagan, M.E., *Advances in Software Inspection*, IEEE Transactions on Software Engineering, SE-12(7), pp. 744-751, July 1986.

Galambos, J.A., R.P. Abelson, and J.R. Black (Eds), *Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

Hartway, B., J. Young, and D. Thomas, *Simulation Characterization*, Proceedings of Third International Conference on Software for Strategic Systems, 27-28 February 1990, Huntsville, Alabama, pp. 64-85.

Hatley, D. and I. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, New York, New York, 1987.

Howden, W.E., *Functional Program Testing*, IEEE Transactions on Software Engineering. SE-6(2), pp. 162-169, March 1980.

Huffman (Hayes), J.E., *Partially Automated In-Line Documentation (PAID): Design and Implementation of a Software Maintenance Tool*, Proceedings of the 1988 IEEE Conference on Software Maintenance, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York, October 1988.

Humphrey, W.S., *Managing the Software Process*, Addison Wesley Publishing Company, Reading, Massachusetts, 1990.

IEC 880, *Software for Computers in the Safety Systems of Nuclear Power Stations*, Bureau Central de la Commission Electrotechnique Internationale, 3 rue de Varemoe, Geneve Suisse, 1986.

Janusz, P.E., *Readiness Growth Model: A Quantitative Analysis of Software Risk*, Presented at the U.S. Army Armament Research, Development & Engineering Center, Picatinny Arsenal, New Jersey, August 1992.

Jensen, R., and C. Tonies, *Software Engineering*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic Testing Approach. Expert Systems with Applications: An International Journal*, Special Issue: Verification and Validation of Knowledge-Based Systems, Vol. 1, No. 3, pp. 249-269, 1990.

Miller, L.A., *A Realistic Industrial-Strength Life-Cycle Model for Knowledge-Based System Development and Testing*, Knowledge Based Systems Verification and Validation Workshop Proceedings, AAAI-90, The MIT Press, Cambridge, Massachusetts 02142, July 1990.

Miller, L.A., J.E. Hayes, and S.M. Mirsky, *Guidelines for Verification and Validation of Expert Systems. Volume 4 of this report.* Science Applications International Corporation for US Nuclear Regulatory Commission and Electric Power Research Institute, September 1993.

Mills, H., V. Basili, J. Gannon, and R. Hamlet, *Principles of Computer Programming: A Mathematical Approach*, William. C. Brown, New York, New York, 1987.

Mirsky, S.M., J.E. Hayes, and L.A. Miller, *Guidelines for Verification and Validation of Expert Systems. Volume 6 of this report.* Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute, September 1993. Available upon request from Science Applications International Corporation, P.O. Box 1303, McLean, VA 22102.

Myers, G.J., *The Art of Software Testing*, Wiley, New York, New York 1979.

NBS 500-93, *Software Validation, Verification, and Testing Technique and Tool Reference Guide*, The National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, September 1982.

Ng, P., and R. Yeh (Eds.), *Modern Software Engineering: Foundations and Current Perspectives*, Van Nostrand Reinhold, New York, New York, 1990.

NSAC-39, *Verification and Validation for Safety Parameter Display Systems*, Nuclear Safety Analysis Center, Atlanta, Georgia, December 1981.

NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

NUREG/CR-4640, *PNL-5784, Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, August 1987.

NUREG/CR-6018, EPRI TR-102106, Miller, L.A., E. Groundwater, and S.M. Mirsky, *Survey and Assessment of Conventional Software Verification and Validation Methods*, April 1993.

Omar, A., and F. Mohammed, *A Survey of Software Functional Testing Methods*, ACM SIGSOFT Software Engineering Notes, Vol. 16, No. 2, pp. 75-82, 1991.

Parnas, D.L., D.G. Smith, and T. Pearce, *Making Formal Software Documentation More Practical: A Progress Report*, Technical Report #88-236, ISSN 0836-0227, November 1988, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N7.

Preece, A.D., and R. Shinghal, *Practical Approach to Knowledge Base Verification*, Ed. M. Trivedi Proceedings of Applications of Artificial Intelligence IX, pp. 608-619, Concordia University, Montreal, Canada H3G 1M8, April 1991.

Pritsker, A.A.B., *Introduction to Simulation and SLAMII*, John Wiley and Sons, New York, New York, 1986.

