NUREG/CR-6316
SAIC-95/1028
Vol. 7

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

User's Manual

Manuscript Completed: February 1995
Date Published: March 1995

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

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, 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.

ABSTRACT

This report provides a step-by- step guide, or user manual, for personnel responsible for the planning and execution of the verification and validation (V&V), and developmental testing, of expert systems, conventional software systems, and various other types of artificial intelligence systems. While the guide was developed primarily for applications in the utility industry, it applies well to all industries.

The user manual has three sections. In Section 1 the user assesses the stringency of V&V needed for the system under consideration, identifies the development stage the system is in, and identifies the component(s) of the system to be tested next. These three pieces of information determine which Guideline Package of V&V methods is most appropriate for those conditions.

The V&V Guideline Packages are provided in Section 2. Each package consists of an ordered set of V&V techniques to be applied to the system, guides on choosing the review/evaluation team, measurement criteria, and references to a book or report which describes the application of the method.

Section 3 presents details of 11 of the most important (or least well-explained in the literature) methods to assist the user in applying these techniques accurately.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PROJECT BACKGROUND AND BROAD OVERVIEW

This project was jointly sponsored by the U.S. Nuclear Regulatory Commission and the Electric Power Research Institute, and arose out of their joint concern for the increasing use of expert systems in nuclear utility applications. While the means for developing these systems are mature, there are not well-established and accepted procedures for their verification and validation (V&V).

The project began in late 1990 and provided a systematic investigation of a number of issues over the course of ten different activities. These activities included surveys of both conventional and expert system V&V.

The end-goal of the project, represented here, was to develop a set of guidelines for accomplishing the V&V of expert systems. As it turns out, these suggestions can be applied to all types of software, including conventional, object-oriented, neural networks, and other kinds of artificial intelligence systems other than expert systems.

The guidelines can be used for developmental testing of system software, as well as for V&V.
Nothing in the guidelines is dependent on the specific application, so any software system for any purpose can use these suggestions.

The suggestions for V&V are presented in the form of a set of ordered methods called packages. There are sixteen packages covering sixteen development situations based on the stage of development of the software, the stringency of V&V needed, and the type of system component being examined. Eleven of the more important methods are described in detailed step-by-step procedures.

The major theoretical underpinning of the guidelines derive from an approach to V&V called Fault-Specific Verification (FSV). From the FSV perspective, the choice of methods should be based on the faults that can occur in the particular development artifact being tested, whether it is the requirements document, a graphic depiction of the design architecture, or the actual program code. Once the faults are identified, a set of methods should be chosen based on the methods' effectiveness in identifying the specific faults and the degree of rigorous V&V believed to be necessary. An optimal set of methods will address all the possible faults with an effectiveness which is constrained to be the lowest total cost consistent with the judged level of need for rigor in the software testing.

KEY USER INFORMATION

Purpose of this manual:

To provide suggestions for different methods to use in the Verification & Validation (V&V) of software.

Who should use it:

Persons responsible for planning and executing these activities. (Step-by-step procedures are provided for 11 of the recommended methods because these are believed to be the most important or least well-documented.)

What types of software systems this applies to:

Conventional systems and expert systems (including knowledge-based, rule-based, and object-oriented). Includes modifications to existing systems.

What information is provided:

For each of the 16 different development situations, a set of suggested methods is given, with the recommended order of use and the following information:

- the assurances it provides
- the extent to which it should be applied
- the decision thresholds
- the skills needed to apply it
- the performance measures to be taken
- the decisions that may be appropriate

What steps I must take to find the methods appropriate to my situation:

Step 1: Decide what stage of development your system is in (choose from a table).

Step 2: Decide what aspect(s) of the system you need to examine (choose from a table).

Step 3: Decide how rigorously your system needs to be examined (answer the questionnaire).

Step 4: Find the appropriate set of suggested methods (choose from a table).

ORGANIZATION OF THIS MANUAL

Section 1	Contains the four steps needed to identify the appropriate set of guideline packages.
Section 2	Contains the guideline packages for the different development situations.
Section 3	Contains the detailed step-by-step instructions to apply 11 different procedures.

SECTION 1: OVERVIEW

OVERVIEW OF STEPS TO DETERMINE APPROPRIATE GUIDELINE PACKAGES

Contents:

This section contains the four steps needed to identify the set of methods you should follow. The beginning of each step will be indicated by a heading at the top of the page. A step may have more than one page.

What to do:

- Record the decision you reach about your software in each of the first three steps.
- Use this information in Step 4 to determine the suggested package of methods for your situation.
- Turn to Section 2 to find the suggested package.

STEP 1
DETERMINING WHAT STAGE OF DEVELOPMENT YOUR SYSTEM IS IN

Purpose: To determine your system's stage of development.

What to do:

- Examine the table below.
- Find the development stage that most closely describes the status of your software.
- Record this information.

Development Stage	Description of this stage	What to examine
Requirements	The beginning stage. Exactly what the system is supposed to do is being formulated. A careful write-up of these desired features will constitute the Requirements Document.	Requirements Document
Design	A Requirements Document exists. Planning and description of the logical functions and physical elements that will satisfy the requirements is under way or just completed. This constitutes the Design Document. Implementation has not yet begun in earnest.	Design Document
Implementation	A Design Document exists in some form. The design is being (or has been) translated into programming language code, expert system rules, object-oriented classes and methods, or some other executable form. The result of this phase is a working version of the system on some hardware platform(s.) All the software files that are needed for such a working system comprise the System Source Code. The system is not yet installed and fully operational.	System Source Code

STEP 2
DETERMINE WHAT ASPECTS OF THE SYSTEM YOU NEED TO TEST

Purpose: To determine what aspects of the system you need to test.

What to do:

- Examine the table below. Three types of system components are described. (The last type includes the system as a whole.)
- Find the type that most closely describes the type of system component you are now ready to test.
- Record the type of system component.

Type of System Component	Description of Type of System Component To Be Tested	Type of V&V Suggested
Knowledge Base	• The rules, frames, or knowledge elements of expert and knowledge-based systems. • The classes and class hierarchies of object-oriented systems. • The information stored in tables that drive the processing in certain kinds of conventional systems (e.g., table-driven parers).	Static analysis and formal verification methods
Highly Reusable Components	• The inference engine of expert and knowledge-based systems. • Most frequently used tools and utilities. • Commercial software packages.	Certification methods
Modification	A working system based on the system source code exists, has been fully installed, and is operational. Now it is desirable to modify this existing system in some way. Theses changes are in some state of development, corresponding to one of the above three stage.	Modification requirements, modification design, or modified system source code.
System and Other	• The overall system. • Data and programming or system interfaces. • Conventional custom-written software of all kinds	Traditional systems V&V methods

STEP 3
OVERVIEW OF STEPS TO DETERMINE APPROPRIATE LEVEL OF V&V

Purpose:

To determine how rigorously your system needs to be examined.

What to do:

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

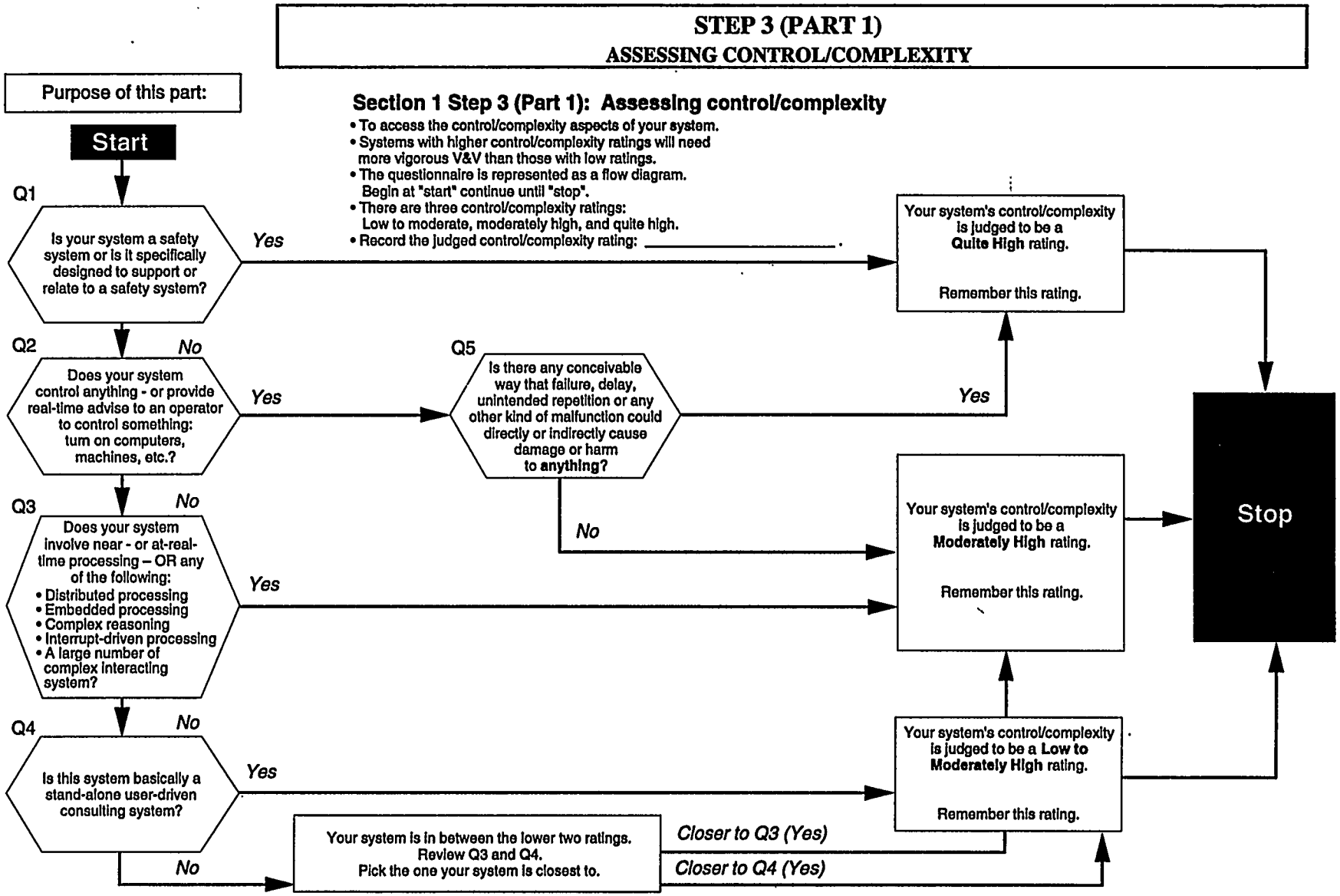


Figure 7.3-1 Three-part questionnaire to determine recommended V&V Class.

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

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

STEP 4:
DETERMINING THE APPROPRIATE "SUGGESTED GUIDELINE PACKAGE"

Purpose:

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

What to do:

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

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

SECTION 2:
OVERVIEW TO SUGGESTED GUIDELINE PACKAGES

Contents

This section contains the 16 V&V guideline packages, A-P. These packages appear after this overview. Refer to Guideline Package A during the following explanation. Note that identifiers at the top of each package show the name of the package, its life-cycle phase, V&V class, and component type. The package is in the form of two tables and a reference list. The first table lists all the V&V methods that are recommended for this situation. The second table (located after the reference list) tells how you should apply these methods.

What to do:

Look at the first table. Each column is labeled and contains notes about the contents of that column.

- The first column. labeled "#", gives an ID number for the V&V methods described in that row. This ID number is referred to in the guideline plan table.
- The second 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 sixth column, "Intensity", 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.
- 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 "Reject", the development team must make a decision 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 Volume 8). If the entry begins with a "P", then it is a Guideline Procedure to be found in Section 3.

In the case of Guideline Package A, only two V&V methods are recommended: Requirements Analysis and Formal Requirements Review. (Sometimes there are as many as 17.) Look at the second table, the Guideline Plan. According to the plan, the V&V methods should be performed in the order they are given. For Requirements Analysis, which should 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%.

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 S2-1. 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.

The thresholds are based at the present time only on engineering judgment. They are derived from the kinds of considerations given below. These 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 Deficient and Major Problems metrics:

- One or two 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 less stringent.
- 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.
- Since V&V is in addition to all normal developmental testing, more than 4 major errors 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.

Table S2-1- Standardized decision thresholds (based on engineering judgment) for Guideline Packages

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

[1] For example, the conditional acceptance 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 justify that decision; when logic is "or" any of the metrics being true justifies the decision.

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 with failure, and a considerably stricter and more 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 flaws will be hunted down and eliminated.

Please note: This report only provides reasonable guidance, and the users should modify the recommendations based on their own particular situations.

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 requirement measures, it is assumed that a requirements 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), separate the true requirements from the "goals" by examining sentences for the presence of "shall" and "will": sentences containing "shall" are true requirements; sentences containing "will" are only objectives to strive for and are not requirements. They should be ignored for the purposes of the rating of requirements as critical or non-critical.

For all the true requirements, the procedure to use is to examine each requirement carefully, and ask 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 on the ordered list around this change-over point until there appears a partition point above which the requirements all seem to share a necessity, and below which they seem less strong. This is then a good decision point for defining "critical" and "non-critical": those requirements above the line are critical, those below are 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 (however nice they might be as functions). Each separate occurrence of unintended function should be noted.

Concerning the quality measures of "show-stoppers," "majors," and "minors," a different procedure needs to 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 -- how much rework is necessary to correct the problem? A "show-stopper" is something which implies something seriously 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 flawed architectural approach to achieve functional diversity in a safety-critical control system.

Show-stoppers can only occur as a broad problem 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 made any number of additional illogical decisions." 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 requiring only a little 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.

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.[1] Major factors 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 performs 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.

---

[1] The sixteenth package, **P**, is for the maintenance situation and provides guidance for selecting one or more of the other packages **A-O**. P itself does not contain any methods, but refers to one of the other packages and provides general guidance.

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 system needs testing from a wide variety of perspectives.

Table S2-2 shows the use of the 32 chosen methods in the 15 Guideline Packages A-O. Nine of the methods are used for a third or more 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 seven 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 in the table, 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 the table (given by the last row), one can see that, except for the Requirements phase, the difference in 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 -- Packages **G, H**, and **I** -- reveals that a number of dynamic methods are included. The question could be asked: "if the knowledge base is tested alone, it really can't be executed without at least the inference engine; therefore, isn't it possible to have only static analysis methods for knowledge base testing?" The answer, and rationale, is that if one is focusing on testing the knowledge base, one can use dynamic methods by running the system cut off from the full operational interfaces with data and other systems, stubbing these or writing special device drivers as necessary. Under these conditions, the knowledge base will be interacting with the inference engine, a necessary test circumstance, but in a greatly simplified operational environment. When it is not feasible to run the knowledge base with the inference engine in such an isolated manner, then only the static analysis methods need be employed for the limited assessment of just the knowledge base in isolation.

One last aspect of the table 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, 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.

Table S2-2 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**

GUIDELINE PACKAGE A

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 $\leq$ 4 AND %NRD < 8 **CA:** #CRD $\leq$ 4 AND #NRD $\leq$ 6 AND %NRD $\leq$ 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 within a column in any order. Stop the V&V activity at "Exit" column.

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

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

GUIDELINE PACKAGE B

GUIDELINE PACKAGE: B

Lifecycle Phase: Requirements **V&V Class: 2** **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	**Major Team:** 2- IV&V, 2- Users, 2-4 SMEs, 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2- IV&V Requirement Analysts **Team B:** 2- IV&V System Engineering Specialists **Team C:** 2- Experienced in Formal Verification	**Moderately Extensive:** Thorough individual and joint analysis and investigation of the requirements, with several formal team meeting(s) to discuss issues and problems until resolved.	**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.	Team A	Moderately Extensive	A: #CRD = 0 AND #NRD ≤ 2 AND %NRD ≤ 4 CA: #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 R: #CRD > 2 OR #NRD > 4 OR %NRD > 8	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, 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	Major Team	Moderately Extensive	A: #CRD = 0 AND #NRD ≤ 2 AND %NRD < 4 CA: #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 R: #CRD > 2 OR #NRD > 4 OR %NRD > 8	R32
3	Operational Concept Analysis	S	Adequate Concept of Operations	Team B	Moderately Extensive	A: #CRD = 0 AND #NRD ≤ 2 AND %NRD < 4 CA: #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 R: #CRD > 2 OR #NRD > 4 OR %NRD > 8	R41
4	User Interface Inspection	S	Adequate HCI	Team B	Moderately Extensive	A: #CRD = 0 AND #NRD ≤ 2 AND %NRD < 4 CA: #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 R: #CRD > 2 OR #NRD > 4 OR %NRD > 8	R35

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R41 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.

Guideline Plan: *for Guideline Package B*

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

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

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

Start Here	Do Next	Do Next	Exit V&V Activity
1	2	3 4	Yes

GUIDELINE PACKAGE C

GUIDELINE PACKAGE: C

Lifecycle Phase: Requirements **V&V Class: 1** **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	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2- IV&V Requirement Analysts **Team B:** 2- IV&V System Engineering Specialists **Team C:** 2-Experienced in Formal Verification	**Extensive:** Extremely thorough individual and joint analysis and investigation of the requirements, with several formal team meeting(s) to discuss issues and problems until resolved.	**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	Team A	Extensive	**A:** #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 **CA:** #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 **R:** #CRD > 1 OR #NRD > 2 OR %NRD > 4	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

[3] SMEs : Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Formal Requirements Review	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	Major Team	Extensive	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4	R32
3	Operational Concept Analysis	S	Adequate Concept of Operations	Team B	Extensive	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4	R41
4	User Interface Inspection	S	Adequate HCI	Team B	Extensive	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4	R35

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
5	Requirement Modeling & Animation with Computer Tool (e.g., System Requirements Engineering Methodology[4])	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	Team B	Extensive	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4	R1
6	Formal Verification (e.g., PCS or EHDM)	S	Completeness, Correctness, Clarity, Consistency	Team C	Extensive	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4	R44

[4] An automated tool is very strongly advised. If this is impossible, *Prototyping* is highly recommended to insure the feasibility of the requirements.

REFERENCES

R1 Alford, M. *SREM At The Age of Eight: The Distributed Computing Design System*, Computer, 18(4), pp. 36-46, 1985.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R41 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.

R44 Rushby, J., *An Introduction to Formal Specification and Verification Using EHDM*, SRI International, CSL Technical Report SRI-CSL-91-02, February 1991.

Guideline Plan: *for Guideline Package C*

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

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

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

Start Here	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2	3 4	5	6 *(Condition: Use this method only if it fits well to the problem)*	Yes

GUIDELINE PACKAGE D

GUIDELINE PACKAGE: D

Lifecycle Phase: Design **V&V Class: 3** **Component Type: All**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Formal Design Review (FDR) applies to all assurances	S = Static D = Dynamic	All assurances are considered at the same time when reviewing the Design Document with FDR	**Small Team:** 1- IV&V Leader, 1- Subject Matter Expert, 1- System Software Engineer **Team A:** 1 or 2 IV&V Requirements Analysts **Team B:** 1 or 2 IV&V System Engineer/ Human Factors Analysts **Team C:** 1 or 2 IV&V Knowledge Engineers	**Limited:** Reasonably thorough review of design 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 Tracing	S	Completeness No Unintended Functions.	Team A	Limited	**A:** #CRD = 0 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 6 **CA:** #CRD ≤ 4 AND #NRD ≤ 6 AND %NRD ≤ 12 AND %UF ≤ 10 **R:** #CRD > 4 OR #NRD > 6 OR %NRD > 12 OR %UF > 10	R32 P1.0

[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	Operational Concept Analysis	S	Adequate Concept of Operations	Team B	Limited	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min >12	R41
3	User Interface Inspection	S	Adequate Human-Computer Interface	Team B	Limited	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min >12	R35
4	Formal Design Review	S	Completeness, No Unintended Functions, Correctness, Clarity, Explicitness, General Consistency, Interfaces Consistent, Adequate Concept of Operations, Adequate Human-Computer Interface, Adequate Database Design	Small Team	Limited	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min >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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units,* W. Gilmore, July 1985.
NUREG/CR-5908, *Advanced Human-System Interface Design Review Guideline, Vol. 1 & Vol. 2,* J.M. O'Hara, July 1994.

R41 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.

Guideline Plan: *for Guideline Package D*

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

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

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

Start Here	Do Next	Do Next	Exit V&V Activity
1	2 3	4	Yes

GUIDELINE PACKAGE E

GUIDELINE PACKAGE: E

Lifecycle Phase: Design **V&V Class: 2** **Component Type: All**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Formal Design Review (FDR) applies to all assurances in addition to whatever other methods are proposed.	S = Static D = Dynamic	All assurances are considered at the same time when reviewing the Requirements Document and the Design Documents with the formal Design Review method.	**Major Team:** 2- IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2- Requirement Analysts **Team B:** 2- Systems Engineers **Team C:** 1-2 Human Factors Specialists **Team D:** 1-2 Database Engineers **Team E:** 1-2 Software Engineers **Team F:** 1-2 Knowledge Engineers	**Extensive:** Extremely thorough analysis and investigation of the design, with periodic meeting(s) to discuss the problems. Often involves detailed modeling. **Moderate:** Thorough analysis, but not super-detailed, often as precursor to next method.	**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 Tracing	S	Completeness No Unintended Functions.	Team A	Moderate	**A:** #CRD = 0 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 4 **CA:** #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 5 **R:** #CRD > 2 OR #NRD > 4 OR %NRD > 8 OR %UF > 5	R32 P1.0

[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

[3] **SMEs :** Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Operational Concept Analysis	S	Adequate Concept of Operations	Team C	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R41
3	User Interface Inspection	S	Adequate Human-Computer Interface	Team C	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R35
4	Database Analysis	S	Adequate Database Design	Team D	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R33
5	Software Practices Review	S	Adequate Software Engineering Practices and Standards Compliance	Team E	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R10

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Knowledge Engineering Analysis	S	Adequate Knowledge Structures	Team F	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R18
7	Formal Design Review	S	Completeness, Correctness, Clarity, Explicitness, Consistency, Adequate DB Design, 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	Major Team	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2	R32

REFERENCES

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk*, Elsevier, New York, New York, 1988.

R18 Galambos, J.S., R.P. Abelson, and J.R. Black (Eds.), *Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R41 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.

Guideline Plan: *for Guideline Package E*

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

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

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

Start Here	Do Next	Do Next	Exit V&V Activity
1	2 3 4 5 6	7	Yes

GUIDELINE PACKAGE F

GUIDELINE PACKAGE: F

Lifecycle Phase: Design **V&V Class: 1** **Component Type: All**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Formal Design Review (FDR) applies to all assurances in addition to whatever other methods are proposed.	S = Static D = Dynamic	All assurances are considered at the same time when reviewing the Requirements Document and the Design Documents with the formal Design Review method.	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2- Requirement Analysts **Team B:** 2- Systems Engineers **Team C:** 1-2 Human Factors Specialists **Team D:** 1-2 Database Engineers **Team E:** 1-2 Software Engineers **Team F:** 1-2 Knowledge Engineers	**Extensive:** Extremely thorough analysis and investigation of the design, with periodic meeting(s) to discuss the problems. Often involves detailed modeling. **Moderate:** Thorough analysis, but not super-detailed, often as precursor to next method.	**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 Tracing	S	Completeness No Unintended Functions.	Team A	Extended	**A:** #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 AND %UF ≤ 1 **CA:** #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 3 **R:** #CRD > 1 OR #NRD > 2 OR %NRD > 4 OR %UF > 3	R32 P1.0

[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

[3] **SMEs :** Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Operational Concept Analysis	S	Adequate Concept of Operations	Team C	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R41
3	User Interface Inspection	S	Adequate Human-Computer Interface	Team C	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R35
4	Database Analysis	S	Adequate Database Design	Team D	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R33
5	Software Practices Review	S	Adequate Software Engineering Practices and Standards Compliance	Team E	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R10

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Knowledge Engineering Analysis	S	Adequate Knowledge Structures	Team F	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R18
7	Formal Design Review	S	Completeness, Correctness, Clarity, Explicitness, Consistency, Adequate DB Design, 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	Major Team	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R32
8	System Engineering Review	S	Completeness, Correctness, 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	Team B	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R7

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
9	Failure Mode, Effects, Causality Analysis	S	Contained Safety and Hazard Conditions, Fail-Safe/Soft-Failure, Adequate System Architecture	Team B	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R49
10	Design Modeling and Animation with Computer Tool (e.g., System Requirement Engineering Methodology)	S	Completeness, Correctness, Adequate Concept of Operations, Adequate Human-Computer Interface, Adequate System Architecture, Adequate Function Partitioning, Adequate Database Design, Contained Safety and Hazard Conditions, Adequate Knowledge Structures, Fail-Safe/Soft-Failure	Team B	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R1

REFERENCES

R1 Alford, M. *SREM At The Age of Eight: The Distributed Computing Design System*, Computer, 18(4), pp. 36-46, 1985.

R7 Blanchard, B.S., and W.J. Fabrycky, *Systems Engineering and Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk*, Elsevier, New York, New York, 1988.

R18 Galambos, J.S., R.P. Abelson, and J.R. Black (Eds.), *Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

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.

R33 Ng, P., and R. Yeh, (Eds.) *Modern Software Engineering: Foundations and Current Perspectives*, Van Nostrand Reinhold, New York, New York, 1990.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R41 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.

R49 United Kingdom Ministry of Defense Draft Interim Defense Standard 00-55, *Requirements for the Procurement of Safety Critical Software in Defense Equipment*, The National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, May 9, 1989.

Guideline Plan: *for Guideline Package F*

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

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

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

Start Here	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3 4 5 6	7	8 9	10	Yes

GUIDELINE PACKAGE G

GUIDELINE PACKAGE: G

Lifecycle Phase: Implementation **V&V Class: 3** **Component Type: Knowledge Structure**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Formal Customer Review (FCR) applies to all assurances.	S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 1- IV&V leader, 1-Subject Matter Expert, 1- System/Software Eng. **Team A:** 1 or 2 IV&V Requirements Analysts **Team B:** 1-2 IV&V Knowledge Engineers **Team C:** 1-2 Experienced Dynamic Testers	**Limited:** *STATIC* -- Reasonably thorough review of code via independent study followed by limited group discussion. *DYNAMIC* --Exercise most of the knowledge structure and all functions, but no intense or exhaustive testing.	**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 Tracing[2]	S	Completeness No Unintended Functions.	Team A	Limited	**A:** #CRD = 0 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 6 **CA:** #CRD ≤ 4 AND #NRD ≤ 6 AND %NRD ≤ 12 AND %UF ≤ 10 **R:** #CRD > 4 OR #NRD > 6 OR %NRD > 12 OR %UF > 10	R32 P1.0

[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	Automated Syntax Checking of Knowledge Structure (e.g., with VERITE')	S	Assurance that known types of syntax errors in the knowledge structure are detected	Team B	Limited	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R29 P10.0
3	Semantic/Meta-Level Knowledge Checking (e.g., with Meta-Check)	S	Assurance that errors are identified that are detectable by input and reasoning about engineering knowledge about the system	Team F	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R29 P7.0
4	Formal Customer Review[2]	S	Completeness, No Unintended Functions, Logic and Control Correct, Data Operations and Computations Correct, General and Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure, Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Engineering Practice, Adequate Knowledge Structure and Contents	Small Team	Limited	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R36

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
5	Functional Testing	S	Knowledge is adequate and correct	Team C	Limited (e.g., one test case per function)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R36 R37 P3.0
6	Random Testing[2]	D	Adequate System Performance under Random Input	Team C	Limited (e.g., 15-30 test cases)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R4 P6.0

REFERENCES

R4 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.

R29 Preece, A.D., *Towards a Methodology for Evaluating Expert Systems*, Expert Systems, Vol. 7, No. 4, pp. 215-223, November 1990.

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.

R36 Howden, W.E., *Functional Program Testing*, IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R37 NUREG/CR-4640, PNL-5784, *Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, August 1987.

Guideline Plan: *for Guideline Package G*

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

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

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

Start Here	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3	4	5	6	Yes

GUIDELINE PACKAGE H

GUIDELINE PACKAGE: H

Lifecycle Phase: Implementation **V&V Class: 2** **Component Type: Knowledge Structure**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Formal Customer Review (FCR) applies to all assurances.	S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial expert, 1-3 Customer Representatives **Team A:** 2- Requirement Analysts **Team B:** 2- Systems Engineers **Team C:** 1-2 Human Factors Specialists **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers	**Moderate:** *STATIC*- Thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Thorough intensive test-case coverage and sampling.	**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 Tracing[4]	S	Completeness, No Unintended functions	Team A	Moderate	**A:** #CRD = 0 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 4 **CA:** #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 5 **R:** #CRD > 2 OR #NRD > 4 OR %NRD > 8	R32 P1.0

[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

[3] **SMEs :** Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Automated Syntax Checking of Knowledge Structure (e.g., with VERITE"; cf. Preece, 1990)	S	Assurance that known types of syntax errors in the knowledge structure are detected	Team F	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R29 P10.0
3	Semantic/Meta-Level Knowledge Checking (e.g., with Meta-Check)	S	Assurance that errors that are detectable by input and reasoning about engineering knowledge of the system are identified	Team F	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R29 P7.0
4	Desk Checking (of Knowledge & Structure)	S	RE: Knowledge Structure Completeness, Syntax & structure OK, Logic & control correct, Data operations and computations correct, Consistency, Adequate Concept of Operations, Adequate Human-Computer Interface (HCI), Adequate Documentation, Adequate Safety/Hazard, Adequate SW practices, Adequate Standards compliance	Teams E and F	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R17 R20 P2.0
5	Knowledge Engineering Analysis	S	Adequate Knowledge Structures	Team F	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R18

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Formal Customer Review[4]	S	Completeness, No Unfounded Functions, Logic and Control Correct, Data Operations and Computations Correct, General & Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Eng. Practice, Adequate Knowledge Structure and Contents	Major Team	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R36 P11.0
7	Functional Testing	D	Knowledge is adequate and correct	Team C	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R36 P3.0
8	Boundary Testing	D	Assurance that errors do not exist at boundaries and inflection points of input space	Team F and/or Team E	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R9 P4.0

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
9	Random Testing[4]	D	Adequate System Performance under Random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R4 P6.0
10	Regression Testing	D	Assurance that fixes to the problems have not introduced new problems	Team E	Moderate: Run after every major set of fixes	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R32 P9.0

[4] While this technique is to be accomplished on the system as a whole, it is indicated here as a necessary step for assuring the quality of the knowledge structure.

REFERENCES

R4 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.

R9 Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, New York, New York, 1990.

R17 Fagan, M.E., *Advances in Software Inspection,* IEEE Transactions on Software Engineering, SE-12(7), pp. 744-751, July 1986.

R18 Galambos, J.A., R.P. Abelson, and J.R. Black (Eds.), *Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

R20 Miller, L.A., J.E. Hayes, and Steven M. Mirsky, *Guidelines for Verification of Expert Systems. Volume 4 of this report.* Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute. September 1993.

R29 Preece, A.D., *Towards a Methodology for Evaluating Expert Systems*, Expert Systems, Vol. 7, No. 4, pp. 215-223, November 1990.

R31 Myers, G.J., *The Art of Software Testing*, Wiley, New York, New York, 1979.

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.

R36 Howden, W.E., *Functional Program Testing*, IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

Guideline Plan: *for Guideline Package H*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 10 after all major fixes. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3	4 5	6	7	8	9	Yes

GUIDELINE PACKAGE I

GUIDELINE PACKAGE: I

Lifecycle Phase: Implementation **V&V Class: 1** **Component Type: Knowledge Structure**

#	Method	Type	Assurances	Agents[1]	Intensity	Criteria/Action[2]	Ref[1]
	Formal Customer Review (FCR) applies to all assurances.	S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 Systems Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing[4]	S	Completeness, No Unintended functions	Team A	Extended	A: #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 AND %UF ≤ 1 CA: #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 3 R: #CRD > 1 OR #NRD > 2 OR %NRD > 4 OR %UF > 3	R32 P1.0

[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

[3] **SMEs:** Subject Matter Experts
[4] While this technique is to be accomplished on the system as a whole, it is indicated here as a necessary step for assuring the quality of the knowledge structure.

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Automated Syntax Checking of Knowledge Structure (e.g., with VERITE')	S	Assurance that syntax errors in the knowledge structure are detected	Team F	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R29 P10.0
3	Semantic/Meta-Level Knowledge Checking (e.g., with Meta-Check)	S	Assurance that errors that are detectable by input and reasoning about engineering knowledge of the system are identified	Team F	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R20 R29 P7.0
4	Desk Checking (of Knowledge & Structure)	S	RE: Knowledge Structure Completeness, Syntax & structure OK, Logic & control correct, Data operations and computations correct, Consistency, Adequate Concept of Operations, Adequate Human-Computer Interface (HCI), Adequate Documentation, Adequate Safety/Hazard, Adequate SW practices, Adequate Standards compliance	Team E and Team F	Extensive	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R17 P2.0
5	Knowledge Engineering Analysis	S	Adequacy of the structure and the general contents of the knowledge component	Team F	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R18

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	FMECA (Failure Mode, Effects, Causality Analysis)	S	Knowledge & Containment of hazards and safety problems "fail-safe" and "soft failure"	Team B and Team E, with SMEs	Extended (Envision and test knowledge structure for all possible failures and causes thereof)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R49
7	Formal Customer Review[4]	S	Completeness, No Unfounded Functions, Logic and Control Correct, Data Operations and Computations Correct, General & Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure, Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Eng. Practice, Adequate Knowledge Structure and Contents	Major Team	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R36 P11.0
8	Functional Testing	D	Knowledge is adequate and correct	Team C	Extended (e.g., multiple tests for each function)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R36 P3.0

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
9	Boundary Testing	D	Assurance that errors do not exist at boundaries and inflection points of input space	Team F and/or Team E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R31 P4.0
10	Random Testing[4]	D	Adequate System Performance under Random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R4 P6.0
11	Structural Testing	D	Assurance each major knowledge element has been exercised	Team F and/or Team E	Moderate (generate test cases only for elements of knowledge structure not exercised by test cases of other methods)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R5
12	Regression Testing	D	Assurance that fixes to problems have not introduced new problems	Team E	Moderate to Extended: Run after every major set of fixes.	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R32 P9.0

REFERENCES

R4 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.

R5 Dunn, R., *Software Defect Removal,* McGraw-Hill, New York, New York, 1984.

R9 Beizer, B., *Software Testing Techniques,* Van Nostrand Reinhold, New York, New York, 1990.

R17 Fagan, M.E., *Advances in Software Inspection,* IEEE Transactions on Software Engineering, SE-12(7), pp. 744-751, July 1986.

R18 Galambos, J.S., R.P. Abelson, and J.R. Black (Eds.), *Knowledge Structures,* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

R20 Miller, L.A., J.E. Hayes, and Steven M. Mirsky, *Guidelines for Verification of Expert Systems. Volume 4 of this report.* Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute. September 1993.

R29 Preece, A.D., *Towards a Methodology for Evaluating Expert Systems,* Expert Systems, Vol. 7, No. 4, pp. 215-223, November 1990.

R31 Myers, G.J., *The Art of Software Testing,* Wiley, New York, New York, 1979.

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.

R36 Howden, W.E., *Functional Program Testing,* IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R49 United Kingdom Ministry of Defense Draft Interim Defence Standard 00-55, *Requirements for the Procurement of Safety Critical Software in Defense Equipment,* The National Institute of Standards and Technology Computer Systems Laboratory, Gaithersburg, Maryland 20899, May 9, 1989.

Guideline Plan: *for Guideline Package I*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 12 after all major fixes. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3	4 5	6	7	8	9	10	11	Yes

GUIDELINE PACKAGE J

GUIDELINE PACKAGE: J

Lifecycle Phase: Implementation **V&V Class: 3** **Component Type: Highly Reusable**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Key assumption: Highly Reusable Components will seldom be built; rather, they will be bought as part of shells, etc., and source code usually won't be available. Accordingly, can't use very many static V&V methods. Must rely on typical dynamic certification means.	S = Static Testing D = Dynamic Testing SD = Static Analysis after Dynamic Execution	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2- IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 System Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers **Team G:** 2 Experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No Unintended functions	Team A	Extended	A: #CRD = 0 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 6 CA: #CRD ≤ 4 AND #NRD ≤ 6 AND %NRD ≤ 12 AND %UF ≤ 10 R: #CRD > 4 OR #NRD > 6 OR %NRD > 12 %UF > 10	R32 P1.0

[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

[3] SMEs : Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Software Practices Review	S	Adequate Software Engineering Practices and Standards Compliance	Team E	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R29 P10.0
3	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R36 P3.0
4	Beta Testing	D	Assures that component performs adequately under the varied conditions of the test sites	Teams E, C	Extended (depends on the number of -- presumably internal -- beta test sites)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R46
5	Random Testing	D	Adequate System Performance under Random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R4 P6.0

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Robustness Testing	D	Assurance that the system can perform adequately or fail - safely for intended input and operating conditions	Team E, SMEs	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 4 CA: #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 R: #SS > 1 OR #Maj > 4 OR #Min > 12	R28 P8.0

REFERENCES

R4 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.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

R28 Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic 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.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units,* W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing,* IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R41 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.

R46 Schulmeyer, G.G. and J.I. McManus, *Handbook of Software Quality Assurance,* Van Nostrand Reinhold, New York, New York, 1992.

Guideline Plan: *for Guideline Package J*

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

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

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

Start Here	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2	3	4 5	6	Yes

GUIDELINE PACKAGE K

GUIDELINE PACKAGE: K

Lifecycle Phase: Implementation **V&V Class: 2** **Component Type: Highly Reusable**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Key assumption: Highly Reusable Components will seldom be built; rather, they will be bought as part of shells, etc., and source code usually won't be available. Accordingly, can't use very many static V&V methods. Must rely on typical dynamic certification means.	S = Static Testing D = Dynamic Testing SD = Static Analysis after Dynamic Execution	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2- Requirement Analysts **Team B:** 2- System Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2- Database Engineers **Team E:** 2- Software Engineers **Team F:** 2- Knowledge Engineers **Team G:** 2- Experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No Unintended functions	Team A	Extended	**A:** #CRD = 0 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 4 **CA:** #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 5 **R:** #CRD > 2 OR #NRD > 4 OR %NRD > 8 OR %UF > 5	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

[3] SMEs : Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	User Interface Inspection	S	Adequate HCI	Teams C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R35
3	Operational Concept Analysis	S	Adequate CONOPS	Teams B, C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 #Maj > 2 OR #Min > 8	R41
4	Bench Marking	D	Assures that component performs as advertised and shows no difficulty with devilish problems in the test suite.	Team E	Extended (depends on size of bench marking test suite)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R46
5	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R36 P3.0

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Random Testing	D	Adequate System Performance under Random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R4 P6.0
7	Robustness Testing	D	Assurance that system can perform adequately or fail-safely for unintended input and operating conditions	Team E, SMEs	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R28 P8.0
8	Software Practices Review	S	Adequate Software Engineering Practices and Standards Compliance	Team E	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R10
9	Beta Testing	D	Assures that component performs adequately under the varied conditions of the test sites	Teams E, C	Extended (depends on the number of -- presumably internal -- beta test sites)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R46

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
10	Regression Testing	D	Assurance that the new versions of these components work as the previous ones	Team E	Moderate to Extended: (it depends on the effort an dsevoted to maintaing a current regression testing suite)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R32 P9.0

REFERENCES

R4 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.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

R28 Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic 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.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units,* W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing,* IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R41 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.

R46 Schulmeyer, G.G. and J.I. McManus, *Handbook of Software Quality Assurance,* Van Nostrand Reinhold, New York, New York, 1992.

Guideline Plan: *for Guideline Package K*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 10 after all major fixes, releases, and updates. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3	4 Note: The next method may be begun before this method completes	5	6	7 8	9	Yes

GUIDELINE PACKAGE L

GUIDELINE PACKAGE: L

Lifecycle Phase: Implementation **V&V Class: 1** **Component Type: Highly Reusable**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
	Key assumption: Highly Reusable Components will seldom be built; rather, they will be bought as part of shells, etc., and source code usually won't be available. Accordingly, can't use very many static V&V methods. Must rely on typical dynamic certification means.	S = Static Teating D = Dynamic Testing SD = Static Analysis after Dynamic Execution	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 Systems Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers **Team G:** 2 experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No unintended functions	Team A	Extended	**A:** #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 AND %UF ≤ 1 **CA:** #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 3 **R:** #CRD > 1 OR #NRD > 2 OR %NRD > 4 OR %UF > 3	R32 P1.0

[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

[3] SMEs : Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	User Interface Inspection	S	Adequate HCI	Teams C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R35
3	Operational Concept Analysis	S	Adequate CONOPS	Teams B, C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R41
4	Data Interface Inspection	S	Consistent, complete, compliant, and correct data interfaces	Team B, D, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R32
5	Software Practices Review	S	Adequate Software Engineering Practices and Standard Compliance	Team E	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R10

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Bench Marking	D	Assures that component performs as advertised and shows no difficulty with devilish problems in the test suite.	Team E	Extended (depends on size of bench marking test suite)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R46
7	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R36 P3.0
8	Random Testing	D	Adequate system performance under random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R4
9	Robustness Testing	D	Assurance that system can perform adequately or fail safely for unintended input and operating conditions	Team E, SMEs	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R28

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
10	Beta Testing	D	Assures that component performs adequately under the varied conditions of the test sites	Teams E, C	Extended (depends on the number of -- presumably internal -- beta test sites)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R46
11	Regression Testing	D	Assurance that new versions of these components work as the previous ones	Team E	Moderate to Extended: (it depends on the effort devoted to maintaining a current regression testing suite)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R32

REFERENCES

R4 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.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

R28 Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic 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.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units,* W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing,* IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R41 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.

R46 Schulmeyer, G.G. and J.I. McManus, *Handbook of Software Quality Assurance,* Van Nostrand Reinhold, New York, New York, 1992.

Guideline Plan: *for Guideline Package L*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 11 after all major fixes, releases, and updates to components. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3 4	5	6 Note: This method may not be completed for a long period. OK only for this method to begin next method before this completes.	7	8	9	10	Yes

GUIDELINE PACKAGE M

GUIDELINE PACKAGE: M

Lifecycle Phase: Implementation **V&V Class: 3** **Component Type: Other**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
		S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 System Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers **Team G:** 2 Experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No unintended functions	Team A	Extended	A: #CRD = 0 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 6 CA: #CRD ≤ 4 AND #NRD ≤ 6 AND %NRD ≤ 12 AND %UF ≤ 10 R: #CRD > 4 OR #NRD > 6 OR %NRD > 12 OR %UF > 10	R32 P1.0

[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

[3] **SMEs:** Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	User Interface Inspection	S	Adequate Human-Computer Interface	Teams C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R35
3	Operational Concept Analysis	S	Adequate Operational Concept	Teams B, C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R41
4	Formal Customer Review	S	Completeness, No Unfounded Functions, Logic and Control Correct, Data Operations and Computations Correct, General and Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure, Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Engineering Practice, Adequate Knowledge Structure and Contents	Major Team	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R37

#	Method	Type	Assurances	Agents[1]	Intensity	Criteria/Action	Ref
5	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R36 P3.0
6	Random Testing	D	Adequate system performance under random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 4 **CA:** #SS ≤ 1 AND #Maj ≤ 4 AND #Min ≤ 12 **R:** #SS > 1 OR #Maj > 4 OR #Min > 12	R4 P6.0

REFERENCES

R4 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.

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.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing*, IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R37 NUREG/CR-4640, PNL-5784, *Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, August 1987.

R41 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.

Guideline Plan: *for Guideline Package M*

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

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

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

Start Here	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2 3	4	5	6	Yes

GUIDELINE PACKAGE N

GUIDELINE PACKAGE: N

Lifecycle Phase: Implementation **V&V Class: 2** **Component Type: Other**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
		S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 System Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers **Team G:** 2 Experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No unintended functions	Team A	Extended	**A:** #CRD = 0 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 4 **CA:** #CRD ≤ 2 AND #NRD ≤ 4 AND %NRD ≤ 8 AND %UF ≤ 5 **R:** #CRD > 2 OR #NRD > 4 OR %NRD > 8 OR %UF > 5	R32 P1.0

[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

[3] **SMEs:** Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Automated Anomaly Testing (e.g., LINT)	S	Assurance of good software practices, elimination of post compiler syntax errors.	Team E	Limited (with automated tool runs quickly)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R33
3	User Interface Inspection	S	Adequate Human-Computer Interface	Teams C, E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R35
4	Operational Concept Analysis	S	Adequate CONOPS	Teams B, C, E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R41
5	Data Interface Inspection	S	Consistent, complete, compliant, and correct data interfaces	Team B, D, E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R32

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
6	Database Analysis	S	Adequate Database Design	Team D	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R33
7	Process Orientated Audits	S	Adherence to good software engineering principles Adherence to levied standards	Team E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R10 P5.0
8	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R36 P3.0
9	Boundary Testing	D	Assurance that errors do not exist at boundaries and inflection points of input space	Teams F, E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 3 CA: #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 R: #SS > 1 OR #Maj > 2 OR #Min > 8	R31 P4.0

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
10	Formal Customer Review	S	Completeness, No Unfounded Functions, Logic and Control Correct, Data Operations and Computations Correct, General and Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure, Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Engineering Practices, Adequate Knowledge Structure and Contents	Major Team	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R36
11	Random Testing	D	Adequate system performance under random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R4 P6.0
12	Robustness Testing	D	Assurance that system can perform adequately or fail safely for unintended input and operating conditions	Team E, SMEs	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R28

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
13	Validation Scenario Testing	D	Correctness, Consistency Adequate Concepts of Operation Adequate Human-Computer Interface Logic and Control Correct Data Operations and Computations Correct Adequate Knowledge	Team E	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R30 A
14	Regression Testing	D	Assurance that fixes to problems have not introduced new problems	Team E	Moderate to Extended: Run after every major set of fixes.	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 3 **CA:** #SS ≤ 1 AND #Maj ≤ 2 AND #Min ≤ 8 **R:** #SS > 1 OR #Maj > 2 OR #Min > 8	R32

REFERENCES

R4 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.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

R28 Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic 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.

R30a Mirsky, S.M., J.E. Hayes, and L. A. Miller, *Guidelines for Verification and Validation of Expert Systems in the Nuclear Industry. Volume 6 of this report,* Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute, June 18, 1993.

R31 Myers, G.J., *The Art of Software Testing,* Wiley, New York, New York, 1979.

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.

R33 Ng, P., and R. Yeh, (Eds.) *Modern Software Engineering: Foundations and Current Perspectives,* Van Nostrand Reinhold, New York, New York, 1990.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units,* W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing,* IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R41 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.

R46 Schulmeyer, G.G. and J.I. McManus, *Handbook of Software Quality Assurance,* Van Nostrand Reinhold, New York, New York, 1992.

Guideline Plan: *for Guideline Package N*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 14 after all major fixes. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Do _Next	Do Next	Do Next	Exit V&V Activity
1	2	3 4 5 6	7	8	9	10	11	12	13	Yes

GUIDELINE PACKAGE O

GUIDELINE PACKAGE: O

Lifecycle Phase: Implementation **V&V Class: 1** **Component Type: Other**

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action[2]	Ref[1]
		S = Static Testing D = Dynamic Testing	All assurances are considered at the same time when reviewing the Requirements Document with FCR	**Major Team:** 2-IV&V, 2- Users, 2-4 SMEs[3], 1- Editorial Expert, 1-3 Customer Representatives **Team A:** 2 Requirement Analysts **Team B:** 2 System Engineers **Team C:** 1-2 Human Factors Engineers **Team D:** 2 Database Engineers **Team E:** 2 Software Engineers **Team F:** 2 Knowledge Engineers **Team G:** 2 Experienced in Formal Verification methods	**Extended:** *STATIC* - Extra thorough analysis & investigation of system, usually involves detailed modeling & use of automated tools *DYNAMIC* - Extra intensive test-case coverage and sampling.	**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 Tracing	S	Completeness, No unintended functions	Team A	Extended	**A:** #CRD = 0 AND #NRD ≤ 1 AND %NRD ≤ 1 AND %UF ≤ 1 **CA:** #CRD ≤ 1 AND #NRD ≤ 2 AND %NRD ≤ 4 AND %UF ≤ 3 **R:** #CRD > 1 OR #NRD > 2 OR %NRD > 4 OR %UF > 3	R32 P1.0

[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

[3] SMEs : Subject Matter Experts

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
2	Automated Anomaly Testing (e.g., LINT)	S	Assurance of good software practices, elimination of post compiler syntax errors.	Teams E	Limited (with automated tool runs quickly)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R33
3	FMECA (Failure Mode, Effects, Causality Analysis)	S	Containment of hazards and safety problems, "fail-safe" and "soft failure"	Team B and Team E, with SMEs	Extended (Envision and test knowledge structure for all possible failures and causes thereof)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R49
4	Algorithm Analysis (e.g., Trace Assertion Method)	S	Verification of correctness of control and processing algorithms	Team G	Extended (by the very nature of this method)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R30
5	Process Trigger/Timing Analysis	S	Knowledge that control algorithm timing, initiation, and scheduling is correct	Team E, B	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R13 R20

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
.6	User Interface Inspection	S	Adequate Human-Computer Interface	Teams C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R35
7	Operational Concept Analysis	S	Adequate Operational Concept	Teams B, C, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R41
8	Data Interface Inspection	S	Consistent, complete, compliant, and correct data interfaces	Team B, D, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R32
9	Database Analysis	S	Adequate database structure	Team D	Moderate	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R33

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
10	Process Oriented Audits	S	Adherence to good software engineering principles Adherence to levied standards	Team E	Extended	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R10 P5.0
11	Software Practices Review	S	Adequate Software Engineering Practices and Standards Compliance	Team E	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R10 R37
12	System Engineering Review	S	Completeness, Correctness, Adequate System Architecture, Adequate Function Partitioning, Adequate Concept of Operations, Adequate HCI, Adequate Database Design, Contained Safety and Hazard Conditions, Fail-safe/Soft Failure	Team B	Moderate	A: #SS = 0 AND #Maj = 0 AND #Min ≤ 1 CA: #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 R: #SS > 0 OR #Maj > 1 OR #Min > 4	R7 R14

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
13	Formal Customer Review	S	Completeness, No Unfounded Functions, Logic and Control Correct, Data Operations and Computations Correct, General and Interface Consistency, Adequate CONOPS, Adequate HCI, Adequate DB Structure, Compliance with Levied Standards, Proper Operation under Random Input, Adequate System Documentation, Compliance w/ Good Software Engineering Practice, Adequate Knowledge Structure and Contents	Major Team	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R36
14	Functional Testing	D	Knowledge is adequate and correct	Team E	Extended (e.g., multiple tests for each function)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R36 P3.0
15	Boundary Testing	D	Assurance that errors do not exist at boundaries and inflection points of input space	Teams F, E	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R31

#	Method	Type	Assurances	Agents	Intensity	Criteria/Action	Ref
16	Random Testing	D	Adequate system performance under random Input	Team C	Moderate (generate a number of random test cases, but do not perform any already run)	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R4
17	Robustness Testing	D	Assurance that system can perform adequately or fail safely for unintended input and operating conditions	Team E, SMEs	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R28
18	Validation Scenario Testing	D	Correctness, Consistency Adequate Concepts of Operation Adequate Human-Computer Interface Logic and Control Correct Data Operations and Computations Correct Adequate Knowledge	Team E 2 SMEs	Extended	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R30A
19	Regression Testing	D	Assurance that fixes to problems have not introduced new problems	Team E	Moderate to Extended: Run after every major set of fixes.	**A:** #SS = 0 AND #Maj = 0 AND #Min ≤ 1 **CA:** #SS ≤ 0 AND #Maj ≤ 1 AND #Min ≤ 4 **R:** #SS > 0 OR #Maj > 1 OR #Min > 4	R32

REFERENCES

R4 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.

R7 Beizer, B. *Software Testing Techniques,* Van Nostrand Reinhold, New York, New York, 1990.

R10 Bryan, W.L., and S.G. Siegal, *Software Product Assurance: Techniques for Reducing Software Risk,* Elsevier, New York, New York, 1988.

R13 Hatley, D., and I. Pirbhai, *Strategies for Real-Time System Specification,* Dorset House, New York, New York, 1987.

R14 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.

R28 Miller, L.A., *Dynamic Testing of Knowledge Bases Using the Heuristic 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.

R30 Mills, H., V. Basili, J. Gannon, and R. Hamlet, *Principles of Computer Programming: A Mathematical Approach,* Wm. C. Brown, New York, New York, 1987.

R30a Mirsky, S.M., J.E. Hayes, and L. A. Miller, *Guidelines for Verification and Validation of Expert Systems in the Nuclear Industry. Volume 6 of this report,* Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute, June 18, 1993.

R31 Myers, G.J., *The Art of Software Testing,* Wiley, New York, New York, 1979.

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.

REFERENCES (Conintued)

R33 Ng, P., and R. Yeh, (Eds.) *Modern Software Engineering: Foundations and Current Perspectives*, Van Nostrand Reinhold, New York, New York, 1990.

R35 NUREG/CR-4227, *Human Engineering Guidelines for the Evaluation and Assessment of Video Display Units*, W. Gilmore, July 1985.

R36 Howden, W.E., *Functional Program Testing*, IEEE Transactions on Software Engineering. WE-6(2), pp. 162-169, March 1980.

R37 NUREG/CR-4640, PNL-5784, *Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, August 1987.

R41 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.

R49 United Kingdom Ministry of Defense Draft Interim Defense 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, 1989.

Guideline Plan: *for Guideline Package O*

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

Complete the methods in each column before proceeding to the next. You may perform the methods within a column in any order. Run Method 19 after all major fixes. 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	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Do Next	Exit V&V Activity
1	2	3 4 5 6 7 8 9	10 11	12	13	14	15	16	17	18	Yes

GUIDELINE PACKAGE P

GUIDELINE PACKAGE: P

Lifecycle Phase: Maintenance Modification **V&V Class: All** **Component Type: All**

Note: The V&V maintenance situation can range from a minuscule correction of a typographical error in a displayed message to a complete overhaul of a system. It can begin in a maintenance requirements phase and run through design and implementation or just be for implementation. It could cover any of the three types of components. It could occur for any of the three classes of V&V. Thus, all 27 V&V situations could occur, and all 15 guideline packages could apply.