Rasmussen, J., and K.J. Vicente, *Cognitive Control of Human Activities and Errors: Implications for Ecological Interface Design*, Presented at The Fourth International Conference on Event Perception and Action, Trieste, Italy, August 24-28, 1987, Riso National Laboratory, Roskilde, Denmark.

Redmill, F.J. (Ed.), *Dependability of Critical Computer Systems. Volume 2: Guidelines produced by the European Workshop on Industrial Computer Systems Technical Committee 7*, Elsevier Applied Science, New York, New York, 1989.

Regulation Guide 1,52 (Task IC 127-5) *Criteria for Programmable Digital Computer System Software in Safety-Related System of Nuclear Power Plants*, U.S. Nuclear Regulatory Commission, November 1985.

Rushby, J., *Quality Measures and Assurance for AI Software*, NASA Contractor Report 4187, SRI International, Menlo Park, California 94025, October 1988.

Rushby, J., F. von Henke, and S. Owre, *An Introduction to Formal Specification and Verification Using EHDM*, SRI International, SRI-CSL-91-02, CSL Technical Report, SRI International, Menlo Park, California 94025, February 1991.

Schulmeyer, G., *Zero Defect Software*, McGraw-Hill, Inc., New York, New York, 1990.

Schulmeyer, G.G. and J.I. McManus, *Handbook of Software Quality Assurance*, Van Nostrand Reinhold, New York, New York, 1992.

Smith, S.L., and J.N. Mosier, *Guidelines for Designing User Interface Software*, ESD-TR-86-278, Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts, United States Air Force, 1986.

Stachowitz, R.A., and J.B. Combs, *Validation of Expert Systems*, Proceedings of the Hawaii International Conference on Systems Sciences, Kona, Hawaii, Lockheed Corporation, Palo Alto, California 94304, January 1987.

United Kingdom Ministry of Defense Draft Interim Defence Standard 00-55, *Requirements for the Procurement of Safety Critical Software in Defense Equipment*, National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, May 9, 1990.

Von Mayrhauser, A., *Software Engineering: Methods and Management*, Academic Press, Inc., Boston, Massachusetts, 1990.

Wallace, R., J. Stockenberg, and R. Charette., *A Unified Methodology for Developing Systems*, McGraw-Hill, New York, New York, 1987.

Wallace, D.R., and R.U. Fujii,. *Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards*, NIST Special Publication 500-165, National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, September 1989.

Ward, P., *The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing*, IEEE Transactions on Software Engineering, 12(2), pp. 128-210, 1986.

Williges, R.C., B.H. Williges, and J. Elkerton, *Software Interface Design*, In G. Salvendy (Ed.) Handbook of Human Factors, John Wiley & Sons, New York, New York, pp. 1416-1449, 1987.

Winblad, A.L., S.D. Edwards, and D.R. King, *Object Oriented Software, Reading*, Addison-Wesley Publishing Company, Order Department, Jacob Way, Reading, Massachusetts 01867, 1990.

APPENDIX A

Example Guideline Package:

Guideline Package A for Requirements Phase, V&V Class 3, and All Components

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

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Formal Requirements Review	S	Completeness, No Unfounded Functions, 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	Small Team	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

REFERENCES

R32 NBS 500-93, *Software Validation, Verification, and Testing Technique and Tool Reference Guide*, The National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, September 1982.

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 in 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

APPENDIX B

Example Guideline Procedure:

P0.0 Sample Template

P6.0 Random Testing Procedure

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

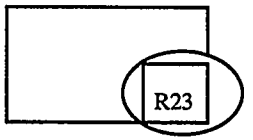
Terminal, lowest level activity, is not discussed on subsequent pages

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

P6.0 RANDOM TESTING

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform random testing (uniform whole program testing) on the system. A requirements specification has been received and has been the subject of requirements analysis. Source code has been written and has been the subject of static analysis. Possibly, unit testing has been conducted.
NOTE: Random testing should never be used as a sole testing technique.