Accordingly, the V&V of maintenance activity will call for a good deal of judgment as supported by the following set of guidelines:

1. Encourage the developers to go through a maintenance requirements and design phase, and participate in these phases.
2. In accomplishing maintenance V&V, use as a guide the guideline package corresponding to the characteristics of the maintenance activity (e.g., for testing the whole implemented maintenance change to an entire Class 1 system, use guideline package O).
3. If at all possible, execute all the recommended V&V methods of the appropriate package, even though the maintenance changes might be quite limited. It is in the maintenance activities, which are usually much less carefully planned and executed, that the most disastrous and expensive problems can be introduced.
   Even though all the methods will be applied, they need focus primarily only on the system areas which were changed, and therefore the time to complete the methods should be less.
4. If the budget or schedule does not permit full-scale maintenance then cut back according to the following principles:

   4.1 Reduce or eliminate structural testing, reduce functional testing, reduce desk-checking
   4.2 Keep some kind of group review process because it greatly increases the chances of perceiving the flaws
   4.3 Use computed-automated tools whenever possible
   4.4 Do some random and robustness testing
   4.5 Do extensive regression testing (if the test suites are still valid; else do overall system functional testing)

SECTION 3: GUIDELINE PROCEDURES
OVERVIEW TO DETAILED V&V PROCEDURES

This section provides detailed step-by-step actions for a small number of the most important V&V procedures recommended in the Guideline Packages: Requirements Tracing, Desk Checking, Functional Testing, Boundary/Domain Testing, Process-Oriented Audits, Random Testing, Semantic/Meta-Level Knowledge Checking, Robustness Testing, Regression Testing, Automated Syntax Checking of Knowledge Structures, and Formal Customer Review.

The V&V guideline procedures provide information on the step-by-step execution of a specific V&V method. A sample template for a guideline procedure is shown in Figure S3-1. Some guideline procedures are self-contained. Others are broken into several well-defined activities. The format of the lower level activities is identical to that of the highest level procedures. Throughout the procedures, a hollow bullet (●) refers to an optional step, whereas a filled bullet (●) refers to a mandatory step. The guideline procedures are broken into the following sections:

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

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

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

PROGRAM MANAGER APPROVAL - Some guideline procedures represent such large undertakings that Program Management approval is recommended prior to commencement.

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

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

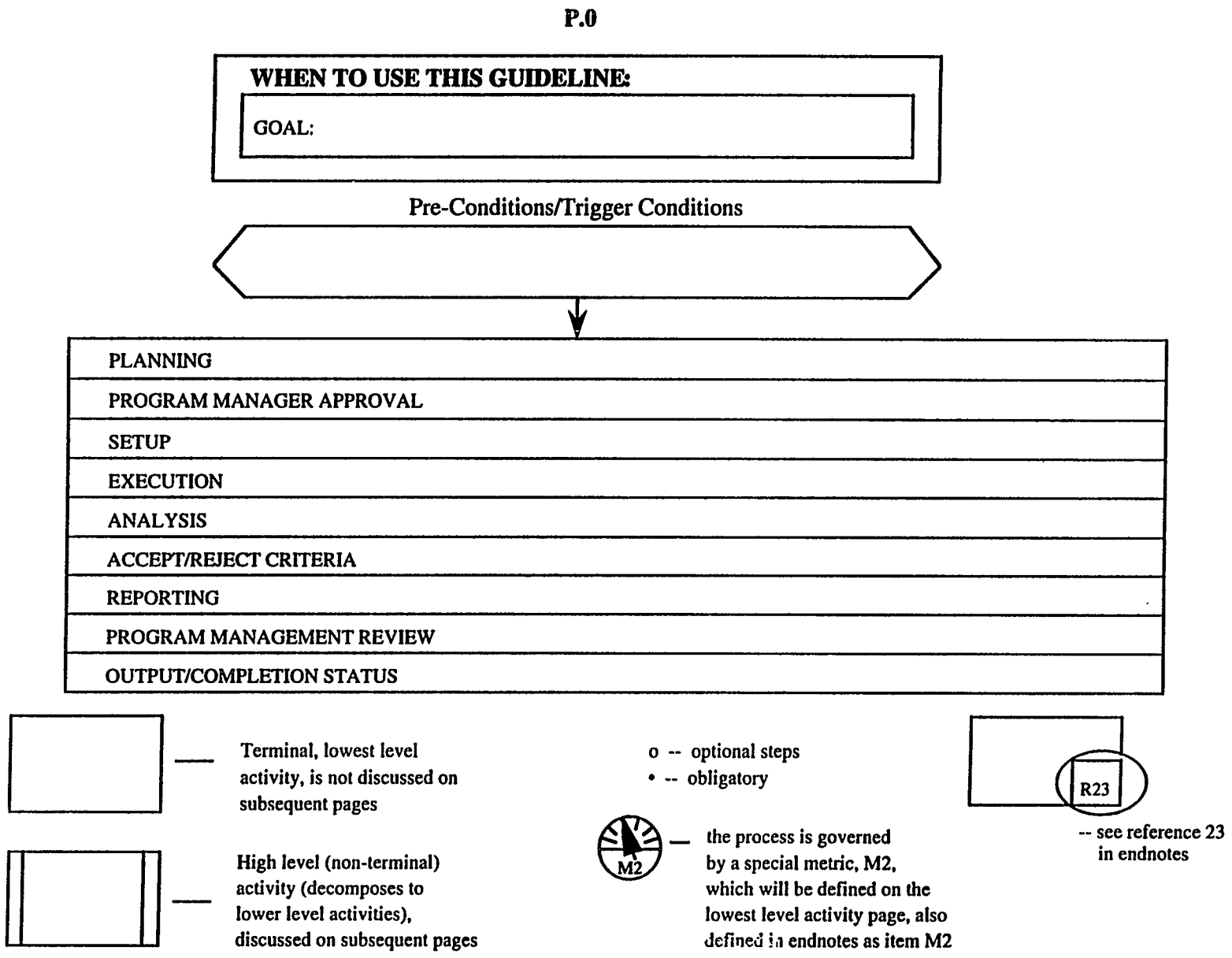


Figure S3-1 Sample V&V Guideline Procedure

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, which are located at the end of the entire procedure (e.g., the endnotes for the entire procedure 1.0 are located after P1.8). 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 also be found in the endnotes.

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 any other suggested analysis that could be performed. As in the Execution section, metrics and/or references may be found here.

ACCEPT/REJECT CRITERIA - For top-level guideline procedures, there are three possible results: ACCEPT, CONDITIONALLY ACCEPT, and REJECT. 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. It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

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

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

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.

GUIDELINE PROCEDURES

P1.0 Requirements Tracing/Traceability Analysis
- P1.1 Identify Code Elements
- P1.2 Assign Unique Identifier to Code Elements
- P1.3 Categorize Code Elements (Grouping)
- P1.4 Perform Traceability Analysis
- P1.5 Identify Non-Traceable Code Elements (Unintended Functions)
- P1.6 Perform Completeness Analysis
- P1.7 Identify Unsatisfied Requirements (Unimplemented Requirements)
- P1.8 Prepare Requirements Tracing Report

P2.0 Desk Checking
- P2.2 Manually "Execute" Code
- P2.5 Look for Subset of Defect Types
- P2.9 Checklist Code Review
- P2.10 Prepare Code Analysis Report

P3.0 Functional Testing (Black Box)

P4.0 Boundary/Domain Testing

P5.0 Process-Oriented Audits

P6.0 Random Testing

P7.0 Semantic/Meta-Level Knowledge Checking
- P7.1 Develop Meta-Level Engineering Database
- P7.3 Execute Meta-Level Knowledge Checking

P8.0 Robustness Testing

P9.0 Regression Testing

P10.0 Automated Syntax Checking of Knowledge Structures

P11.0 Formal Customer Review

GUIDELINE PROCEDURE P1.0

P1.0 Requirements Tracing/Traceability Analysis

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform requirements tracing/tracebility analysis for a requirements specification and the code. A requirements specification has been received and has been the subject of requirements analysis. The analysis has successfully resulted in explicit labeling of all independent requirements and a priority has been assigned to each. A design document may or may not exist, but source code has been received (and/or detailed documentation describing the code). Use this guideline as part of the development process or to perform an audit.

Note: Requirements tracing is not an effective method for tracing performance and reliability requirements to the code.

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Resources are available to perform the activity
- Source code has been delivered
- 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.)
- Prepare informal requirements tracing plan showing tasks, schedules, resources, etc.
- Select tools to be used (e.g., word processing package, software for scanner, etc.)

PROGRAM MANAGER APPROVAL

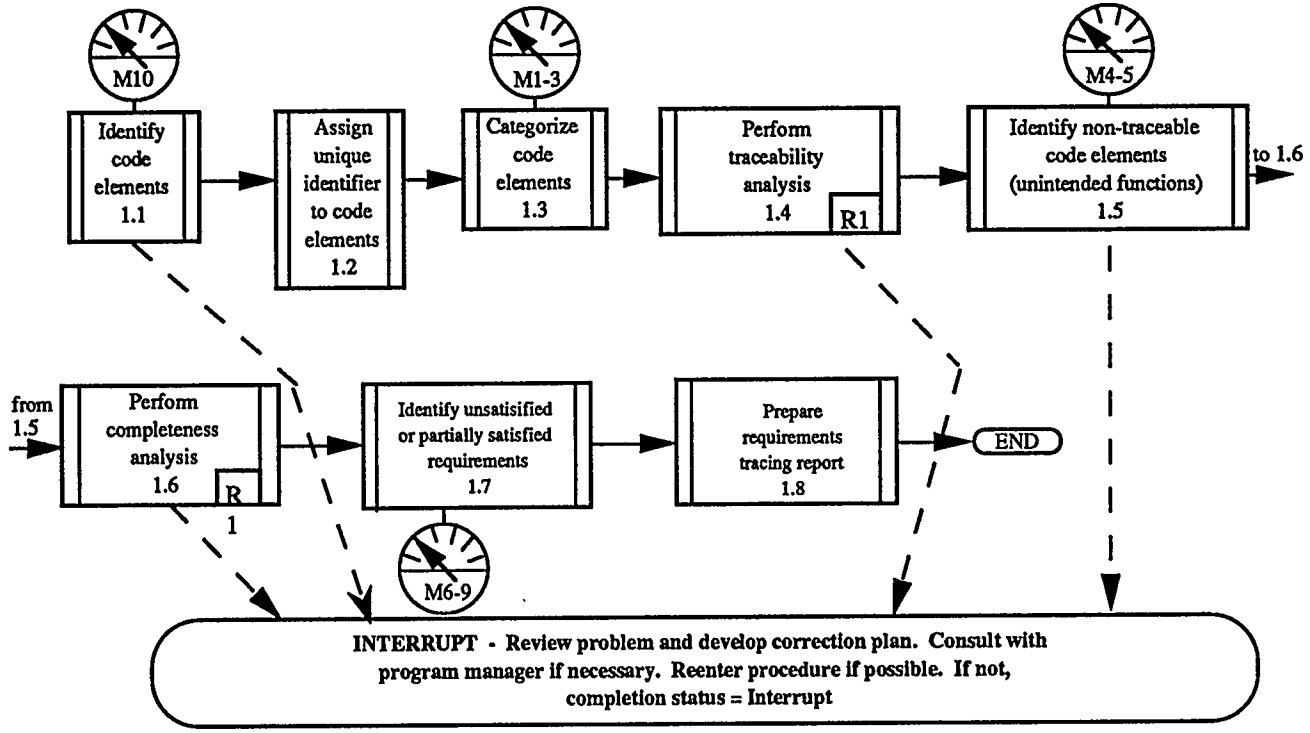
- The informal requirements tracing plan is presented to the Program Manager
- Approval should be given by Program Manager before work commences

SETUP

- Obtain requirements specification in softcopy from developer (ASCII format required). If this is not possible, scan the hardcopy document using an optical scanner and Commercial Off The Shelf (COTS) software
- Ensure all human resources have requirements tracing plan, hard copy of the requirements specification, softcopy of the requirements specification, a computer, and word processing software

*Requirements tracing tools were found to greatly improve this activity. However, this procedure has been written so that no special tools are required for its use. It is expected that word processing packages and database management systems are now universally owned, while hypermedia, SGML, or requirements tracing tools are not.

EXECUTION



ANALYSIS

- The % of code elements which are not traceable (unintended functions) or which are not completely satisfied will be determined (unimplemented)
- The % of critical requirements which are not completely satisfied will be determined (unimplemented critical requirements)
  - Each critical requirement which is anomalous will be analyzed for possible impact to the program. "Impact statements" will be prepared for each
  - Several contingency plans will be prepared describing how the developer can bring the outstanding requirements tracing issues to closure

ACCEPT/ REJECT CRITERIA

- Accept if: (M6+M7)=0 and (M8+M9) ≤2% and M5 ≤1%; Completion status = Accept
- Conditionally Accept if: 0<(M6+M7)≤2% and 2<(M8+M9)≤ 5% and 1<M5≤2%; Completion status =Conditionally Accept
- Reject if: (M6+M7)>2% or (M8+M9)>5% or M5>2%; Completion status = Reject

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare V&V requirements tracing report
  - ° Prepare "impact statements" for critical requirements (could be appendix to the report)
  - ° Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the requirements tracing deficiencies

PROGRAM MANAGER REVIEW

- Present the V&V requirements tracing report to the Program Manager
  - ° Present the informal "Plan of Action" for correcting the deficiencies to the Program Manager
- V&V tracing report cannot be given to the developer until Program Manager approval given
  - ° Request that a Technical Interchange Meeting between V&V and the developer be held to ensure that the developer understands the V&V report and that V&V understands the developer's plans for correcting deficiencies

OUTPUT/ COMPLETION STATUS

if ACCEPT	→ continue to next activity
if CONDITIONALLY ACCEPT	→ correct deficiencies and repeat this activity
if REJECT	→ hold review with program manager to determine course of action, correct deficiencies
if INTERRUPT	→ develop correction plan and reenter procedure if possible

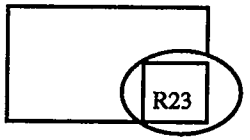
Terminal, lowest level activity, is not discussed on subsequent pages

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

P1.1 IDENTIFY CODE ELEMENTS

GOAL OF THIS ACTIVITY:

The goal of this activity is to identify the code elements which will be traced to the requirements specification.

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Resources are available to perform the activity
- Source code has been delivered
- Schedule dictates that activity commence

PLANNING

- Determine the level at which code elements will be identified (units, modules, functions, low level CSC, etc.)
- Ensure that necessary resources are available (computer, printer, etc.)

PROGRAM MANAGER APPROVAL

N/A for this step (covered by Requirements Tracing 1.0)

SETUP

N/A for this activity (covered by Requirements Tracing 1.0)

EXECUTION

- Using the source code and/or detailed design documentation and the determined code element level (from planning), identify code elements:
  - 1) Use the table of contents of the designed document to identify code elements:
  - 2) Use the source code listing to identify code elements
  - 3) Construct a list of the code elements, consisting of code element name, page number, section number, section title, and a short description R2
  - 4) Compare the code element list to the table of contents of the design document or to a listing of the source code modules to ensure the code element list is complete.

ANALYSIS

- The % of code elements which are marked as "TBD" in the design document (if applicable) shall be determined [if excessive (> 2%),INTERRUPT]

ACCEPT/ REJECT CRITERIA

- Accept if: $0 < M10 \leq 2\%$; completion status = ACCEPT
- Reject if: $M10 > 2\%$; completion status = REJECT

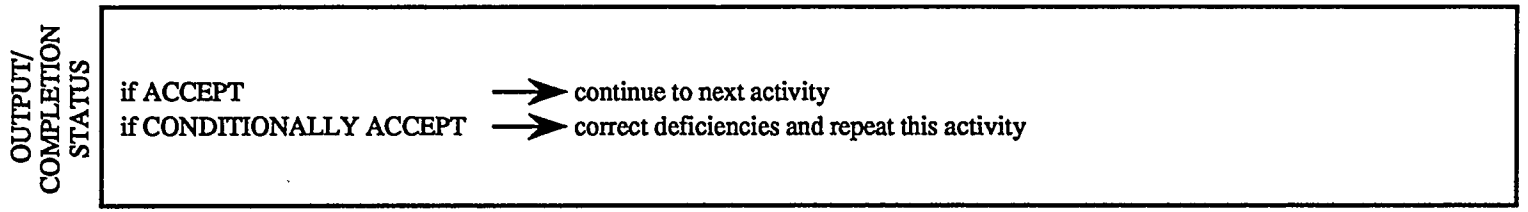
Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Code element list prepared

PROGRAM MANAGER REVIEW

N/A for this activity



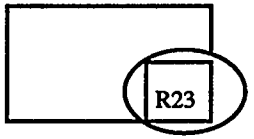
Terminal, lowest level activity, is not discussed on subsequent pages

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

P1.2 ASSIGN UNIQUE IDENTIFIER TO CODE ELEMENTS

GOAL OF THIS ACTIVITY:

The goal of this activity is to assign a unique identifier to each of the code elements identified in activity 1.1.

Pre-Conditions/Trigger Conditions

- Code elements have been identified

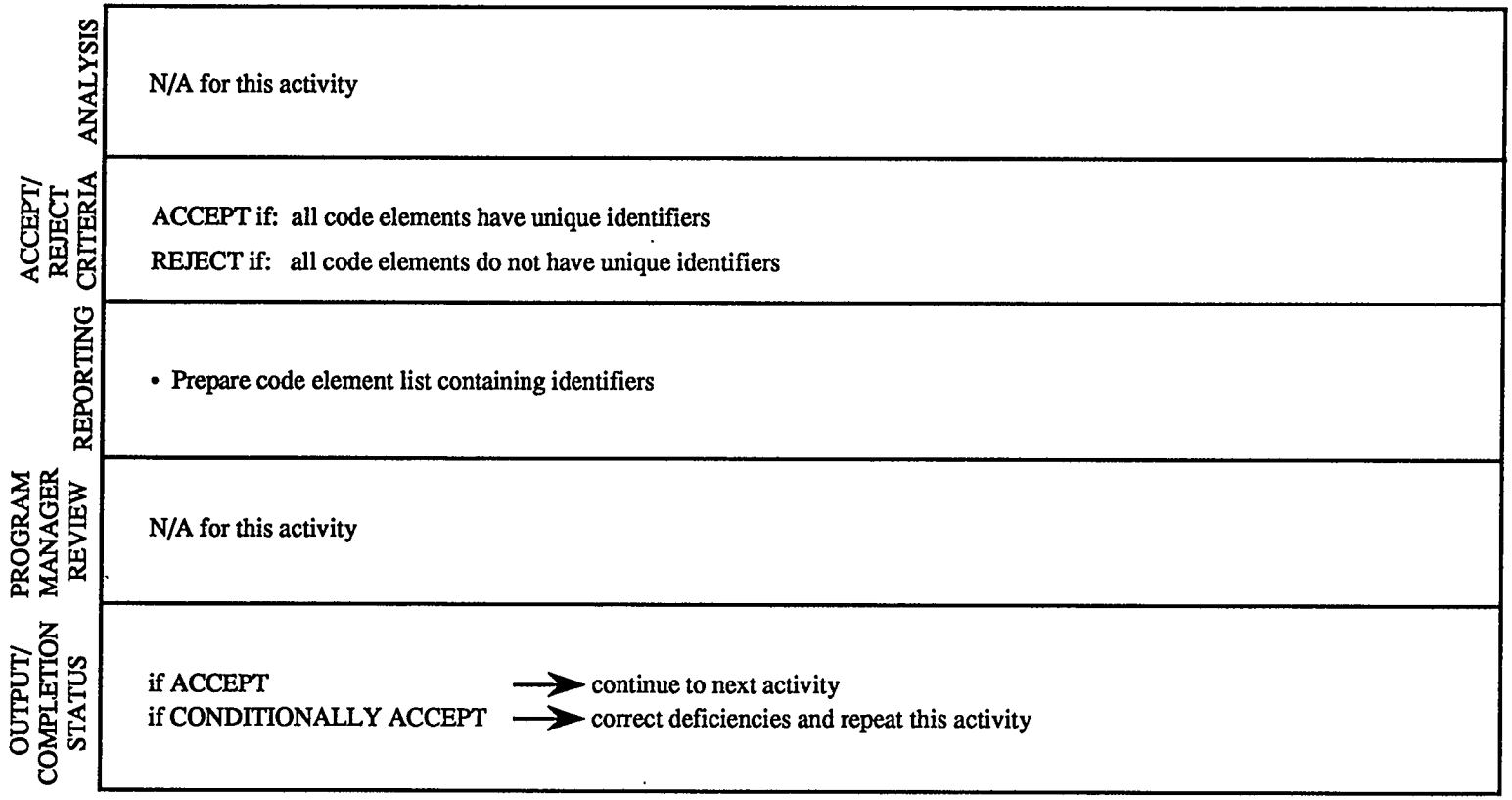
PROGRAM MANAGER APPROVAL	N/A for this activity (covered by Requirements Tracing 1.0)
SETUP	N/A for this activity (covered by Requirements Tracing 1.0)

PLANNING

- Develop identification scheme (e.g., all units will be identified by a sequential identifier AAXXXX where AA defines the document and XXXX is a sequential value)
- Ensure that necessary resources are available (computer, printer, etc.)

EXECUTION

- 1) Using the selected identification scheme (from planning step), assign a unique identifier to each code element
- 2) Add the identifiers to the code element list prepared in activity 1.1
- 3) Examine the code element list to ensure that all code elements have a unique identifier

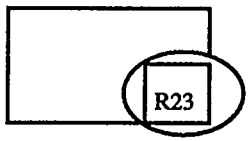


Terminal, lowest level activity, is not discussed on subsequent pages

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

P1.3 CATEGORIZE CODE ELEMENTS (GROUPING)

GOAL OF THIS ACTIVITY:

The goal of this activity is to categorize group code elements identified and numbered in activities 1.1 and 1.2.

Pre-Conditions/Trigger Conditions

- Categories list for requirements specification is available*
- All code elements have a unique identifier

PLANNING	• Categories can be derived from many sources: a) types of requirements listed in DoD-STA-2167A or IEEE 830-1984 (e.g., language requirements, functional requirements, etc.); b) table of contents of the system specification or requirements specification, etc. • Ensure that necessary resources are available (computer, printer, etc.)
PROGRAM MANAGER APPROVAL	N/A for this activity (covered by Requirements Tracing 1.0)
SETUP	N/A for this activity

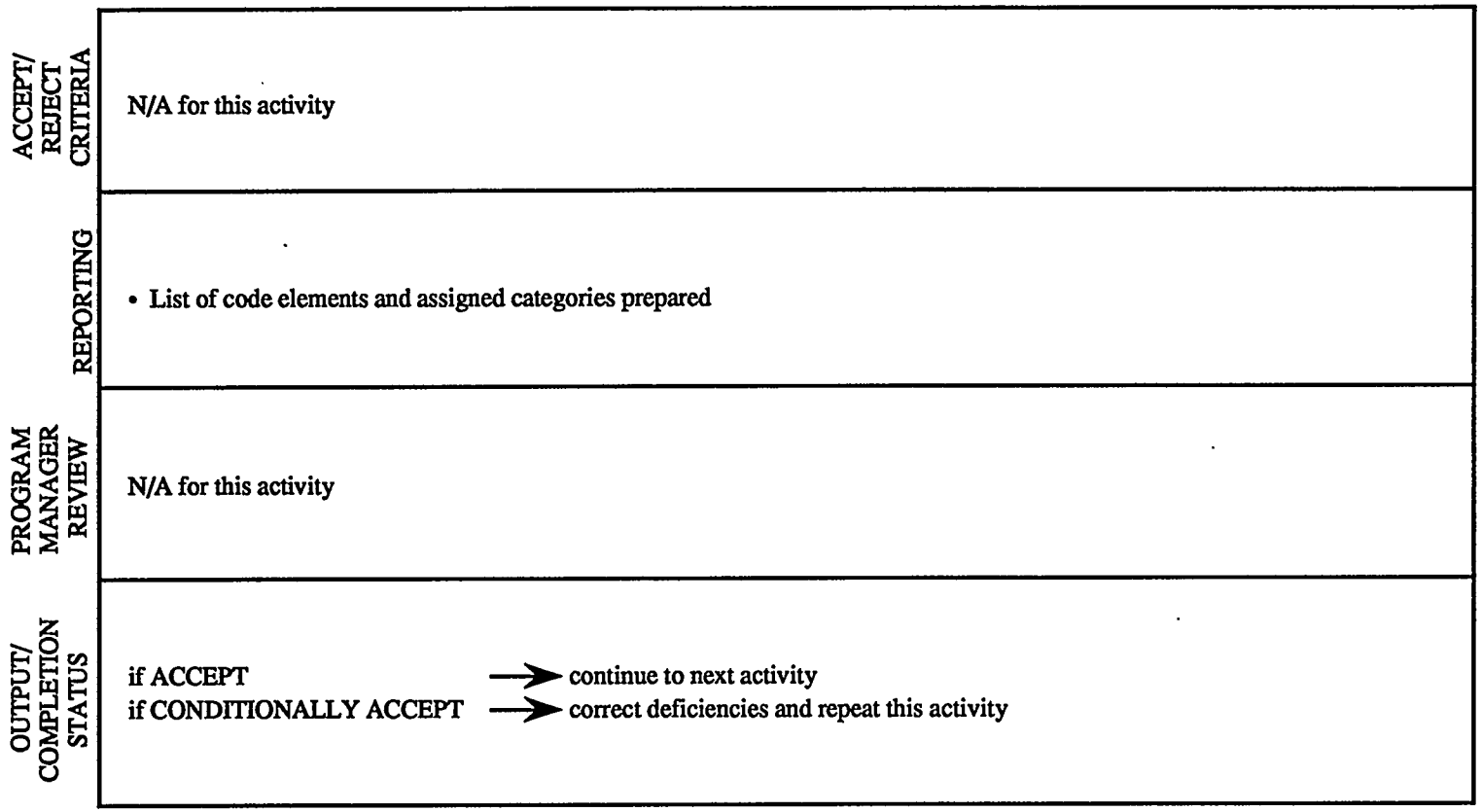
*An output of the requirements analysis process is a list of derived requirements.

EXECUTION

- 1) Determine the categories that will be assigned (on-line help, ad hoc query, etc.). These should be identical to the already existing category list used to assign categories to the requirements. To perform the trace, lower level categories may be required. If this is the case, extend the existing category list hierarchically.
- 2) Using the category list and associated criteria, assign categories to each code element. From reading the documentation associated with the code element, as well as examining the code element itself, infer the primary function(s) accomplished by the element. Then review the categories list and determine which categories might pertain to this code element. Before stopping with this element, examine the code closely to see if the element is facilitating, possibly in a subtle way, any other categories.
- 3) Build a list showing which categories apply to which code elements. The format could be the code element identifier followed by its assigned categories (separate categories with a space, comma, or some delimiter).
- 4) Ensure that all code elements have at least one category assigned to them.
- ° 5) Have several analysts review the assigned categories and modify them appropriately until concurrence.