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Source code has been delivered (or executable)
- Static analysis has been performed
- Resources are available to perform the activity
- Schedule dictates that activity commence

PLANNING

- Determine the high level tasks for this activity (see Execution below)
- Establish a schedule for the activity
- Assign human resources to the high level tasks
- Ensure that human resources agree that schedule is reasonable
- Ensure that other necessary resources are available (computer, printer, etc). Ensure that the integrated hardware/software test platform is functional and reliable enough to support testing.
- Prepare informal functional testing plan showing tasks, schedules, resources, etc.
- Select tools to be used if any select test cases
- Determine whether or not people independent of the development process will perform the testing
- Determine the applicable specifications to test against (what level will be evaluated)

PROGRAM MANAGER APPROVAL

- The informal functional testing plan is presented to the Program Manager
- Approval should be given by Program Manager before work commences

SETUP

- Ensure all human resources have functional testing plan, hard copy of the requirements specification source code/executable, etc.

EXECUTION

- 1) Depending on the level of the testing, use the requirements specification or the source code to identify decision points (paths/branches).
- 2) Determine the probability of branching in a given direction (p) for each decision point (utilize historical data, operational data, representative scenarios, live data, etc. to assist in this process.)
- 3) Utilize an automated tool (random test data generator) to generate test data. Ensure that expected results (desired outputs) can be predicted. Tools which permit extraction and checking of files after testing is completed are commercially available.
- 4) Randomly generate test data so that the probability of branching for the test cases is 1-P (complement of the probability found in Step 2 above) at each decision point.
- 5) Have a third party (preferably independent) review the test cases/test data generated and the expected results and update/modify them as appropriate.
- 6) Perform each test case and record the results in the test log, noting each defect discovered in a Software Trouble Report (STR). STRs should be prioritized according to a CM standard (such as DOD-STD-2167A) with priority 1 being the most severe.

[If an unacceptable number of STRs are found at any point during the testing, INTERRUPT]

ANALYSIS

- The number of functional test cases will be determined 

- The total number of Software Trouble Reports Defects 

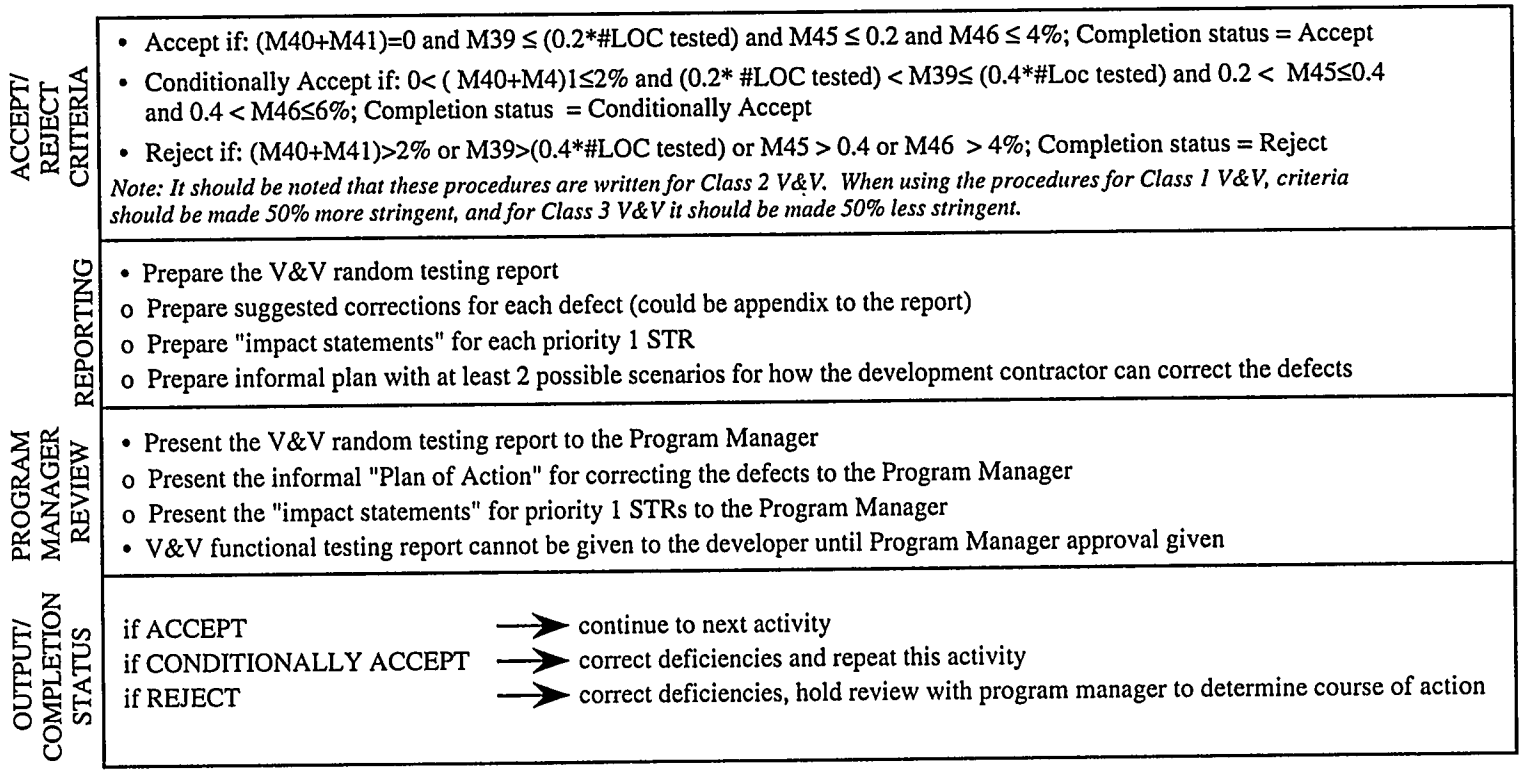
- The number of Software Trouble Reports Defects (by priority (1-5)) will be determined 