ANALYSIS

- The number of critical code elements shall be determined (count the number of code elements with the category "critical")
- The total number of code elements shall be determined (look at the highest identifier)
- The % of critical code elements shall be determined $\left(\frac{M1}{M3} * 100 \right)$

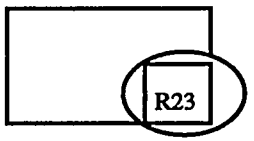


Terminal, lowest level activity, is not discussed on subsequent pages

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

P1.4 PERFORM TRACEABILITY ANALYSIS

GOAL OF THIS ACTIVITY:

The goal of this activity is to trace the code elements identified in activity 1.1 to the requirements specification.

Pre-Conditions/Trigger Conditions

- Code elements have been categorized

PLANNING

- Ensure that necessary resources are available (computer, printer, etc.)
- ° Select a relational database management system to be used*

PROGRAM MANAGER APPROVAL

N/A for this activity (covered by Requirements Tracing 1.0)

SETUP

N/A for this activity

* Requirements tracing tools were found to greatly improve this activity. However, this procedure has been written so that no special tools are required for its use. It is expected that word processing packages and database management systems are now universally owned, while hypermedia, SGML, or requirements tracing tools are not.

EXECUTION

1.4.1 • 1) From reading the categories associated with a code element, and perusing the requirements listing (and categories associated with them), R4 decide which requirements pertain to this code element. Also, examine the requirements to see if the code element is facilitating, even in a minor way, any other requirements.

a) There will likely be more than one code element tracing back to the same requirement, and a requirement might trace to more than one code element also.

b) A "matching ratio" can be used in finding what requirements a code element traces back to . For example, a ratio of 0.5 means that if half of a code element's categories match with the categories of a requirement (or vice versa), the code element will be said to trace to that requirement:

Requirement Z Category A, Category B
Requirement Y Category B, Category C, Category D

Code Element T Category A
Code Element U Category B, Category E

Therefore:
- Code Element T will trace back to Requirement Z (100% of its categories also are assigned to Requirement Z).
- Code Element U will trace back to Requirement Y (50% (1 of 2) of its categories are assigned also to Requirement Y).

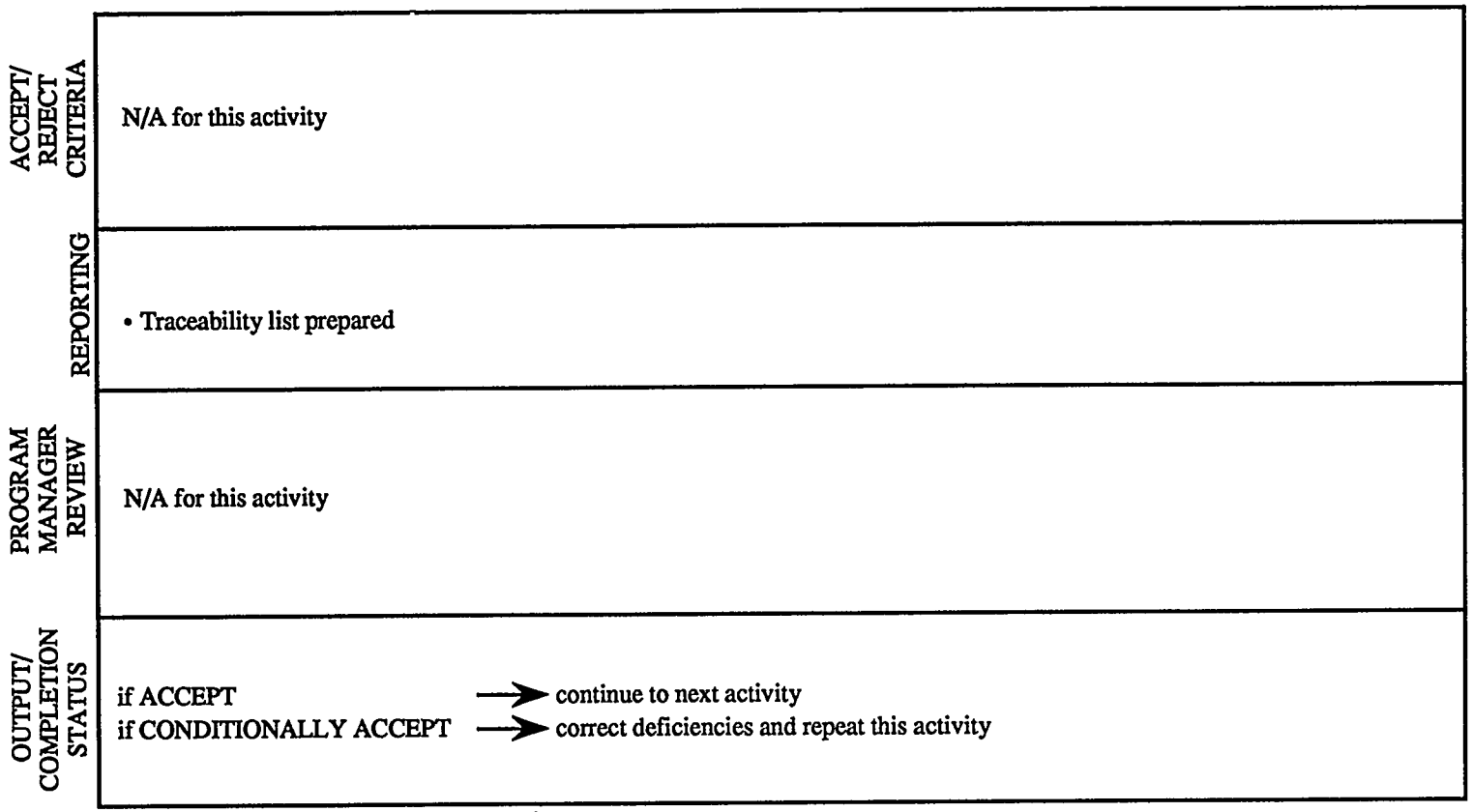
Any type of scheme could be used to apply the categories in determining what requirements a code element traces to. [if an unreasonably small number of code elements are tracing back to requirements, INTERRUPT]

1.4.2 • 2) Build a traceability list consisting of the code element IDs with their associated requirement IDs R5

1.4.3 o 3) Have several analysts review the traceability list and modify it appropriately until concurrence.

ANALYSIS

N/A for this activity

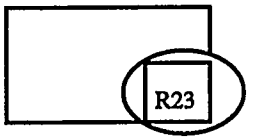


Terminal, lowest level activity, is not discussed on subsequent pages

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

P1.5 IDENTIFY NON-TRACEABLE CODE ELEMENTS (Unintended Functions)

GOAL OF THIS ACTIVITY:

The goal of this activity is to identify derived code elements* as well as the code elements which do not trace back to any requirements.

Pre-Conditions/Trigger Conditions

- Traceability analysis has been performed

PLANNING	• Determine criteria for "derived" functions (should be in line with criteria already established for derived requirements during the earlier requirements analysis) • Ensure that necessary resources are available (computer, printer, etc.)
PROGRAM MANAGER APPROVAL	N/A for this activity (covered by Requirements Tracing 1.0)
SETUP	N/A for this activity

* An output of the requirements analysis process is the set of derived requirements.

EXECUTION

- Using the traceability list, R 5 identify non-traceable code elements (derived functions and unintended functions):
  - 1) Search the traceability list for all code elements which have no requirements listed.
  - 2) Use the criteria for a derived code element (established in the planning step), and determine if the code element is derived. If so, note this on the traceability list.
  - 3) If the code element is not derived, look through the requirements list again to ensure there is no code element.
  - 4) Build a list of all code elements not derived and not tracing back to a requirement (unintended functions). R 6
  - 5) Have several analysts review the unintended functions list and modify appropriately until concurrence.

ANALYSIS

- The number of non-tracing code elements (unintended functions) is determined 

- The % of unintended functions shall be determined $\left(\frac{M4}{M3} * 100\right)$ 

[if M5 > 2%, INTERRUPT*]

ACCEPT/ REJECT CRITERIA

- ACCEPT if: M5= 0; Completion Status = ACCEPT*
- REJECT if: 0 < M5≤2%; Completion Status = REJECT*

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Unintended function list prepared

PROGRAM MANAGER REVIEW

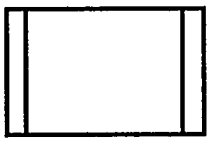
° Informally present the results of traceability analysis to the Program Manager

OUTPUT/ COMPLETION STATUS

if ACCEPT → continue to next activity
if CONDITIONALLY ACCEPT → correct deficiencies and repeat this activity

*Note: If revising code, these criteria must be tailored appropriately.

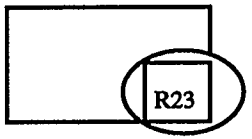
Terminal, lowest level activity, is not discussed on subsequent pages



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

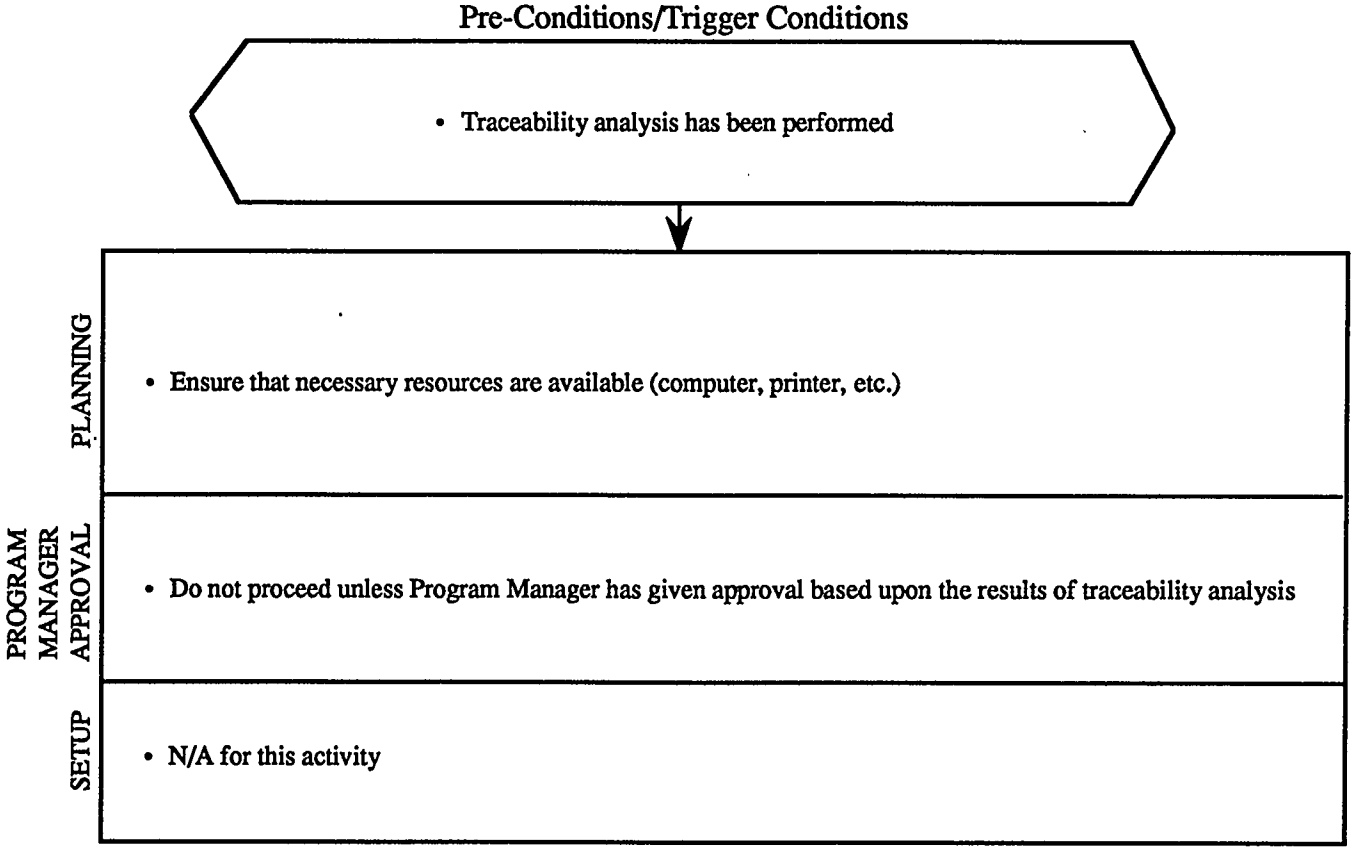


-- see reference 23 in endnotes

P1.6 PERFORM COMPLETENESS ANALYSIS

GOAL OF THIS ACTIVITY:

The goal of this activity is to list, by requirement ID, the code elements which trace back to the requirements specification.



EXECUTION

1.6.1

- 1) Using the traceability list R5 , prepare a list of code elements by requirement traced to

  C01000<TAB>R10001,R10002

  •
  •
  •

  Will be used to build a list looking at the requirements: R10001<TAB>C01000, R10002<TAB>C01000.
  This list will be the completeness list. R7

  [If an unreasonably large number of requirements have no code element listed, INTERRUPT]

1.6.2

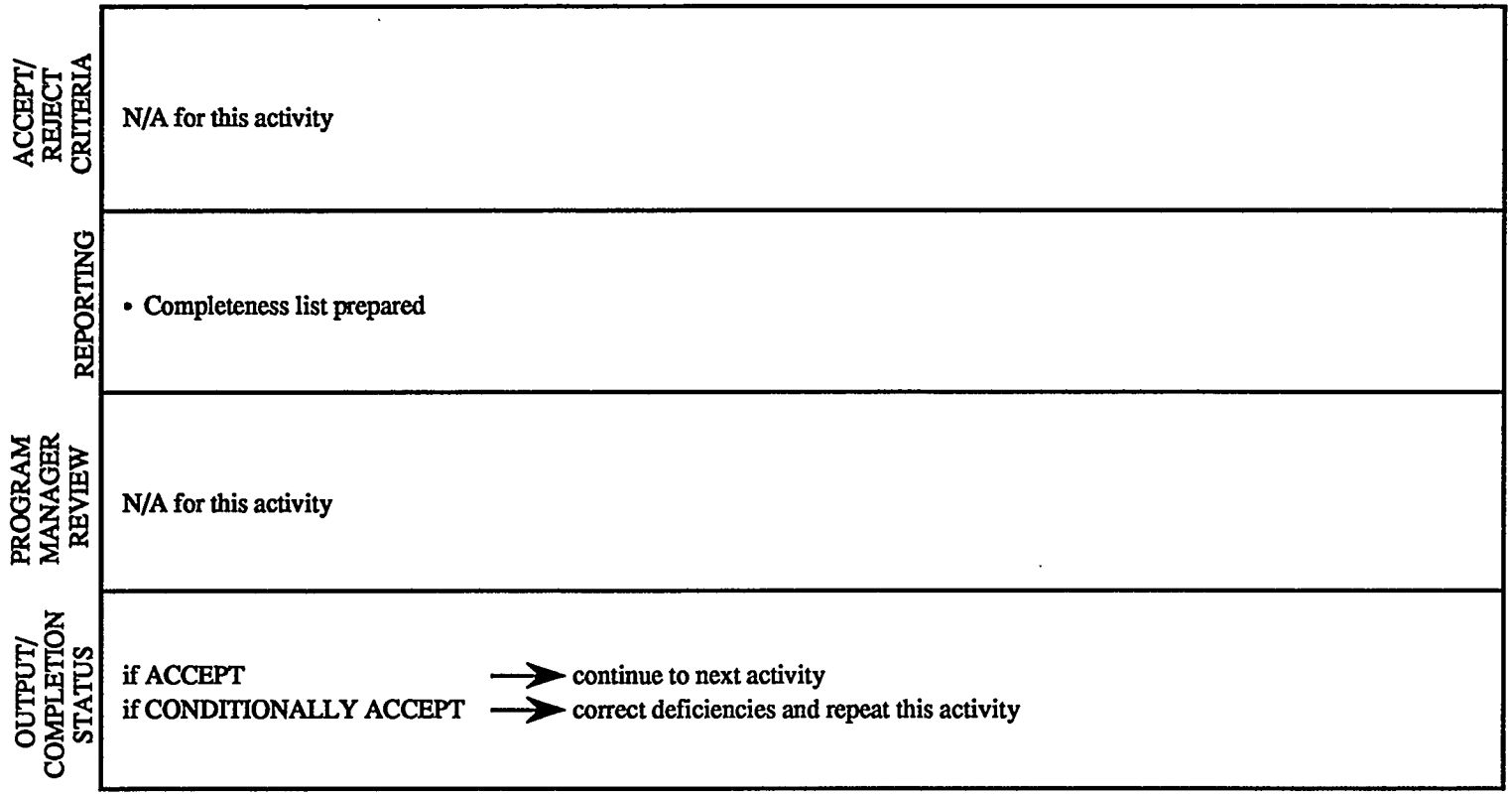
- 2) Using the code element list, requirements list, and the completeness list, examine the requirements and the associated code elements. Ensure that requirements flow down to the code elements listed. Make any necessary adjustments to the completeness list.

1.6.3

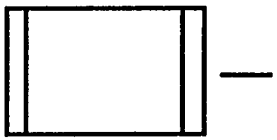
° 3) Have a team of analysts review the completeness list and make appropriate adjustments until concurrence.

ANALYSIS

N/A for this activity



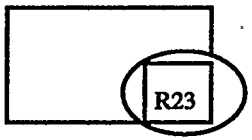
Terminal, lowest level activity, is not discussed on subsequent pages



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

P1.7 IDENTIFY UNSATISFIED REQUIREMENTS (Unimplemented Requirements)

GOAL OF THIS ACTIVITY:

The goal of this activity is to determine whether requirements are completely stisfied, partially satisfied, or not satisfied by the code elements traced to them during activity 1.4.

Pre-Conditions/Trigger Conditions

- Completeness analysis has been performed

PLANNING	• Ensure that necessary resources are available (computer, printer, etc.)
PROGRAM MANAGER APPROVAL	N/A for this activity
SETUP	N/A for this activity

EXECUTION

- Using the completeness list, identify unsatisfied and partially satisfied requirments (unimplemented requirements):
  - 1) Using the actual code listing or design document, the actual requirements specification, and the completeness list, determine the satisfaction of each requirement:
    a) Look at the first entry in the completeness list (e.g., requirement B10010 traces down to code elements C01000 and C01009). Read the text of the requirement (for B10010).
    b) Examine the relevant documentation for the first code element (C01000). Examine the code itself. Continue this process for all code elements (also C01009).
    c) Determine if the code elements fully satisfy the requirement.
  - 2) Build a list of satisfied requirements [R8] and a list of requirements which are only partially satisfied [R9] by the code elements. For requirements which are only partially satisfied, add a comment to this list explaining which aspect of the requirement is not covered.
  - 3) Build a list of all requirements in the completeness list which have no code elements listed. [R8]

ANALYSIS

- The number of unsatisfied (unimplemented requirements) is determined

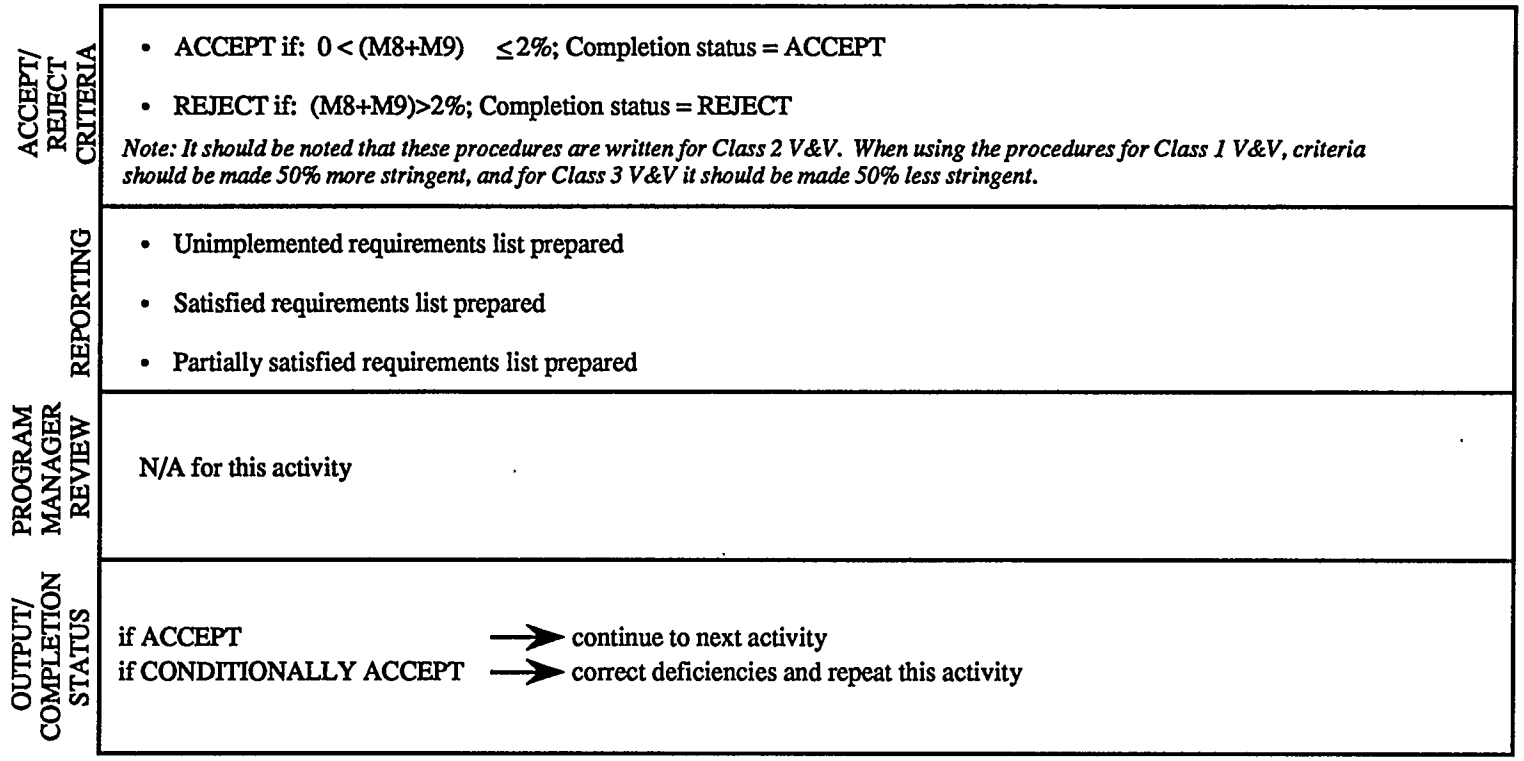
- The number of partially satisfied requirements is determined

- The % of unsatisified requirements is determined

$$\left[\frac{M6}{\text{Total \# Reqts.}} * 100\right]$$

- The % of partially satisfied requirements is determined

$$\left[\frac{M7}{\text{Total \# Reqts.}} * 100\right]$$

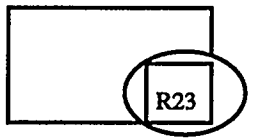


Terminal, lowest level activity, is not discussed on subsequent pages

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

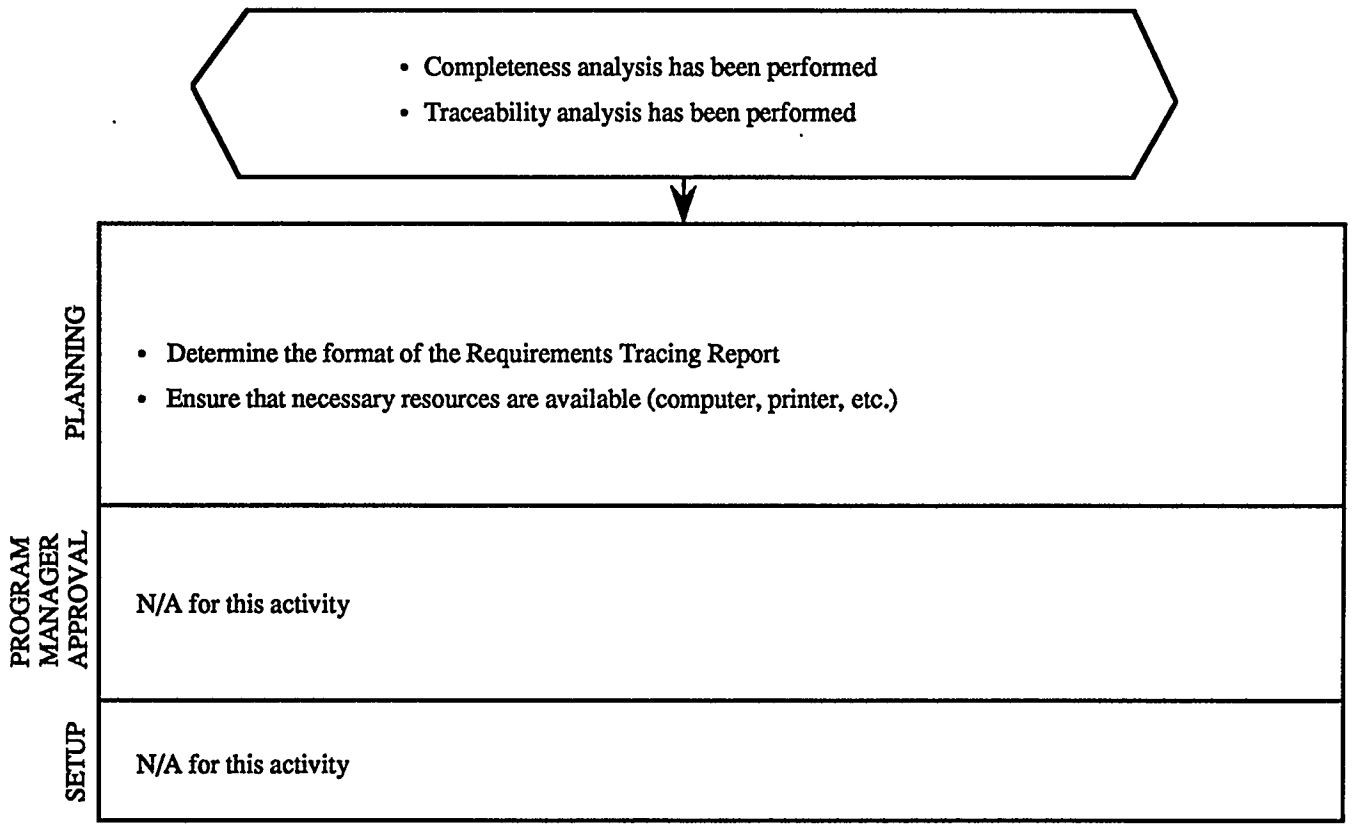


-- see reference 23 in endnotes

P1.8 PREPARE REQUIREMENTS TRACING REPORT

GOAL OF THIS ACTIVITY:

The goal of this activity is to document the results of requirements tracing/traceability analysis in a V&V report.



EXECUTION

- Using the lists generated in the previous steps (see below), prepare a Requirements Tracing Report:

  - Code Element List
  - Code elements with assigned categories
  - Traceability list
  - Unintended function list
  - Completeness list
  - Unimplemented requirement list
  - Satisfied requirement list
  - Partially satisfied requirement list

- 1) Prepare a table describing the results of requirements tracing, e.g.:

Requirements Specification to Code Elements	Total # Requirements	Number (%) Satisfied	Number Partially Satisfied	Number (%) Unimplemented
	1000	600 (60%)	350 (35%)	50 (5%)

Code Elements to Requirements Specification	Total # Code Elements	Number (%) Traced	Number (%) Derived	Number (%) Unintended Functions
	10,000	9,500 (95%)	200 (2%)	300 (3%)

- 2) Using the table as base information, prepare the report, possibly following this outline:

  Executive Summary
  1.0 INTRODUCTION
  2.0 SCOPE - Describe the documents/levels traced, the IDs used, etc.
  3.0 Summary of Results
    Tables (as above in Step 1)
    reference Appendices
  4.0 Conclusions and Recommendations
  Appendices
    - Code element list
    - Requirement list
    - Unintended function list
    - Unimplemented requirement list
    - Partially satisfied requirement list

ANALYSIS

N/A for this activity

ACCEPT/ REJECT CRITERIA

N/A for this activity

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Code analysis report prepared

PROGRAM MANAGER REVIEW

- Present the V&V Requirements Tracing Report to the Program Manager
- ° Present the informal "Plan of Action" for correcting the deficiencies to the Program Manager
- V&V tracing report cannot be given to the developer until Program Manager approval given
- ° Request that a Technical Interchange Meeting between V&V and the developer be held to ensure that the developer understands the V&V report and that V&V understands the developer's plans for correcting deficiencies

OUTPUT/ COMPLETION STATUS

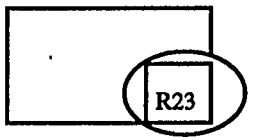
if ACCEPT → continue to next activity
if CONDITIONALLY ACCEPT → correct deficiencies and repeat this activity

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

M1 - Number of critical code elements
M2 - Percentage of critical code elements
M3 - Total number of code elements
M4 - Number of unintended functions
M5 - Percentage of unintended functions
M6 - Number of unsatisfied (omitted) requirements
M7 - Number of partially satisfied requirements
M8 - Percentage of unsatisfied (omitted) requirements
M9 - Percentage of partially satisfied requirements
M10 - Percentage of "TBD" code elements

R1 - Alford, M. "SREM at the age of eight: The distributed computing design system", 1985, 18(4), 36-46.

R2 - ANSI/IEEE-7-4.3.2-1982 "Application Criteria for Programmable Digital Computer Systems of Nuclear Power Generating Stations", July 6, 1982.

R4 - Barnes, M., P. Bishop, B. Bjarland, G. Dahl, D. Esp, J. Lahti, H. Valisue, P. Humphreys. "Software Testing and Evaluation Methods (The STEM Project)." OECD Halden Reactor Project Report, No. HWR-210, May 1987.

R5 - Beizer, B. Software Testing Techniques, Second Edition. New York: Van Nostrand Reinhold, 1990.

R6 - Bernard, John, and Tahaski Washio. "Expert Systems Applications Within the Nuclear Industry." American Nuclear Society, La Grange Park, ILL, 1989.

R7 - Blanchard, B.S. and W.J. Fabrycky. Systems Engineering and Analysis. Englewood Cliffs, NJ: Prentice-Hall, 1990.

R8 - Boehm, Barry W. "A Spiral Model of Software Development and Enhancement." COMPUTER, IEEE Computer Society, May 1988, 61-72.

R9 - Boehm, B.W. Software Risk Management. New York: Academic Press, 1990.

GUIDELINE PROCEDURE P2.0

2.0 DESK CHECKING

WHEN TO USE THIS GUIDELINE:

GOAL: To perform desk checking on the system source code. Source code has been received. Use this guideline as part of the development process or to perform an audit.

Pre-Conditions/Trigger Conditions

- Resources are available to perform the activity
- Source code has been delivered
- Schedule dictates that activity commence

PLANNING

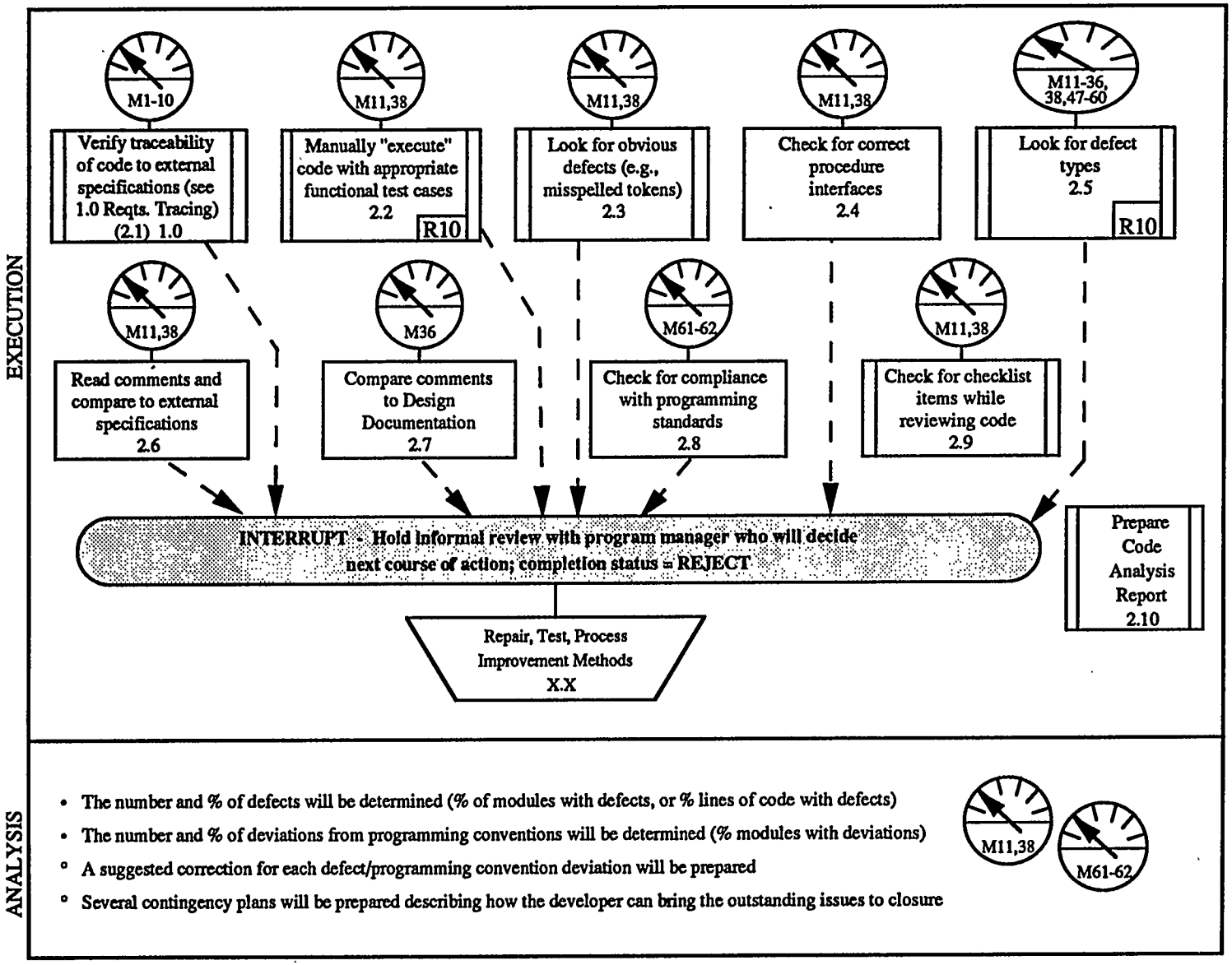
- Determine the high level tasks for this activity (see Execution below)
- Establish a schedule for the activity
- Assign human resources to the high level tasks
- Ensure that human resources agree that schedule is reasonable
- Ensure that other necessary resources are available (computer, printer, etc.)
- Prepare informal desk checking plan showing tasks, schedules, resources, etc.
- Select tools to be used (e.g., word processing package, software for scanner, etc.)

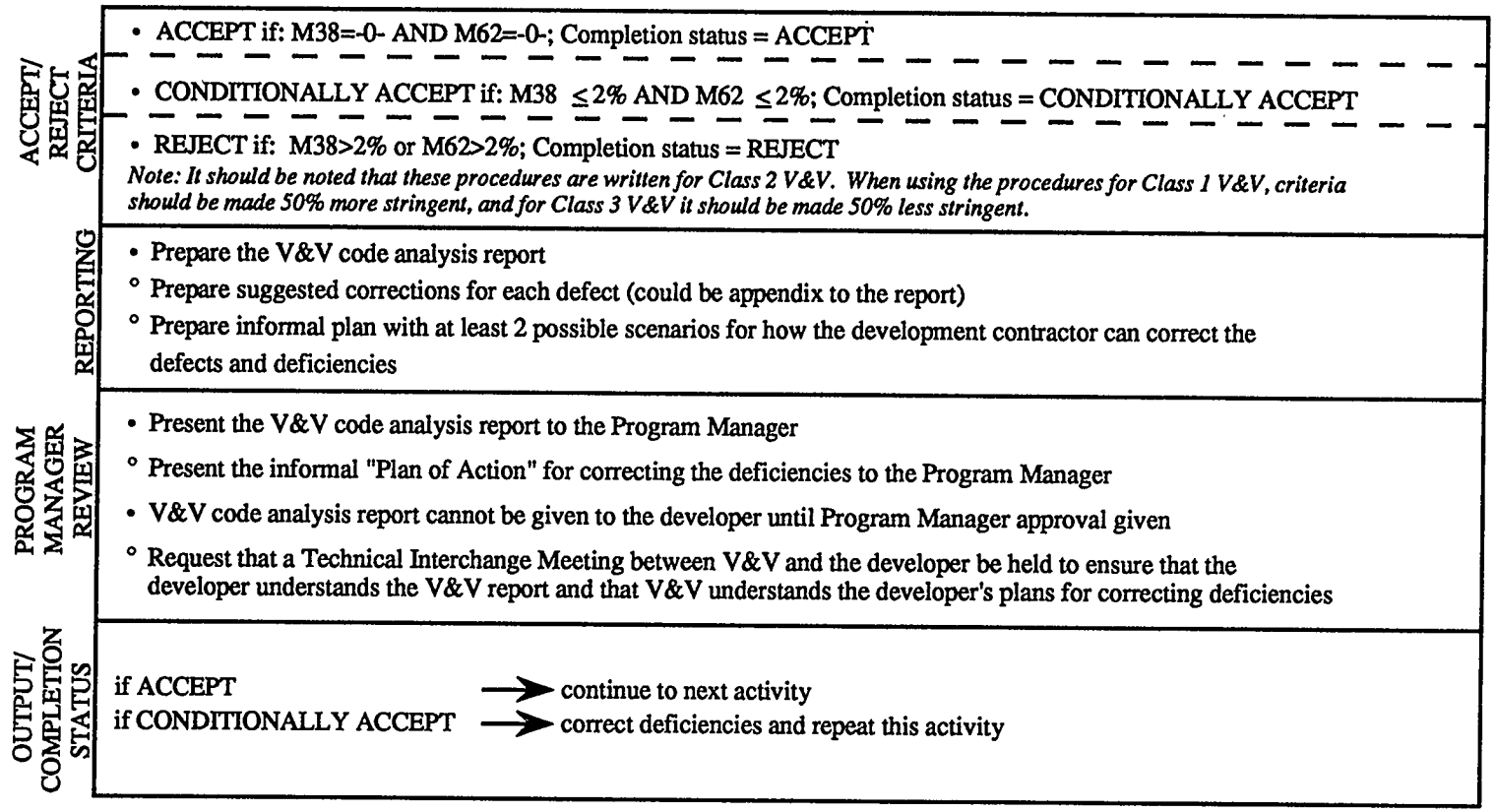
PROGRAM MANAGER APPROVAL

- The informal desk checking plan is presented to the Program Manager
- Approval must be given by Program Manager before work commences

SETUP

- Ensure all human resources have hardcopy (and if possible softcopy) of the source code, a computer, and word processing software


EXECUTION
M1-10
Verify traceability of code to external specifications (see 1.0 Reqts. Tracing) (2.1) 1.0
M11,38
Manually "execute" code with appropriate functional test cases 2.2
R10
M11,38
Look for obvious defects (e.g., misspelled tokens) 2.3
M11,38
Check for correct procedure interfaces 2.4
M11-36, 38,47-60
Look for defect types 2.5
R10
M11,38
Read comments and compare to external specifications 2.6
M36
Compare comments to Design Documentation 2.7
M61-62
Check for compliance with programming standards 2.8
M11,38
Check for checklist items while reviewing code 2.9
INTERRUPT - Hold informal review with program manager who will decide next course of action; completion status = REJECT
Prepare Code Analysis Report 2.10
Repair, Test, Process Improvement Methods X.X
ANALYSIS
• The number and % of defects will be determined (% of modules with defects, or % lines of code with defects)
• The number and % of deviations from programming conventions will be determined (% modules with deviations)
° A suggested correction for each defect/programming convention deviation will be prepared
° Several contingency plans will be prepared describing how the developer can bring the outstanding issues to closure
M11,38
M61-62

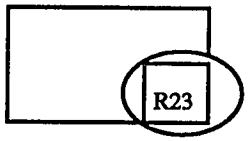


Terminal, lowest level activity, is not discussed on subsequent pages

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

2.2 MANUALLY "EXECUTE" CODE

GOAL OF THIS ACTIVITY:

GOAL: To manually simulate the computer executing the source code to uncover defects.

Pre-Conditions/Trigger Conditions

- Resources are available to perform the activity
- Source code has been delivered
- Schedule dictates that activity commence

PLANNING

- Select module or procedure entry conditions (using a method similar to that described in for code "execution" R10
- Ensure that necessary resources are available (computer, printer, etc.)

PROGRAM MANAGER APPROVAL

N/A for this activity

SETUP

- Obtain any sample data/files necessary to manually "execute" the code
- Build a table of external parameters, local variables, system tables, files, etc., to facilitate recording the results of code "execution"

EXECUTION

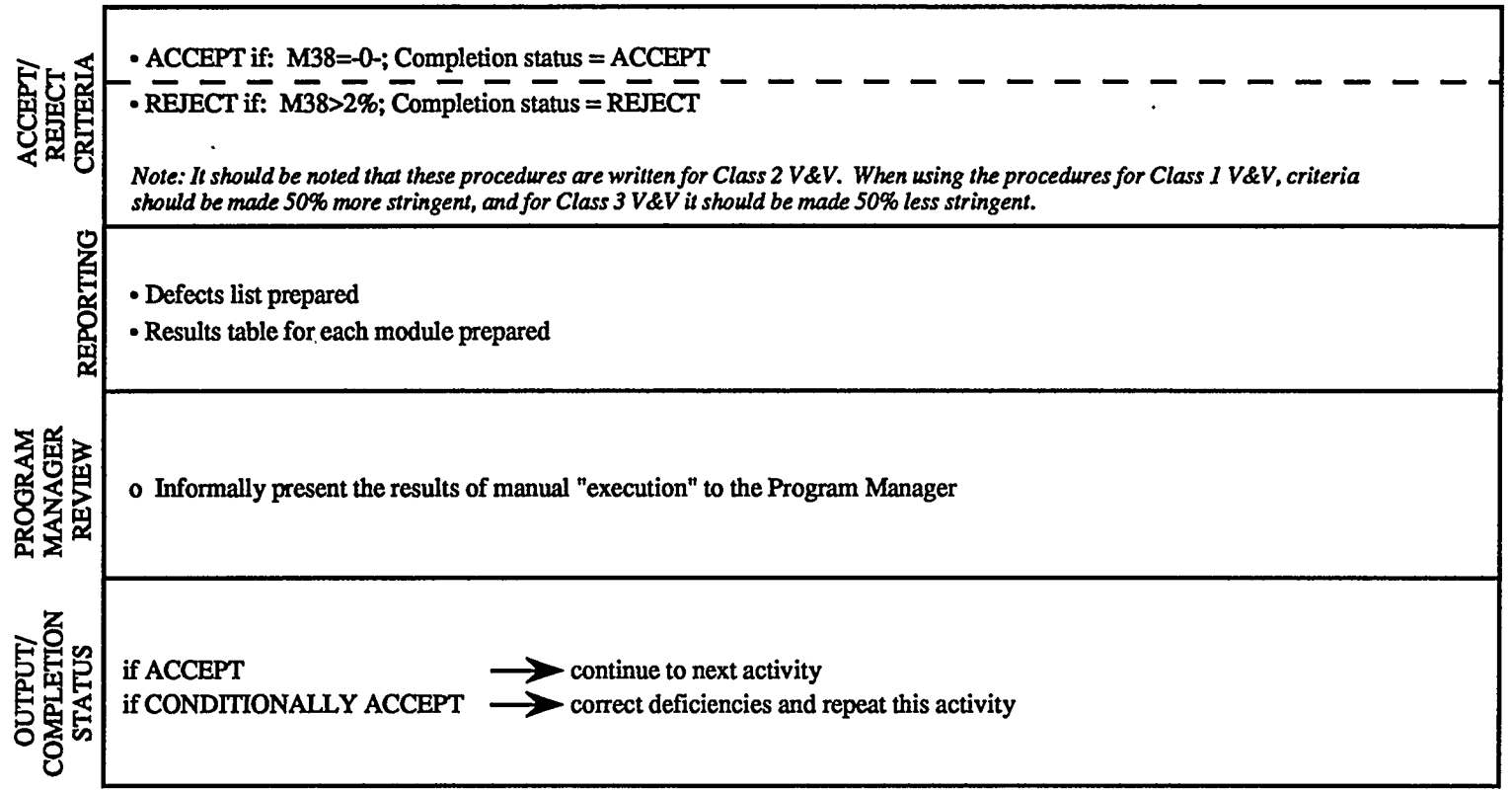
- Using the agreed upon module or procedure entry conditions, sample data/files, and the results table, manually "execute" the code:
  - 1) Starting with the first instruction, write down the starting state of all variables in the results table
  - 2) Manually "execute" the instruction and record the resultant data in the results table
  - 3) Note any errors/defects on a separate sheet of paper. This will be called the defects list R11
  - 4) Repeat steps 1) through 3) until all instructions have been "executed" and exit conditions are satisfied
  - 5) Compare the status of output variables and global data with the status implied by the external specifications

  [if there are an unreasonable number of defects found, INTERRUPT]

ANALYSIS

- The number of defects will be determined 

- The % of defects (several options are available) will be determined 

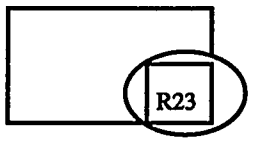


Terminal, lowest level activity, is not discussed on subsequent pages

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

2.5 LOOK FOR SUBSET OF DEFECT TYPES

GOAL OF THIS ACTIVITY:

GOAL: To attempt to locate defects in the source code and/or knowledge by focusing on defect types.

Pre-Conditions/Trigger Conditions

- Resources are available to perform the activity
- Source code has been delivered
- Schedule dictates that activity commence

PLANNING

- Select subset of defect types to be searched for (e.g., see list of defects in Volume 5 and the list of which methods detect the defect types best) (R12)
- Ensure that necessary resources are available (computer, printer, etc.)

PROGRAM MANAGER APPROVAL

o Informally present selected defect types to Program Manager for approval prior to proceeding

SETUP

N/A for this activity

EXECUTION

- Using the source code, look for one or more of the following types of defects: R12

Logic and Control
- -unreachable code
- -improper nesting of loops and branches
- -inverted predicates
- -incomplete predicates
- -improper sequencing of processes
- -infinite loops
- -instruction modification
- -failure to save or restore registers
- -unauthorized recursion
- -missing labels or code
- -unreferenced labels

Data Operations and Computations
- -missing validity tests
- -incorrect access of array components
- -mismatched parameter lists
- -initialization faults
- -anachronistic data
- -undefined variable
- -misuse of variables, both locally and globally
- -data fields unconstrained by natural or defined data boundaries
- -inefficient data transport

All Others
- -calls to subprograms that do not exist
- -improper program linkages
- -input-output faults
- -prodigal programming
- -failure to implement the design

Knowledge/Syntax R15
- -keyword incorrect
- -variable incorrect
- -comparison incorrect
- -reference incorrect
- -reference exceeds allowable range
- -contradiction in IF clauses
- -contradiction in THEN actions
- -apparent inconsistency between IF clauses and THEN action
- -"endless loop" cycle between IF and THEN parts of rule
- -cycle inference chain
- -conflicting rules
- -subsumed rule pair
- -unfirable rule
- -error in THEN clause

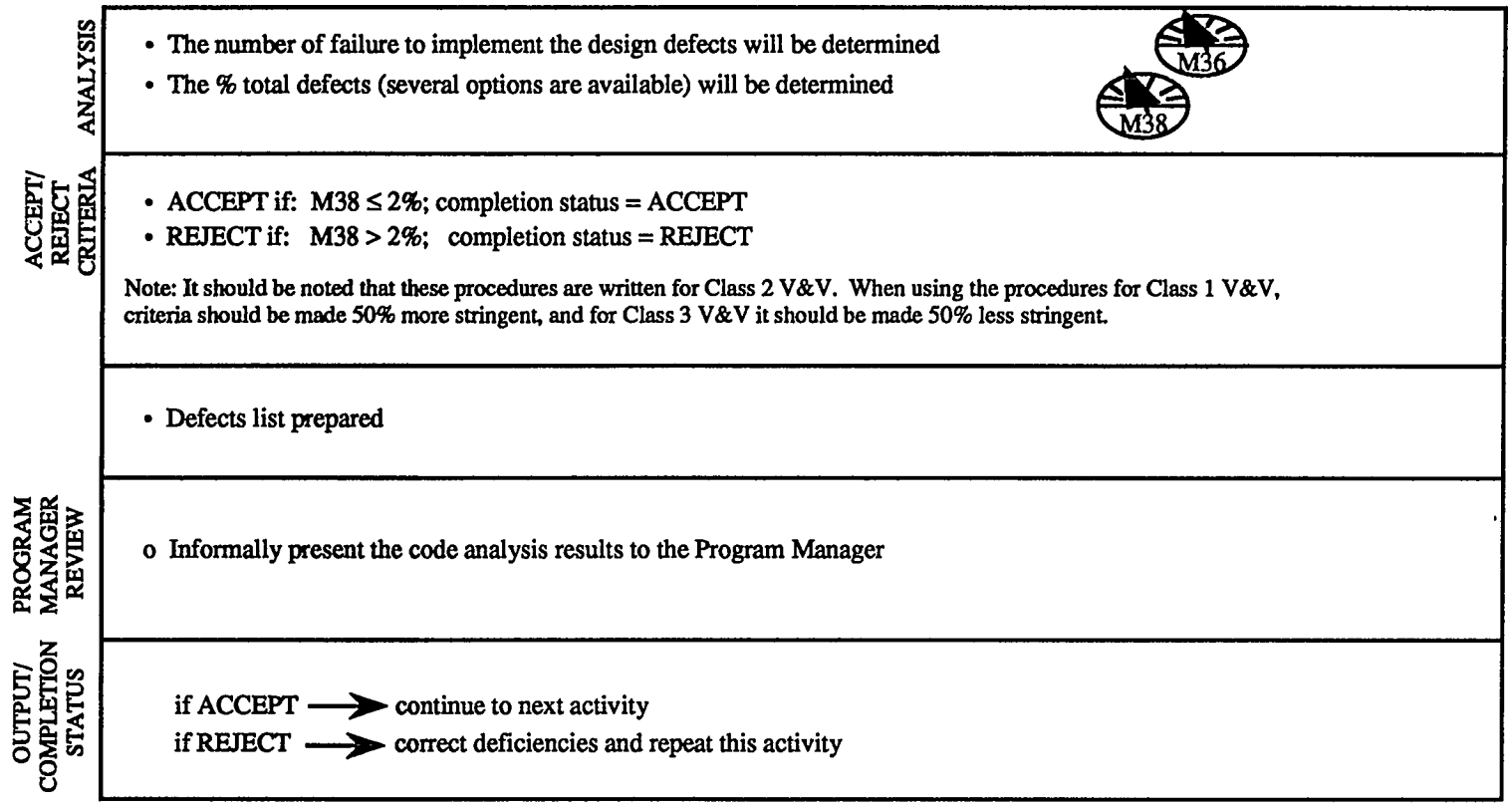
- Note any errors/defects on a separate sheet of paper. This will be called the defects list. R11
  [if there are an unreasonable number of defects found, INTERRUPT]

ANALYSIS

M11 - # of modules with defects
M12 - # of unreachable code defects
M13 - # of improper nesting of loops and branches
M14 - # of inverted predicates defects
M15 - # of incomplete predicates defects
M16 - # of improper sequencing of processes defects
M17 - # of infinite loops defects
M18 - # of instruction modification defects
M19 - # of failure to save or restore registers defects
M20 - # of unauthorized recursion defects
M21 - # of missing labels or code defects
M22 - # of unreferenced labels defects
M23 - # of missing validity tests defects
M24 - # of incorrect access of array components
M25 - # of mismatched parameter lists
M26 - # of initialization faults
M27 - # of an achronistic data
M28 - # of undefined variable
M29 - # of misuse of variables, both locally and globally
M30 - # of data fields unconstrained by natural or defined data boundaries
M31 - # of inefficient data transport
M32 - # of calls to subprograms that do not exist
M33 - # of improper program linkages
M34 - # of input-output faults
M35 - # of prodigal programming
M36 - # of failure to implement the design
M38 - # % of modules with defects.
M47 - # of keyword incorrect defects
M48 - # of variable incorrect defects
M49 - # of comparison incorrect defects
M50 - # of reference incorrect defects
M51 - # of reference exceeds allowable range defects
M52 - # of contradiction IF clauses defects
M53 - # of contradiction THEN clauses defects
M54 - # of apparent inconsistency between IF clauses and THEN action defects
M55 - # of "endless loop" cycle between IF and THEN parts of rule defects
M56 - # of cyclic inference chain defects
M57 - # of conflicting rules defects
M58 - # of subsumed rule pair defects
M59 - # of unfirable rule defects
M60 - # of error in THEN clause defects
M61 - # of deviations form programming conventions modules with
M62 - % of modules with deviations from programming conventions

NOTE: M# characters are represented by a 

throughout all Guideline/Procedures

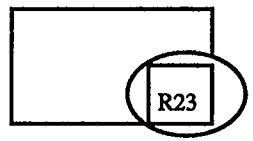


— Terminal, lowest level activity, is not discussed on subsequent pages

— 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

2.9 CHECKLIST CODE REVIEW

GOAL OF THIS ACTIVITY:

GOAL: To attempt to locate defects in the source code using a code checklist.