- The number of defects per line of code (LOC) will be determined $\left(\frac{\text{M39}}{\text{Total \# LOC tested}}\right)$ 

- The % of test cases failed is determined 

- o A suggested correction for each defect will be prepared $\left(\frac{\text{\# cases failed}}{\text{M37}} * 100\right)$
- o Each priority 1 STR will be analyzed for possible impact to the program. "Impact statements" will be prepared for each
- o Several contingency plans will be prepared describing how the developer can bring the STRs to closure

** An oracle will be required to determine correctness of random testing results when it is not self evident, before undertaking this activity.*

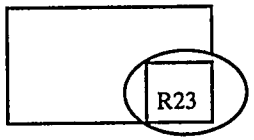


Terminal, lowest level activity, is not discussed on subsequent pages

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

METRICS & REFERENCES

M39 - Total number of Software Trouble Reports Defects
M40 - Number of priority 1 Software Trouble Reports Defects
M41 - Number of priority 2 Software Trouble Reports Defects
M42 - Number of priority 3 Software Trouble Reports Defects
M43 - Number of priority 4 Software Trouble Reports Defects
M44 - Number of priority 5 Software Trouble Reports Defects
M45 - Number defects/LOC
M46 - Percentage test cases failed defects $\left(\frac{\#\text{test cases failed}}{M37} * 100\right)$
M99 - Number of random test cases

R16 - Deutsch, M. Software Verification and Validation: Realistic Project Approaches. Englewood Cliffs, NJ: Prentice-Hall, 1982.

R19 - Hetzel, Bill. The Complete Guide to Software Testing , QED Information Sciences, Inc., Wellesley, Mass., 1988.

NRC FORM 335 (2-89) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

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
SAID-95/1028
Vol. 5

2. TITLE AND SUBTITLE
Guidelines for the Verification and Validation of Expert System Software and Conventional Software

Rationale and Description of V&V Guideline Packages and Procedures

3. DATE REPORT PUBLISHED
MONTH: March
YEAR: 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 is the fifth volume of a series describing the results of the Expert System Verification and Validation (V&V) project jointly funded by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute, to formulate guidelines for the V&V of expert and other systems. This report provides the rationale for and description of those guidelines. The actual guidelines are presented in Volume 7, "User's Manual." Three factors determine what V&V is needed: (1) the stage of the development lifecycle; (2) whether the overall system or a specialized component needs to be tested; and (3) the stringency of V&V that is needed. A V&V guideline package is provided for each of the combinations of these three variables. The package specifies the V&V methods recommended and the order in which they should be administered, the assurances each method provides, the qualifications needed by the V&V team, the performance measures that should be taken, and the decision criteria. In addition to the guideline packages, highly detailed step-by-step procedures are provided for 11 of the most important methods, to ensure that they can be implemented correctly. The guidelines can apply to conventional as well as to AI systems.

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

NRC FORM 335 (2-89)