Pre-Conditions/Trigger Conditions

- Resources are available to perform the activity
- Source code has been delivered
- Schedule dictates that activity commence

PLANNING

- Tailor code checklist to delete items that are not applicable to the development effort and add others that are not on the checklist but should be considered for the development effort
- Ensure that necessary resources are available (computer, printer, etc.)

PROGRAM MANAGER APPROVAL

o Informally present tailored checklist to Program Manager for approval prior to proceeding

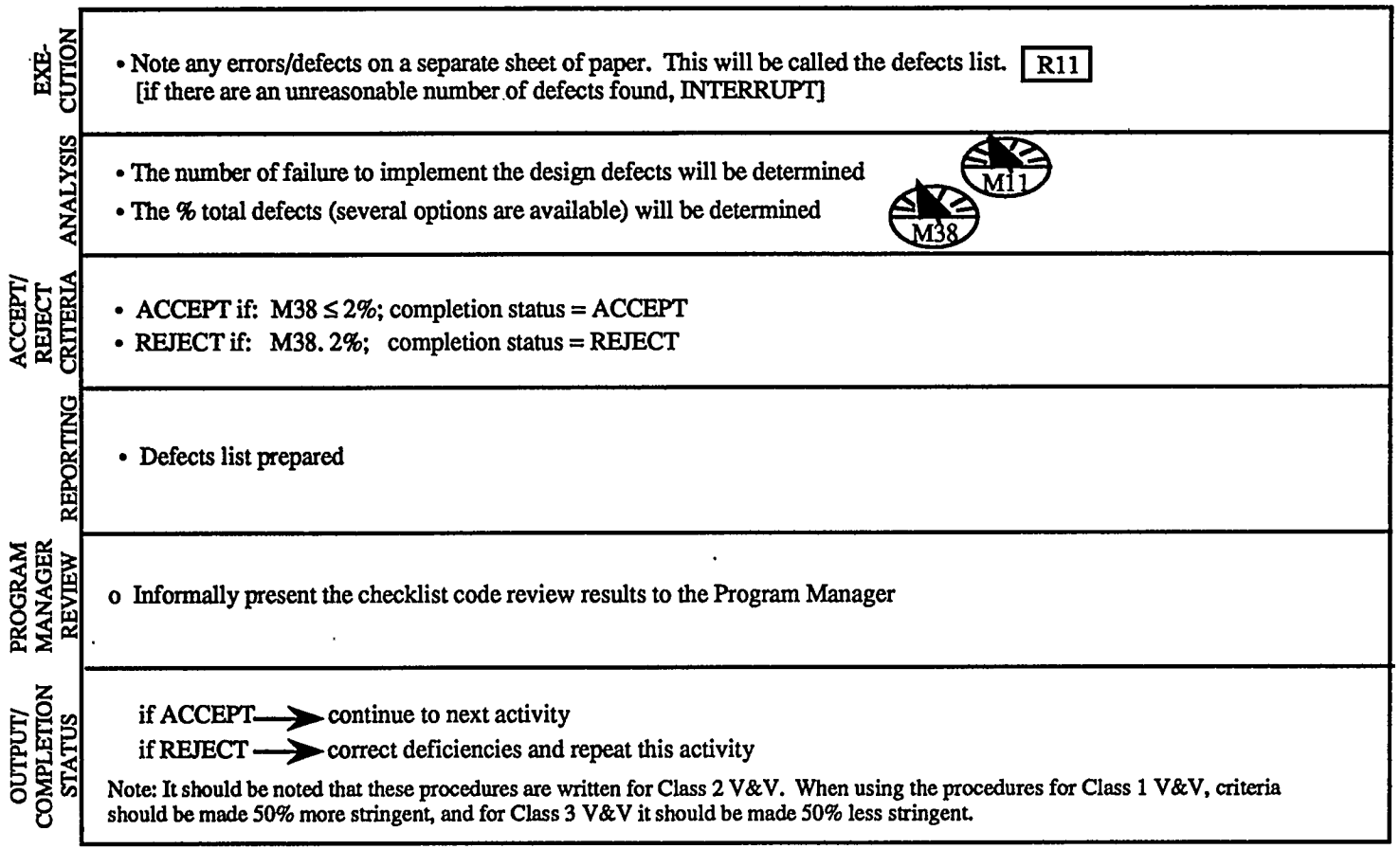
SETUP

N/A for this activity

EXECUTION

- Using the source code, look for the following items: R13

Is the compilation (or assembly) listing free of fault messages?
Have the deficiencies noted in the standards audit been corrected?
Do data definitions exploit typing capabilities to advantage?
Are mixed-mode expressions in violation of project standards?
Do all pointers and indexes adhere to binding conventions?
Except for appropriate delimiters (for example, those in *for* N *in* 1..10), are constants expressed as parameters?
For assembly language programs, have registers been saved on entry and restored on exit? Have stacks been properly initialized?
For assembly language programs, have project standards for register use been observed?
For branch points that correspond to the detailed design documentation, are the conditions sufficient? (That is, if *n* conditions are required for a decision, are all *n* present and properly accounted for?)
Are all conditions expressed in the correct sense (for example, A>B versus B>A)?
Do control constructs conform to structured programming standards?
Are standards that restrict the depth to which loops and branches may be nested adhered to? Are loops and branches properly nested?
Are indexes or subscripts properly initialized?
Are loop termination conditions invariably achievable?
Are any branch conditions mutually exclusive (such that they can lead to unreachable code)?
For assembly language programs, are the expected contents of memory used in lieu of their addresses?
Do processes occur in the correct sequence?
Where applicable, are divisors tested for zero or noise?
Where applicable, are indexes, pointers, and subscripts tested against array, record, or file bounds?
Are imported data tested for validity? Are they ever reassigned except when they were transmitted for the express purpose of updating?
Do actual and formal interface parameter lists match?
Do data declarations observe data boundaries implied by machine architecture, language, or user-define declarations?
Are all variables used? Are all output variables assigned?
Can any statements that are enclosed within loops be placed outside the loops without computational effect?
Is a more efficient mechanism for an input or output operation possible?
Are the correct data being operated on at each statement?
Are any labels unreferenced (a warning that something else may be amiss)?
Are equations properly formed (e.g., if both A and B are to be divided, (A+B)/C and not A+B/C]?
Can a connection to an external device result in an interminable wait?
If there are requirements on execution time, will they be met?
Does the code fit within its allocated storage? Local data?
Have all the elements of the design been implemented as they were specified?
Has each function of the external specification for this code been correctly complied with?



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

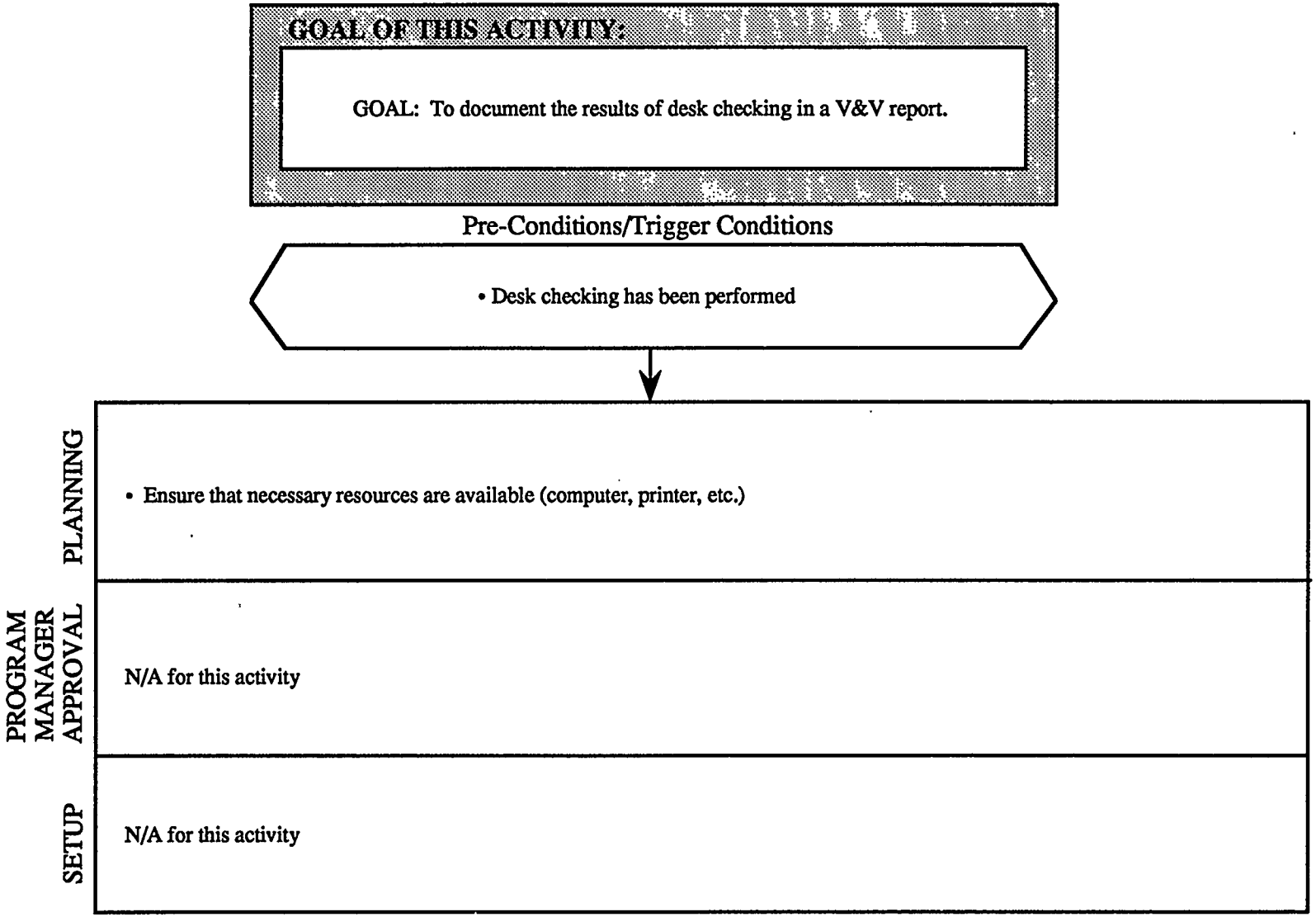
High level (non-terminal) activity (decomposes to lower level activities), discussed on subsequent pages

o -- optional steps
• -- obligatory

M2 — 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

R23 -- see reference 23 in endnotes

2.10 PREPARE CODE ANALYSIS REPORT



EXECUTION

- Using the defects list generated in the previous steps, generate a Code Analysis Report:
  - 1) Prepare table describing the results of desk checking, e.g.:

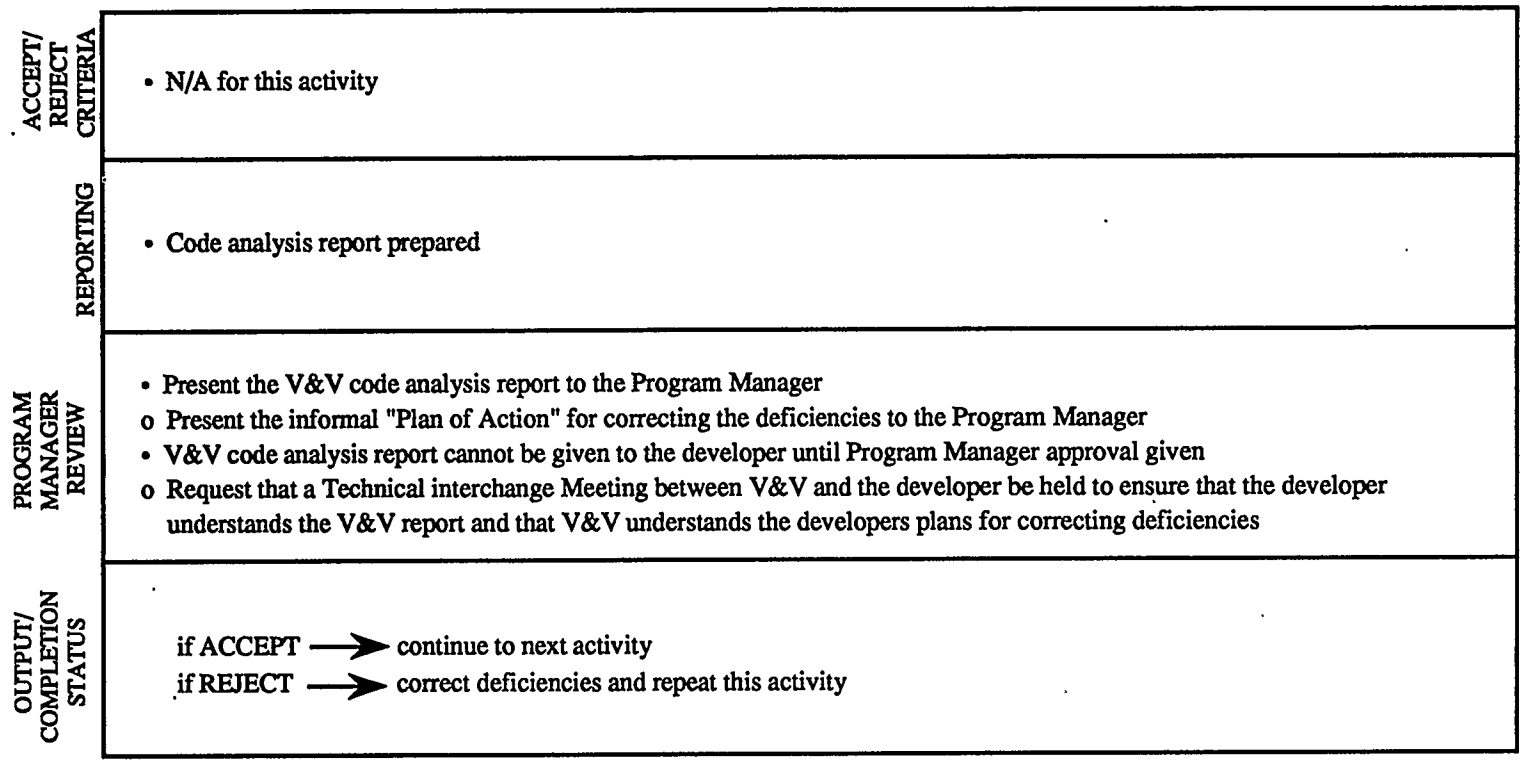
Module Name	# (%) Unreachable Code Defects	# (%) Improper Nesting of Loops Defects	• • •	Total # (%) Module Defects	Cum. Total
AAAA	-0-	5		15	15
•					•
•					•
•					•
ZZZZ	1	-0-		3	295

  - 2) Using the table, prepare the report, possibly following this outline:

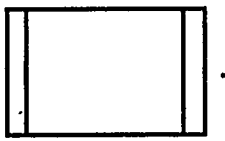
Executive Summary
1.0 INTRODUCTION
2.0 SCOPE - Describe the portion of the system analyzed (perhaps just one Computer Software Configuration Item)
3.0 SUMMARY OF RESULTS
  3.1 First Desk Checking Activity (from 2.1-2.9)
  3.2 Second Desk Checking Activity (from 2.1-2.9)
  3.X
    Table (as above in Step 1)
    Reference the appendix
4.0 CONCLUSIONS AND RECOMMENDATIONS
Appendix
  Defects List

ANALYSIS

N/A for this activity



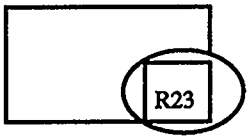
Terminal, lowest level activity, is not discussed on subsequent pages



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

M1 - # of critical code elements
M2 - Percentage of critical code elements
M3 - Total number of code elements
M4 - # of unintended functions
M5 - % of unintended functions
M6 - # of unsatisfied (omitted) requirements
M7 - # of partially satisfied requirements
M8 - % of unsatisfied (omitted) requirements
M9 - % of partially satisfied requirements
M10 - % of "TBD" code elements
M11 - # of modules with defects
M12 - # of unreachable code defects
M13 - # of improper nesting of loops and branches
M14 - # of inverted predicates defects
M15 - # of incomplete predicates defects
M16 - # of improper sequencing of processes defects
M17 - # of infinite loops defects
M18 - # of instruction modification defects
M19 - # of failure to save or restore registers defects
M20 - # of unauthorized recursion defects
M21 - # of missing labels or code defects
M22 - # of unreferenced labels defects
M23 - # of missing validity tests defects
M24 - # of incorrect access of array components
M25 - # of mismatched parameter lists
M26 - # of initialization faults
M27 - # of an achronistic data
M28 - # of undefined variable
M29 - # of misuse of variables, both locally and globally
M30 - # of data fields unconstrained by natural or defined data boundaries
M31 - # of inefficient data transport
M32 - # of calls to subprograms that do not exist
M33 - # of improper program linkages
M34 - # of input-output faults
M35 - # of prodigal programming
M36 - # of failure to implement the design
M38 - # % of modules with defects.
M47 - # of keyword incorrect defects
M48 - # of variable incorrect defects
M49 - # of comparison incorrect defects
M50 - # of reference incorrect defects
M51 - # of reference exceeds allowable range defects
M52 - # of contradiction IF clauses defects
M53 - # of contradiction THEN clauses defects
M54 - # of apparent inconsistency between IF clauses and THEN action defects
M55 - # of "endless loop" cycle between IF and THEN parts of rule defects
M56 - # of cyclic inference chain defects
M57 - # of conflicting rules defects
M58 - # of subsumed rule pair defects
M59 - # of unfixable rule defects
M60 - # of error in THEN clause defects
M61 - # of deviations form programming conventions modules with
M62 - % of modules with deviations from programming conventions $\left(\frac{M61}{\#\text{ modules}} \times 100 \right)$

REFERENCES

R10 - Bryan, W.L. and S.G. Siegal, *Software Product Assurance: Techniques from Reducing Software Risk*, Elsevier, New York, New York, 1988.

R11 - Charette, R.M., *IV&V and Risk Management*, Presentation made to the ANSI/IEEE Standard 1012 Working Group at the National Institute of Standards and Technology, November 12, 1992.

R12 - Chisholm, G.H., B.T. Smith, and A.S. Wojcik, *Formal System Specifications -- A Case Study of Three Diverse Representations* , Argonne National Laboratory Report, ANL-90/43, December, 1990.

R13 - Culbert, C., G. Riley, and R.T. Savely, *Approaches to the Verification of Rule-Based Expert Systems* , Proceedings of First Annual Workshop on Space Operations Automation and Robotics Conference (SOAR '87), Houston, Texas, 1987.

R15 - *DoD (Department of Defense) DOD-STD-2167A Military Standard: Defense System Software Development*, Department of Defense, 29 February 1988, Washington, D.C. 20301.

GUIDELINE PROCEDURE P3.0

P3.0 FUNCTIONAL TESTING (Black Box)

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform functional (black box) 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. Unit testing, as a minimum, has been performed.
Note: Testing will be performed the same even for revised software components such as GVIs or expert system shells.

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Source code has been delivered (or executable)
- Static analysis has been performed
- Unit testing, as a minimum, 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) and how output will be analyzed to determine correctness

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) Using requirements specification, identify functions (per level determined during Planning Step) and partition them into classes.
  - Test cases belong to the same functional class if:
    - -the same input variables are relevant to each
    - -the same type of transformations will result
    - -the same output variables are affected by each R10
- 2) Have a third party (preferably an independent one) review the functional classes and modify/update them as appropriate
- 3) Partition classes into subclasses using the definition from Step 1. For example:

  Class: Status Inquires of File XYZ
  Subclass 1: Valid Inquiries
  Subclass 2: Invalid Inquiries
- 4) Continue this process iteratively until the lowest level is reached
- 5) Select one test case for each subfunction at the lowest hierarchical level of partitioning
- 6) Have a third party (preferably independent) review the subclass hierarchy and selected test cases, and update/modify them as appropriate.
- 7) 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 a DOD-STD-2167A) with priority 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 

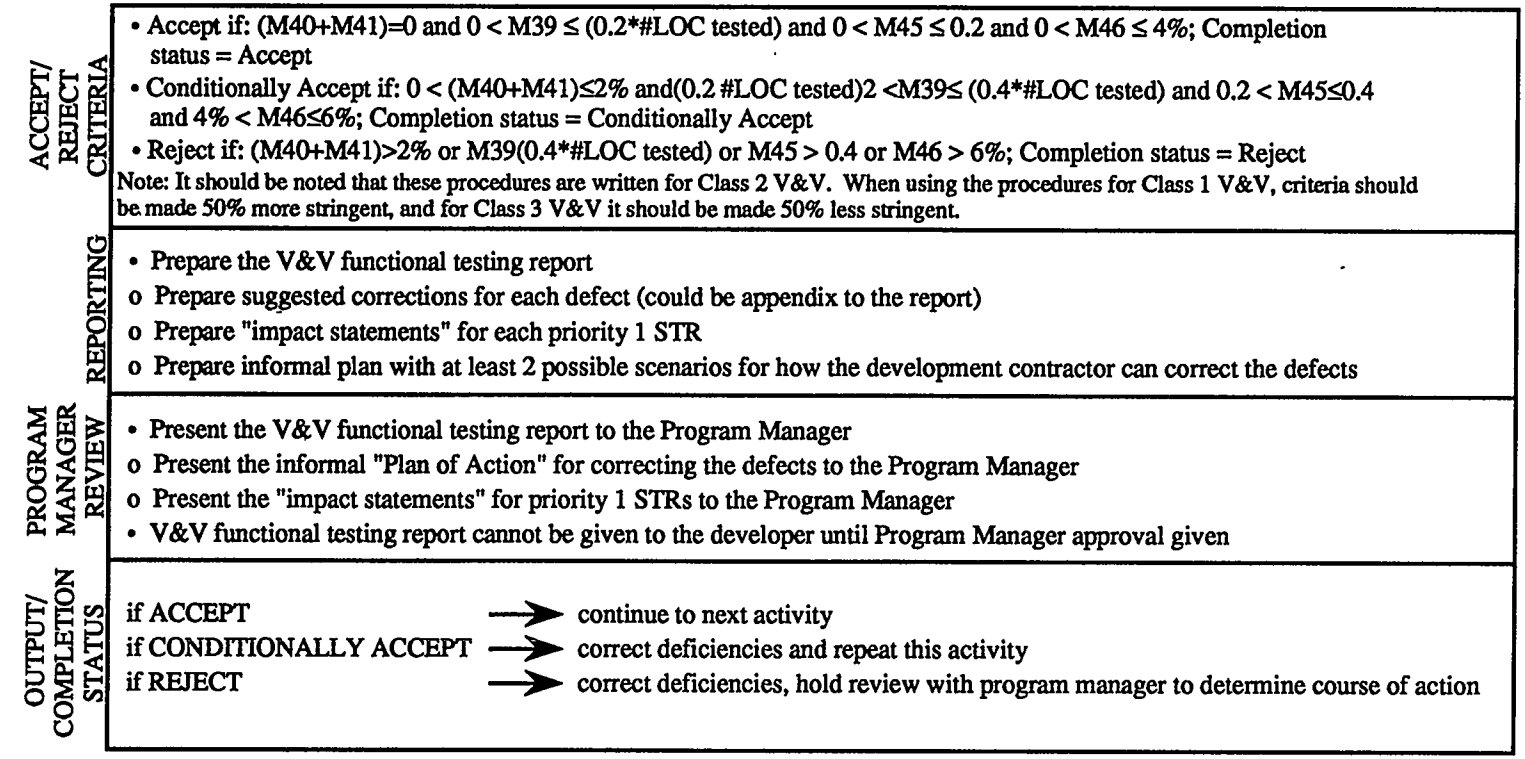
- The number of defects per line of code (LOC) will be determined $\left(\frac{\text{M39}}{\text{Total \# LOC tested}}\right)$ 

- 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



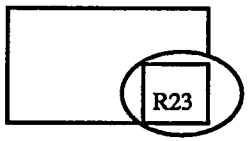
Terminal, lowest level activity, is not discussed on subsequent pages

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

M37 - Number of functional test cases defects
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 of defects/LOC
M46 - Percentage test cases failed defects (#test cases failed *100)
M37

R10 - Bryan, W.L. and S.G. Siegal. Software Product Assurance: Techniques for Reducing Software Risk. New York: Elsevier, 1988.

GUIDELINE PROCEDURE P4.0

P4.0 BOUNDARY\DOMAIN TESTING

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform boundary/domain 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. Unit testing has possibly been conducted (and also possibly integration testing.)

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Source code has been delivered (or executable)
- Static analysis has been performed
- Unit testing, as a minimum, 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 an independent team will perform the testing and how output will be analyzed to determine correctness

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, design documentation, source code/executable, etc.

o 1) Perform testing using the finite domain strategy [R14] *

- a) Identify closed and open borders (domain boundaries per [R16] using the ollowing definitions:
  Border: a single simple predicate in the boundary of a path
  Closed Border: formed by the relational $\leq \geq =$
  Open Border: formed by the relationals $<, >, \wedge, =$
- b) Select test points on or near the closed borders to detect border shifts (closure verification per [R16]) or to provide limits on the maximum displaced domain, using the following definitions:
  Border Shift (or Domain Fault): Results in a displaced domain.
  Displaced Domain: A set of elements placed in an incorrect path domain.
- c) 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.

o 2) Perform testing using the functional boundary method [R10]

- a) Using the functional partitions from functional testing, determine the specified input range (input domain)
- b) Select test points at the extremes, just beyond the extremes, inside the interval, and at 0 (if applicable.) For example, a procedure expecting input in the range of -10 to +10 could have these test points:
  {-11, -10, -4, 0, 4, 10, 12}
- c) Determine the specified output range (output domain), using the functional partitions (note that the extremes of the specified output may not always have an obvious correlation to the range of the input)
- d) Select test points as in step b, but for output domain
- o e) Develop test cases by examining the performance requirements for the system. For example, the time to respond to a status inquiry would be tested given 25 active users, 50 users, 75, and so on.
- o f) Develop test cases by adjusting the volume of data input/output. For example, the average time to locate 50 records in a database of 50 records, in a database of 200 records, 2,000 records, etc.
- g) Perform each test case and record the results in the 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.

ANALYSIS

- The number of functional test cases will be determined 

- 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 $\left(\frac{\text{\# cases failed}}{\text{M37}} * 100\right)$ 

o A suggested correction for each defect will be prepared
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
* *This activity is very complex and requires further decomposition. The reader should refer to the referenced material before undertaking this activity.*

ACCEPT/ REJECT CRITERIA

- Accept if: M38≥90% and (M40+M41)=0 and 0 < M39 ≤ (0.2*#LOC tested) and 0 < M45 ≤ 0.2 and 0 < M46 ≤ 4%; Completion status = Accept
- Conditionally Accept if: 0 < (M40+M41)≤2% and(0.2 #LOC tested)2 <M39≤ (0.4*#LOC tested) and 0.2 < M45≤0.4 and 4% < M46≤6%; Completion status = Conditionally Accept
- Reject if: (M40+M41)>2% or M39(0.4*#LOC tested) or M45 > 0.4 or M46 > 6%; Completion status = Reject

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare the V&V boundary/domain testing report
- o Prepare suggested corrections for each defect (could be appendix to the report)
- o Prepare "impact statements" for each priority 1 STR
- o Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the defects

PROGRAM MANAGER REVIEW

- Present the V&V functional testing report to the Program Manager
- o Present the informal "Plan of Action" for correcting the defects to the Program Manager
- o Present the "impact statements" for priority 1 STRs to the Program Manager
- V&V functional testing report cannot be given to the developer until Program Manager approval given

OUTPUT/ COMPLETION STATUS

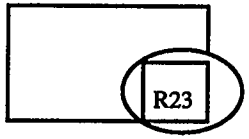
if ACCEPT → continue to next activity
if CONDITIONALLY ACCEPT → correct deficiencies and repeat this activity
if REJECT → correct deficiencies, hold review with program manager to determine course of action

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

M37 - Number of functional test cases defects
M38 - % of domains (or boundaries) tested
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 of defects/LOC
M46 - Percentage test cases failed defects (#test cases failed *100)
M37

R10 - Bryan, W.L. and S.G. Siegal. Software Product Assurance: Techniques for Reducing Software Risk. New York: Elsevier, 1988.

R14 - Davis, A.M. Software Requirements: Analysis and Specification. Englewood Cliffs, N.J.: Prentice Hall, 1990.

R16 - Deutsch, M. Software Verification and Validation: Realist Project Approaches. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

GUIDELINE PROCEDURE P5.0

P5.0 PROCESS-ORIENTED AUDITS

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform an audit of the development process. Requirements specification has been developed. Developer has prepared Software Development Plan. Development has commenced and can be at any phase (requirements, design, etc.)

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- Software development has been delivered
- Resources are available to perform the activity
- Development has commenced
- Schedule dictates that activity commence

PLANNING

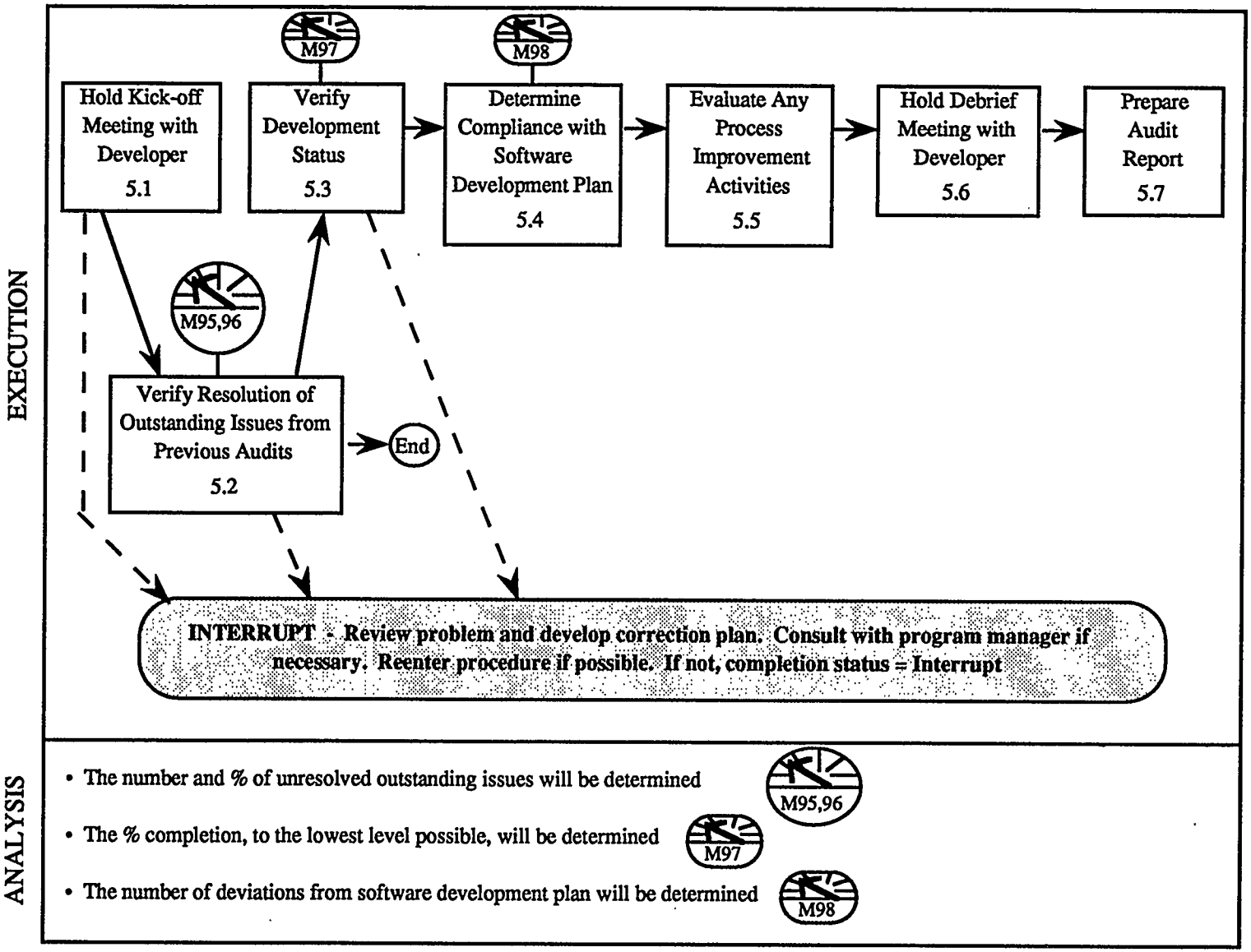
- Determine the high level tasks for this activity (see Execution below)
- Establish a schedule for the activity
- Assign human resources to the high level tasks (1-2 V&V members with software engineering background)
- Ensure that human resources agree that schedule is reasonable
- Ensure that other necessary resources are available (computer, printer, etc).
- Prepare informal process-orientated audit plan showing tasks, schedules, resources, etc.
- Select tools to be used(e.g., database management system, spreadsheet, etc.)

PROGRAM MANAGER APPROVAL

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

SETUP

- Obtain latest status report from developer (if available). This should be detailed enough to show status for at least the CSCI level, and preferably top or low level CSCS (if designing/coding).
- Obtain results from last audit performed (if applicable)
- Prepare audit check list
- Prepare working sheets for status reporting (from spreadsheet or DBMS)


EXECUTION
M97
M98
Hold Kick-off Meeting with Developer 5.1
Verify Development Status 5.3
Determine Compliance with Software Development Plan 5.4
Evaluate Any Process Improvement Activities 5.5
Hold Debrief Meeting with Developer 5.6
Prepare Audit Report 5.7
M95,96
Verify Resolution of Outstanding Issues from Previous Audits 5.2
End
INTERRUPT - Review problem and develop correction plan. Consult with program manager if necessary. Reenter procedure if possible. If not, completion status = Interrupt
ANALYSIS
• The number and % of unresolved outstanding issues will be determined
M95,96
• The % completion, to the lowest level possible, will be determined
M97
• The number of deviations from software development plan will be determined
M98

ACCEPT/ REJECT CRITERIA

- ACCEPT if: M96=0 and developer's % completion: 0 < M97 ≤ 5% and , 0 < M98 ≤ 20; Completion status = ACCEPT
- CONDITIONALLY ACCEPT if: 0 < M96 ≤ 2% AND developer's % completion: 5% < M97 ≤ 8% AND 20 < M98 ≤ 30; Completion status = CONDITIONALLY ACCEPT
- REJECT if: M96 > 2% or Developer's % completion: M97 > 8% or M98 > 30; Completion status = REJECT

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare the V&V boundary/domain testing report
- o Prepare "impact statements" for all unresolved issues, status discrepancies,a nd SDP deviations (could be appendix to this report)
- o Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the issues

PROGRAM MANAGER REVIEW

- Present the V&V functional testing report to the Program Manager
- o Present the informal "Plan of Action" for correcting the defects to the Program Manager
- V&V audit report cannot be given to the developer until Program Manager approval given
- o Request that a Technical Interchange Meeting between V&V and the developer be held to ensure that the developer understands the V&V report and that V&V understands the developer's plans for correcting deficiencies

OUTPUT/ COMPLETION STATUS

if ACCEPT	→ continue to next activity
if CONDITIONALLY ACCEPT	→ correct deficiencies and repeat this activity
if REJECT	→ correct deficiencies, hold review with program manager to determine course of action, correct deficiencies
if INTERRUPT	→ develop correction plan and reenter procedure if possible

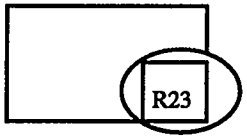
Terminal, lowest level activity, is not discussed on subsequent pages

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

M95 - Number of unresolved outstanding issues

M96 - Percentage of outstanding issues not resolved

$$\left[\frac{M95}{\text{Total \# issues}} * 100 \right]$$

M97 - Percentage completion for each unit/CSC/CSCI

M98 - Number of deviations from Software Development Plan

GUIDELINE PROCEDURE P6.0

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 Manage 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. R16
- 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. R19
- 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 h 9
- The total number of Software Trouble Reports Defects M39
- The number of Software Trouble Reports Defects (by priority (1-5)) will be determined M40-44
- The number of defects per line of code (LOC) will be determined $\left(\frac{M39}{\text{Total \# LOC tested}}\right)$ M45
- The % of test cases failed is determined M46
- o A suggested correction for each defect will be prepared $\left(\frac{\text{\# cases failed}}{M37} * 100\right)$
- 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.*

ACCEPT/ REJECT CRITERIA

- Accept if: (M40+M41)=0 and M39 ≤ (0.2*#LOC tested) and M45 ≤ 0.2 and M46 ≤ 4%; Completion status = Accept
- Conditionally Accept if: 0< (M40+M4)1≤2% and (0.2* #LOC tested) < M39≤ (0.4*#Loc tested) and 0.2 < M45≤0.4 and 0.4 < M46≤6%; Completion status = Conditionally Accept
- Reject if: (M40+M41)>2% or M39>(0.4*#LOC tested) or M45 > 0.4 or M46 > 4%; Completion status = Reject

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare the V&V random testing report
- o Prepare suggested corrections for each defect (could be appendix to the report)
- o Prepare "impact statements" for each priority 1 STR
- o Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the defects

PROGRAM MANAGER REVIEW

- Present the V&V random testing report to the Program Manager
- o Present the informal "Plan of Action" for correcting the defects to the Program Manager
- o Present the "impact statements" for priority 1 STRs to the Program Manager
- V&V functional testing report cannot be given to the developer until Program Manager approval given

OUTPUT/ COMPLETION STATUS

if ACCEPT	→	continue to next activity
if CONDITIONALLY ACCEPT	→	correct deficiencies and repeat this activity
if REJECT	→	correct deficiencies, hold review with program manager to determine course of action

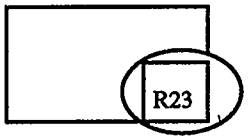
Terminal, lowest level activity, is not discussed on subsequent pages

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 $\left(\frac{\text{\#test cases failed}}{\text{M37}} * 100\right)$
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.

GUIDELINE PROCEDURE P7.0

P7.0 SEMANTIC/META-LEVEL KNOWLEDGE CHECKING

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to use a static testing method to discover problems with the Knowledge Structure which would typically not be caught with the automated syntax-checking packages. These problems require special detailed engineering knowledge about the information in the Knowledge Structure (meta-knowledge) which must be gathered in a specia database. No automated tools are available for most of the problems, but the D-EVA tool does address a few problem, and a full-featured tool METACHECK has been designed and simulated in a behavioral experiment and found to be highly effective. [2]It should be noted that certainty factors, detection of unreachable rules, and detection of whether rules firing in a different order will affect the results are not handled here and would required specialized tool. [1R48A] [2R29]

Pre-Conditions/Trigger Conditions

- Resources are available to develop the meta-level knowledge engineering database (may take several engineer-weeks)
- Resources are available to perform this guideline procedure (may take several days)
- The Knowledge Structure is available for review in a computer text editor
- It is appropriate to begin this guideline procedure, both from a schedule and from an appropriateness point of view

PLANNING

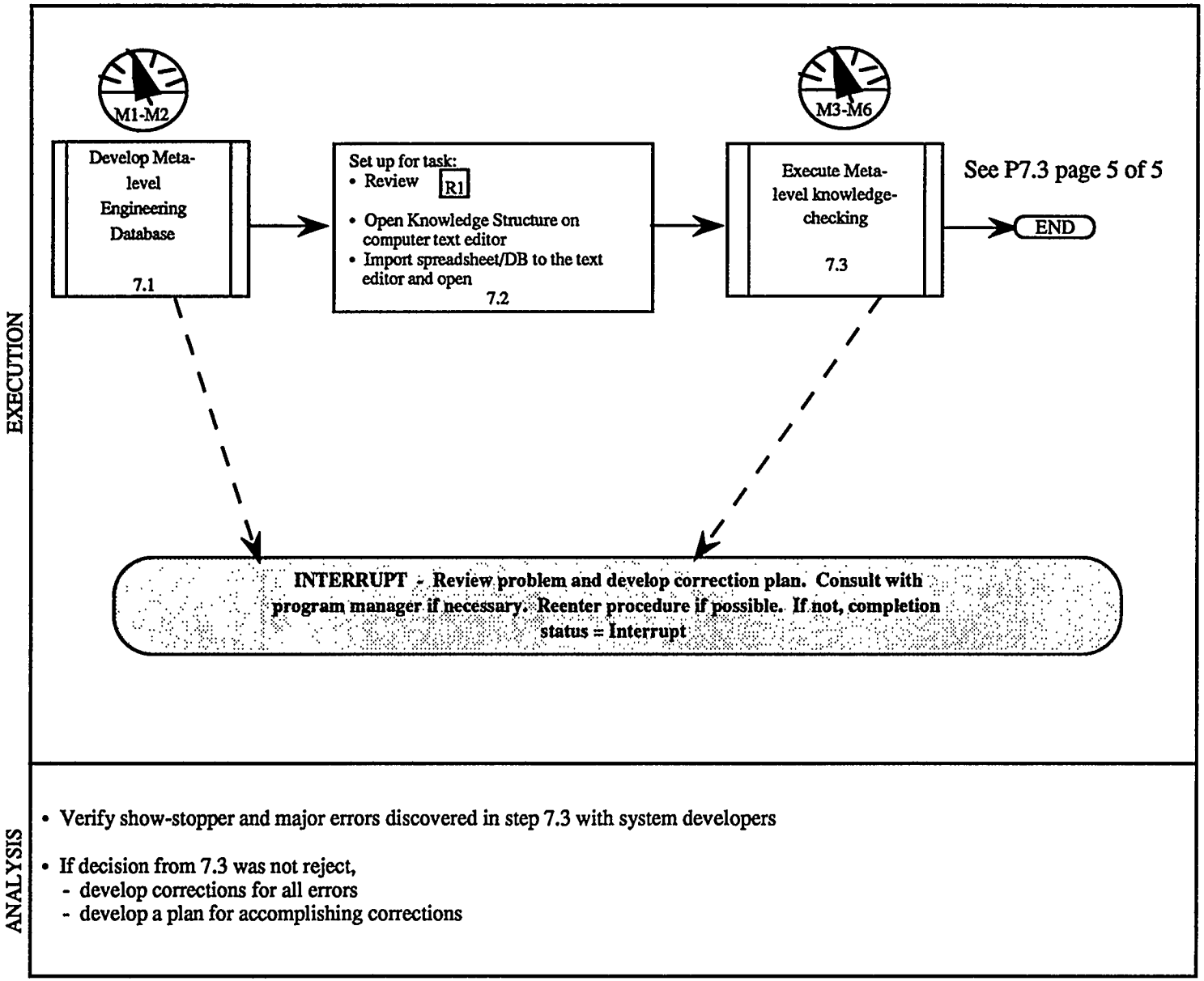
- Identify the specific human and computer resources to do the various tasks (see Execution)
- Ensure that the Knowledge Structure is available in media form or on a computer
- Ensure that the human experts and the technical manuals needed to develop the meta-level engineering database are available (see 7.1) and that a spread sheet or database product is available. Expert system shells may be useful to provide execution tracing and rationale for conclusions.
- Plan to make hard or media copies of the Knowledge Structure and technical manuals as necessary
- Develop a schedule and resource-assignment plan and verify that it is acceptable to all involved

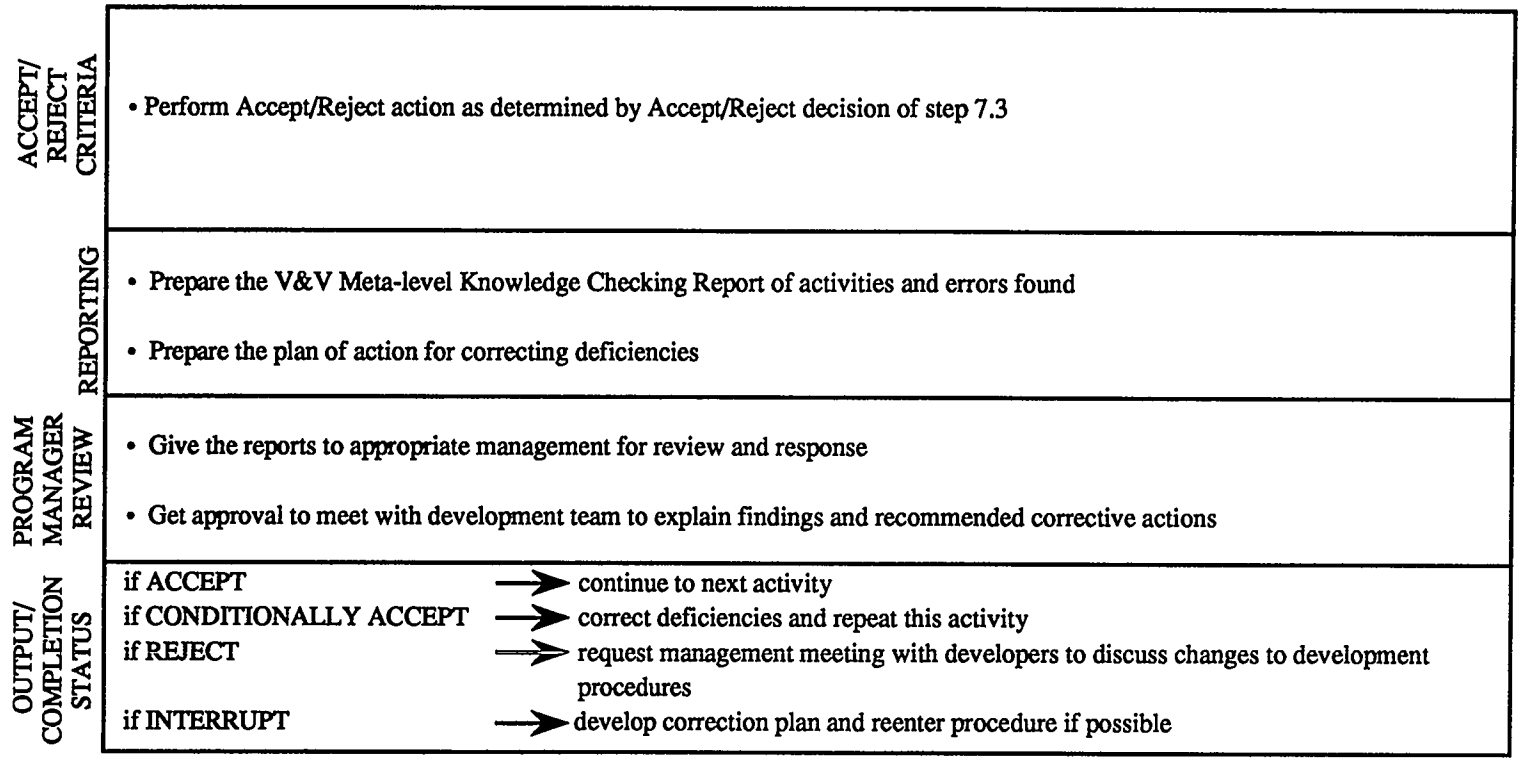
PROGRAM MANAGER APPROVAL

- Present plan to the Program Manager for approval and assistance
- Complete any other approval items that may be needed

SETUP

- Make copies of technical manuals and Knowledge Structure as needed
- Install the knowledge structure, a test editor, and a spread sheet on the correct number of computer terminals
- Arrange that needed number of reference materials for this method are available [R1]


EXECUTION
M1-M2
Develop Meta-level Engineering Database
7.1
Set up for task:
• Review R1
• Open Knowledge Structure on computer text editor
• Import spreadsheet/DB to the text editor and open
7.2
M3-M6
Execute Meta-level knowledge-checking
7.3
See P7.3 page 5 of 5
END
INTERRUPT - Review problem and develop correction plan. Consult with program manager if necessary. Reenter procedure if possible. If not, completion status = Interrupt
ANALYSIS
• Verify show-stopper and major errors discovered in step 7.3 with system developers
• If decision from 7.3 was not reject,
- develop corrections for all errors
- develop a plan for accomplishing corrections

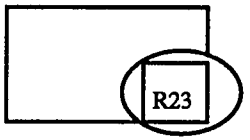


Terminal, lowest level activity, is not discussed on subsequent pages

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

P7.1 DEVELOP META-LEVEL ENGINEERING DATABASE

GOAL OF THIS ACTIVITY:

The goal of this activity is to develop the engineering database which provides the information necessary to detect the semantic and meta-level errors and anomalies in a knowledge structure.

Pre-Conditions/Trigger Conditions

- Resources are available for this task
- The human personnel and technical manual resources necessary to accomplish this task are available
- Schedule dictates this activity begin

PLANNING	• Review description of system to ascertain what physical systems are involved • Examine Knowledge Structure elements in detail to determine type and level of detail of engineering knowledge needed • Determine whether needed knowledge is available in technical manuals or, if not complete, determine which engineers possess the needed knowledge • Study Knowledge Structure to find way of partitioning contents (rules, frames, objects, etc.) into naturally homogeneous and cohesive components • Schedule the knowledge sources for multiple working sessions, estimated 2-15 days, based on partitioning of Knowledge Structure
PROGRAM MANAGER APPROVAL	N/A
SETUP	N/A

EXECUTION

1. Set up a database or spreadsheet with the column fields set those shown below (LOC=unique location identifier for that element, Sys=System, Var=Variable, Dim=Dimensional unit, Ref=Reference, Val=Value, Comp=Comparison value, Op=operations, Assoc=Associated Functions). An example is given for a rule No. 20: "If Hi-Steam < 20psi then open value 1:

LOC	Sys Var	Dim	Ref Var	Ref Val	Dim	Comp	Assoc	Action	Action Val
20.	Hi-Steam	?	- -	20	psi	<	--		
Open	Valve 1								

In this example the dimensional unit of the system variable "Hi-Steam" is not mentioned so it is unknown; there is no mention of a referencevariable (as there would be had the rule been "if Hi-Steam < Low-set-point ...", where "Low-set-point would be the reference variable"); the reference value is "20 and the dimensional unit is "psi"; the comparison is "<"; there is no mention of an associated function (as there would be in "If Hi-steam < AVE (20) psi ...", where AVE, for "average" is such a function); the action is "open", and the action value is "valve 1".

If the above elements do not capture all the aspects of the knowledge elements in the Knowledge Structure under investigation then design and add sufficient new elements to capture and characterize all the knowledge information.

2. Determine from system documentation or from system developer: (1) the rules of syntax of the knowledge elements, and (2) the allowable keywords, symbols, icons, and operators. If this cannot be done, then INTERRUPT.

3. o Start with the first knowledge element of the first partition of the Knowledge Structure. Fill out line(s) in the database setup in (1t) above until all the information in that knowledge element has been extracted.

4. Continue working in this way through all elements of all partitions of the Knowledge Structure until all are completed.

5. Then refer to technical sources or, more likely, discuss with the appropriate engineer, to determine two aspects about each row of information entered in the database.

- o whether there are any omissions, and
- o whether there are any errors

If this cannot be done, then INTERRUPT.

In looking for omissions, do NOT try to find unmentioned new system variables; there will be many, and adding them will not be a fruitful task. However, for each system variable present, ask the engineer (or look in the references) to see if there are any system variable-values that are possible but are not present; ask if there are any reference variables for that system variable that are possible but not mentioned; ask if there are any reference values which are significant (set-points, thresholds, etc.) but not present. For each piece of new information, enter a new row in the database, copying over the system variable.

EXECUTION (cont'd)

The following types of errors should be looked for in the database, and the following actions taken for them (make a record of the location and suspected problem of each erroneous knowledge element):

(1) Misspellings of any kinds -- correct
(2) Unrecognized variables, not misspellings -- remove element from main database and put in a separate one; try to insert an acceptable replacement in that spot that is consistent with the rest of the associated information. If replacement was found, then leave that row in the database; if no reasonable replacement can be found, remove that row and put in a separate database.
(3) Unrecognized dimensional units, not misspellings -- do as above
(4) Actions or operators which are not on key word list, not misspellings -- do as above
(5) Any instances of incorrect syntax -- correct if possible]

Note: These errors are being sought in the engineering information database being created as a result of knowledge acquisition of this information. These errors are not being looked for in the Knowledge Base; that occurs in Step 7.3.

6. The next step is to expand the database to include the additional fields. An alternative database example is shown below:

Variable Name	Description	Units	Range	Allowable Comparison	Reference State(s)	Description	Units	Implication State
Recirc Pump Spee	Operation speed or recirculation flow pump	rpm	0-3600	is (>2660rpm)	High operable range	Upper end of	rpm	Degrading
RHR Water-Leve	Residual Heat Removal Water-level	cm	0-10,000	is above	Shutdown cooling interlock level	Minimum water-level to operate shutdow cooling system	cm	Improving
Rods	Number of control rods that are presently inserted	#	0-12	are inserted	At or Beyond 06	Fully inserted	#	Improving

This will almost certainly require discussions with a knowledgeable engineer to obtain the needed information. (e.g., a nuclear engineer familiar with EOPs)

7. Finally, a knowledgeable engineer should be asked to consider each of the actions, implications, or assertions that occur during inferencing (e.g., the "THEN" part of IF-THEN rules in rule-based systems). The engineer should be shown the particular conditions that were tested or considered in arriving at the consequence and asked two questions about each: are the conditions considered reasonable and appropriate, and (2) is any important condition missing that would normally or logically be assessed before arriving at such a consequence?

To facilitate this inquiry, all of the rows of the database involving the same consequence should be grouped together.

8. The engineer of step 7 should next examine all actions classified as critical and identify the next steps that should be taken and the conditions governing their execution. This information should be added to the database.

9. Check database to see that all key words, symbols, icons, and operators determined from step 2 of execution occur somewhere in the Engineering Database. If they do not, enter missing ones in new rows in appropriate columns, a single missing entry per row.

10. It is only at this point that you are ready to check the knowledge base for semantic and meta-level knowledge errors.

ANALYSIS

- Determine M1, the % of system information which could not be checked in step 5 of execution.
- Determine M2, the % of expanded system information which could not be completed in step 6 of execution.

ACCEPT/ REJECT CRITERIA

- Accept if: M1≤1% AND M2≤2%; completion status = ACCEPT
- Conditionally Accept if: M1≤5% AND M2≤10%; completion status = CONDITIONALLY ACCEPT
- Reject if: M1>5% OR M2>10%; completion status = REJECT

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Engineering Database is prepared

PROGRAM MANAGER REVIEW

N/A for this activity

OUTPUT/ COMPLETION STATUS

if ACCEPT	→	continue to next activity
if CONDITIONALLY ACCEPT	→	correct deficiencies and repeat this activity
if REJECT	→	hold review with program manager to determine course of action, correct deficiencies

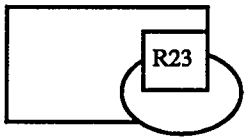
Terminal, lowest level activity, is not discussed on subsequent pages

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

P7.3 EXECUTE META-LEVEL KNOWLEDGE CHECKING

GOAL OF THIS ACTIVITY:

The goal of this activity is to use the Engineering Database created in 7.1 to check the Knowledge Structure for various kinds of semantic and knowledge errors.

Pre-Conditions/Trigger Conditions

- The Engineering Database has been created (cf 7.1)
- The list of knowledge elements discovered to have problems during preparation of the database is available
- The Knowledge Structure and Engineering Database have been installed on a computer text-editor
- Personnel are available
- Schedule dictates this activity begin

PLANNING	• Decide which portions of Knowledge Structure are most important to check first. If no priority, start at beginning.
PROGRAM MANAGER APPROVAL	N/A for this activity
SETUP	• Try to analytically determine the possible sequences or paths through the Knowledge Structure (e.g. rule-chains) and have these available for use in step 15 in execution.

EXECUTION

Definitions

• For the error types listed in 1 below, a "variable" is defined as a character-string name representing a generic class of numbers or character-strings having a particular domain. Variables are characterized as some type of number or as some non-numeric character-string type, and having a range characterized as a numerical interval or set (for numeric variables) or as a set of character-strings (for non-numeric variables). Variables may be associated in Knowledge Structures with other variables or with values of variables in the binding, assignment, or querying statements of knowledge elements. Such associations between a variable and something else are called by the general name "references", either a "reference-value" (a number or a literal character-string which is not a variable name, or a set of these or a "reference-variable" (the name of another variable). The association itself is called here a "comparison".

1. Review the following list of errors and anomalies to be searched for in the Knowledge Structure (those identified with an asterisk are anomalies).

(1) Word, symbol, icon, or operator is not recognized

(2) For numeric variables, the reference value is of an incorrect numeric data type

(3) For numeric variables, the reference value is outside the range of the variable

(4) For non-numeric variables, the reference value is not among those allowed

(5) For a comparison between a variable and a reference variable, one is numeric and the other is non-numeric

(6) For a comparison between a variable and a reference variable, both numeric, the variables have different numeric datatypes

(7) For a comparison between a variable and a reference variable, both numeric, the variables have different nonconformable dimensional units

* (8) For a comparison between a variable and a reference variable, both numeric, and both of the same datatype and unit, the variables are probably non-comparable because they come from different non-interactive physical subsystems or because they are both not meaningful in the same engineering or scientific perspective

* (9) For a comparison between a variable and a reference variable, both non-numeric, the variables do not have any possible values in common

(10) For a comparison between a variable and a reference variable, the variables are probably non-comparable because they come from different non-interactive physical subsystems or because they are both not meaningful in the same engineering or scientific perspective

(11) For a comparison between a variable and a reference variable, one numeric and the other non-numeric, the comparison operator used requires that both be numeric.

(12) For a comparison between a variable and a reference variable, of either kind, the comparison operator requires that they have certain domain features and this is not true of one or both.

(13) For clauses in the antecedent (IF) part of a knowledge IF-THEN rule which encode variable and reference comparisons (or the antecedent in a first-order or other logic statement), there is a logical contradiction between two of the clauses in that both are identical except that one is negated with a negation operator (e.g., "IF X is an adult AND X is not an adult...")

EXECUTION (cont'd)

(14) For the situation in (13), there is a logical contradiction between two of the clauses in that one clause is equivalent to the negation of another (e.g., "IF X is a child AND X is greater than 65 years old")

* (15) For the situation in (13), there is a logical redundancy between one of the clauses in that rule and a clause in (or result of) a previous rule which was chained to, or enabled, this rule (e.g., if the present rule tests for TEMP>100 but a rule which was a precursor already established that TEMP>100). Such redundancy implies erroneous conceptualization that might signal real errors.

* (16) An action (or state) classified as critical has been taken (or asserted). No rule tests for this situation and specifies the appropriate next-step for this situation. Such a rule perhaps should be present.

* (17) An action (or state) classified as critical has been taken (or asserted). There is a rule that tests for this situation and specifies the appropriate next step but it cannot be executed because one of its conditions cannot be true under these circumstances. For example the clause could be incorrect.

* (18) One of the tests in a set of conditions concerns whether some variable is in a normal or acceptable state. An action associated with these tests, however, is appropriate only if the tested variable is in an abnormal or unacceptable state. There may be an error in the conditions.

(19) Several actions (or assertions) are associated with a set of conditions, and there is a contradiction between two of them (e.g., "IF power is off then TURN_OFF machines and CONTINUE normal operations."

2. Review each of the individual knowledge elements believed to be erroneous in step 5 of 7.1. Verify from the Engineering Database that they are true errors.

   For error-types (1)-(12) of step 1, use the list presented in step 1 as the source of information needed to determine each type.

   For error-types (13)-(19) very careful reasoning is needed, and information about reasoning paths can be very helpful. That part of the knowledge base added in step 6 will be most useful.

3. Review the remaining elements of the Knowledge Structure for the 19 problems listed.

ANALYSIS

- Determine the number of errors, M3 

(items 1-7, 10-14, and 19, in step 1)
- Identify the number of show-stoppers, M4 

- Determine the number of major errors, M5 

- Determine the number of anomalous knowledge elements, M6 

(items 8-9, 15-18, in step 1)
  These are equivalent to the "number of minor errors"

ACCEPT/ REJECT CRITERIA

Refer to the thresholds in the Guideline Package that invoked this procedure. Let the number of show-stoppers, #SS, be M4; let the number of major errors, #Maj, be M5; let the number of minor errors, #Min, be M6

- Accept if: #SS ≤ threshold given in Guideline Package for this variable
- Conditionally Accept if: #Maj ≤ threshold given in Guideline Package for this variable
- Reject if: #Min ≤ threshold given in Guideline Package for this variable

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare a list of locations and classifications of all errors and anomalies. Include a copy of the problem knowledge element with the problem marked
- Suggest corrections for the problems, as possible

PROGRAM MANAGER REVIEW

- Review findings of report with program manager, especially if decision is not ACCEPT
- Discuss remediation or correction proposals

OUTPUT/ COMPLETION STATUS

if ACCEPT	→ continue to next activity
if CONDITIONALLY ACCEPT	→ correct deficiencies and repeat this activity
if REJECT	→ correct deficiencies and repeat this activity

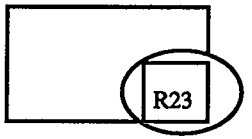
Terminal, lowest level activity, is not discussed on subsequent pages

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

M1 - Percentage of system information which could not be checked in P7.1 step 5 of Execution
M2 - Percentage of expanded system information which could not be completed in P7.1 step 6 of Execution
M3 - Number of errors
M4 - Percentage of errors
M5 - Number of anomalous knowledge elements
M6 - Percentage of anomalous knowledge elements

R29 - Miller, L.A., J.E. Hayes, and Steven M. Mirsky. Guidelines for Verification and Validation of Expert Systems. Volume 4 of this report. Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute. September 1993.

R48A - Stachowitz, R.A. "Validation of KB Sys." Final Report for Rome Laboratory, Report No. RL-TR-91-361, December 1991.

GUIDELINE PROCEDURE P8.0

P8.0 ROBUSTNESS TESTING

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform robustness 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.

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) and how output will be analyzed to determine correctness

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) Determine the systematic tests to be run. These involve a systematic violation of implicit or explicit expectations concerning input or processing features such as the following: R21 R36A
  a) type constraints - classify data into various data types; specify possible values explicitly or implicitly (as ranges, functions, etc.). Prepare test cases to violate these constraints.
  b) range/value constraints - identify input variables which have numeric range specifications or non-numeric constrained sets of values. Prepare test cases to violate these ranges and values.
  c) context constraints - specify the circumstances under which a particular data item or input is valid. This can be ttemporal, spatial, etc. Prepare test cases to violate these constraints.
  d) well-formedness criteria - specify appropriate data formatting, designated sequence of items, etc. Prepare test cases to violate these constraints.

- 2) Determine the brute force tests to be run. Based on the specific hardware and user interface of the system, generate test cases which have a chance of causing havoc. For example, pressing all the function keys singly and simultaneously.

ANALYSIS

- The number of functional test cases will be determined 

- The number of Software Trouble Reports Defects (by priority (1-2) will be determined 

  STRs should be prioritized according to a CM standard (such as DOD-STD-2167A) with priority 1 being the most severe.
- The number of defects per line of code (LOC) will be determined
- The % of test cases failed is determined

o A suggested correction for each defect will be prepared

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.*

ACCEPT/ REJECT CRITERIA

- Accept if: (M40+M41)=0 and M39 ≤ (0.2*#LOC tested) and M45 ≤ 0.2 and M46 ≤ 0.4%; Completion status = Accept
- Conditionally Accept if: 0< (M40+M41)≤2% and (0.2* #LOC tested) < M39≤ (0.4*#LOC tested) and 0.2 < M45≤0.4 and 0.4 < M46≤6%; Completion status = Conditionally Accept
- Reject if: (M40+M41)>2 or M39>(0.4*#LOC tested) or M45 > 0.4 or M46 < 6%; Completion status = Reject

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare the V&V random testing report
- o Prepare suggested corrections for each defect (could be appendix to the report)
- o Prepare "impact statements" for each priority 1 STR
- o Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the defects

PROGRAM MANAGER REVIEW

- Present the V&V random testing report to the Program Manager
- o Present the informal "Plan of Action" for correcting the defects to the Program Manager
- o Present the "impact statements" for priority 1 STRs to the Program Manager
- V&V functional testing report cannot be given to the developer until Program Manager approval given

OUTPUT/ COMPLETION STATUS

if ACCEPT	→ continue to next activity
if CONDITIONALLY ACCEPT	→ correct deficiencies and repeat this activity
if REJECT	→ correct deficiencies, hold review with program manager to determine course of action

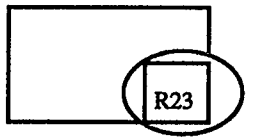
Terminal, lowest level activity, is not discussed on subsequent pages

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
M45 - Number defects/LOC
M46 - Percentage test cases failed defects (#test cases failed / M37 *100)
M101 - Number of robustness test cases

R36A - Miller, L.A., E. Groundwater, S.M. Mirsky. Guideline for Verification and Validation of Expert Systems, Volume 2 of this report. Science Applications International Corporation, prepared for U.S. Nuclear Regulatory Commission and Electric Power Research Institute. September 1993.

GUIDELINE PROCEDURE P9.0

P9.0 REGRESSION TESTING

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform regression testing on the system. Changes have been made to the previously test system (source code and/or knowledge based changes. Results of previous testing are available.

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).
- Prepare informal functional testing plan showing tasks, schedules, resources, etc.
- Select tools to be used if any
- 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

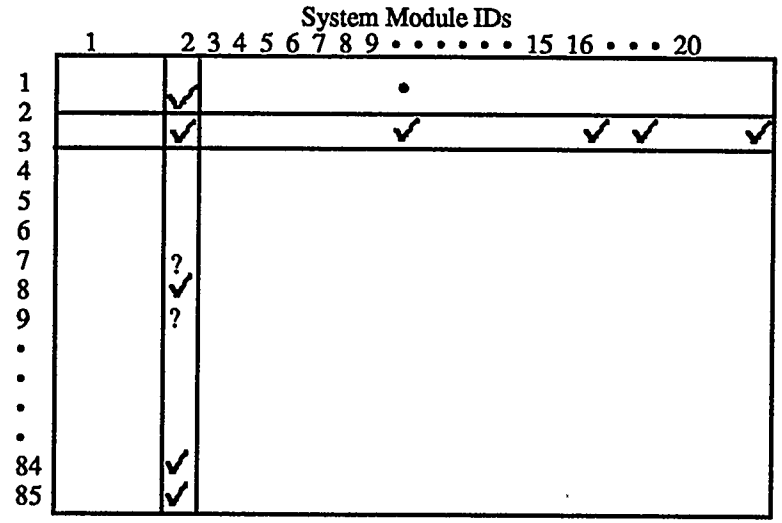
SETUP

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

EXECUTION

- 1) Prepare Regression Planning Matrix" which describes which test procedures, test cases, etc. apply to system programs and modules. For example:

SAMPLE RETEST PLANNING MATRIX

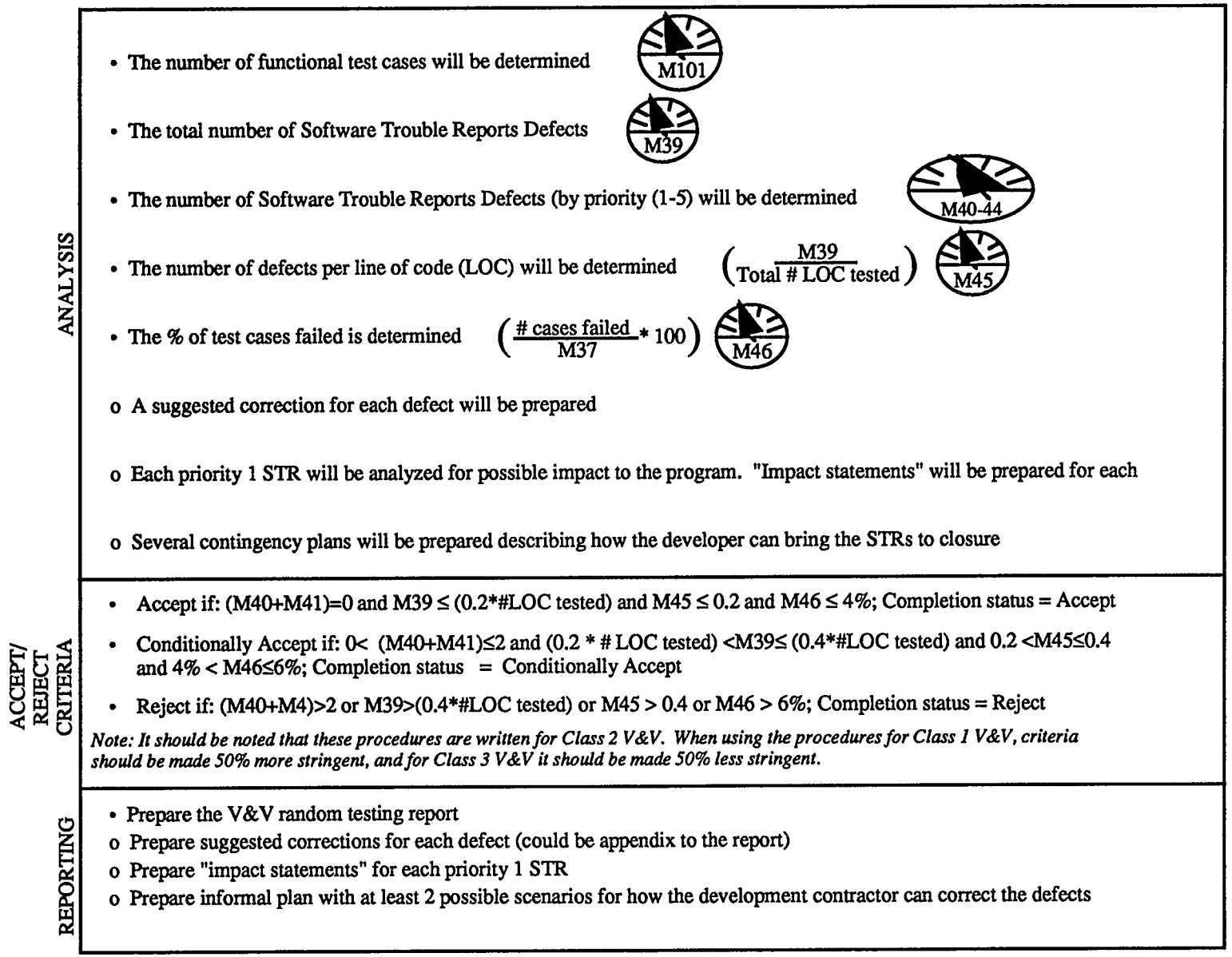


A "check" entry indicates that the case must be retested when the module is changed. A "question" entry indicates that the case may need to be retested (case-by-case basis). No entry indicates the test is not required. Ensure that as cases and/or modules are added, the "Regression Planning Matrix" is updated appropriately.

- 2) Perform a subset of the top-level cases in the system test set. Perform each test case pertaining to a changed module and/or knowledge base. Record the results in the test log, noting each defect discovered in a Software Trouble Report (STR). [IF an unacceptable number of STRs are found at any point during the testing, INTERRUPT] *NOTE: As opposed to developmental testing, regression testing usually involves running large, complex test cases first followed by simpler, localized cases to isolate problems. R20

o 3) Perform all tests in the system test set to ensure that all functions and system capabilities perform as required and have not been corrupted by the changes. Record the results in the test log, noting each defect discovered in an STR. [IF an unacceptable number of STRs are found at any point during the testing, INTERRUPT]

o 4) Utilize a capture/playback tool to perform regression testing. *NOTE: The tests must be "captured" during initial testing so they can be "replayed" for regression testing. It is suggested that the tool possess the following features: test editor (to allow editing of test data); smart comparator (to ensure that blank lines, etc. do not get taken into consideration); internal/external (tool can be used for unit or system testing); single/multi-thread (should be able to run one line at a time or an entire scenario); and exception reports (comparison failures are reported). R16


ANALYSIS
• The number of functional test cases will be determined
M101
• The total number of Software Trouble Reports Defects
M39
• The number of Software Trouble Reports Defects (by priority (1-5) will be determined
M40-44
• The number of defects per line of code (LOC) will be determined
(M39 / Total # LOC tested)
M45
• The % of test cases failed is determined
(# cases failed / M37 * 100)
M46
o A suggested correction for each defect will be prepared
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
ACCEPT/ REJECT CRITERIA
• Accept if: (M40+M41)=0 and M39 ≤ (0.2*#LOC tested) and M45 ≤ 0.2 and M46 ≤ 4%; Completion status = Accept
• Conditionally Accept if: 0< (M40+M41)≤2 and (0.2 * # LOC tested) <M39≤ (0.4*#LOC tested) and 0.2 <M45≤0.4 and 4% < M46≤6%; Completion status = Conditionally Accept
• Reject if: (M40+M4)>2 or M39>(0.4*#LOC tested) or M45 > 0.4 or M46 > 6%; Completion status = Reject
Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.
REPORTING
• Prepare the V&V random testing report
o Prepare suggested corrections for each defect (could be appendix to the report)
o Prepare "impact statements" for each priority 1 STR
o Prepare informal plan with at least 2 possible scenarios for how the development contractor can correct the defects

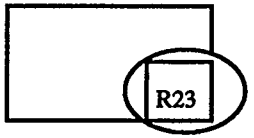
PROGRAM MANAGER REVIEW	• Present the V&V random testing report to the Program Manager o Present the informal "Plan of Action" for correcting the defects to the Program Manager o Present the "impact statements" for priority 1 STRs to the Program Manager • V&V functional testing report cannot be given to the developer until Program Manager approval given
OUTPUT/ COMPLETION STATUS	if ACCEPT → continue to next activity if CONDITIONALLY ACCEPT → correct deficiencies and repeat this activity if REJECT → correct deficiencies, hold review with program manager to determine course of action

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 of test cases failed defects $\left(\frac{\#\text{test cases failed}}{M37} * 100\right)$
M100 - Number of regression test cases

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

R20 - Hatley, D., and I. Pirbhai, Strategies for Real-Time System Specification. New York: Dorset House, 1987.

GUIDELINE PROCEDURE P10.0

P10.0 AUTOMATED SYNTAX CHECKING OF KNOWLEDGE STRUCTURES

WHEN TO USE THIS GUIDELINE:

The goal of this guideline procedure is to perform static analysis of the knowledge structure for known types of errors as well as anomalies using automated testing packages. This guideline can be applied to frame-based, rule-based or object-oriented systems when the knowledge is represented declaratively in some kind of data structure (vs. being represented in a programming procedure). The knowledge structure source must be available in computer media format.

Use this guideline as a means of separately testing the knowledge structure without executing the system of which it is a part. This guideline should be used before any dynamic-testing procedures.

Pre-Conditions/Trigger Conditions

- Knowledge Structure source is available on computer media
- The automated testing package has been chosen, received, installed, and exercised and performs as intended *
- It is known that it is possible and feasible to convert the knowledge structure into the form required by the testing package
- Personnel able to setup, run, and interpret the testing package results are available
- Schedule dictates that activity commence

*K29

PLANNING

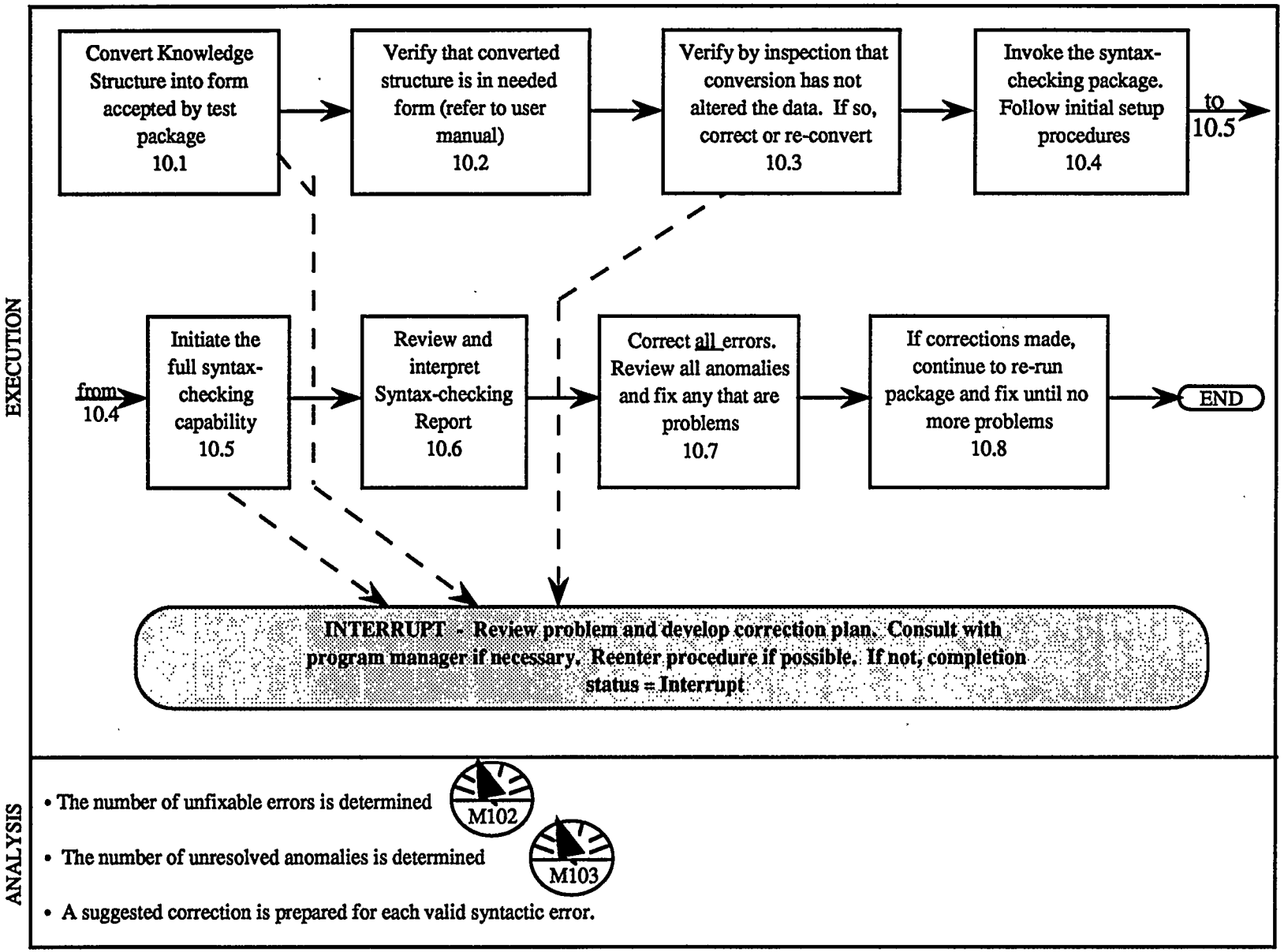
- Review the high level tasks (see Execution below)
- Determine availability of computer resources required by the testing package
- Establish schedule and assign resources, for each task, based on availability of resources and external schedule for this procedure
- Develop an informal syntax checking plan showing expected start/stop times and assigned resources for each task
- Identify any dependencies on other tasks or personnel as well as any approval or management assistance actions needed
- Ensure required editor and other tools are available on same computer terminal as the automated syntax checking package

PROGRAM MANAGER APPROVAL

- Present plan and dependencies/actions-needed to the Program Manager for approval

SETUP

- Load knowledge structure source onto computer terminal
- Have user manual for testing package readily available


EXECUTION
Convert Knowledge Structure into form accepted by test package 10.1
Verify that converted structure is in needed form (refer to user manual) 10.2
Verify by inspection that conversion has not altered the data. If so, correct or re-convert 10.3
Invoke the syntax-checking package. Follow initial setup procedures 10.4
to 10.5
from 10.4
Initiate the full syntax-checking capability 10.5
Review and interpret Syntax-checking Report 10.6
Correct all errors. Review all anomalies and fix any that are problems 10.7
If corrections made, continue to re-run package and fix until no more problems 10.8
END
INTERRUPT - Review problem and develop correction plan. Consult with program manager if necessary. Reenter procedure if possible. If not, completion status = Interrupt
ANALYSIS
• The number of unfixable errors is determined
M102
• The number of unresolved anomalies is determined
M103
• A suggested correction is prepared for each valid syntactic error.

ACCEPT/ REJECT CRITERIA

- Accept if: M102=0 and M103 = 0
- Conditionally Accept if: $0 < M102 \leq 1$ and $0 < M103 \leq 3$
- Reject if: M102>1 or M103>3

Note: It should be noted that these procedures are written for Class 2 V&V. When using the procedures for Class 1 V&V, criteria should be made 50% more stringent, and for Class 3 V&V it should be made 50% less stringent.

REPORTING

- Prepare report on errors and anomalies found on each run
- Describe attempts to fix errors and resolve anomalies
- Describe termination status and why. IF in "Interrupt" mode, describe plan for recovery

PROGRAM MANAGER REVIEW

- Inform the Program Manager of results, provide a copy of findings
- Discuss any recovery plans

OUTPUT/ COMPLETION STATUS

if ACCEPT	→	continue to next activity
if CONDITIONALLY ACCEPT	→	correct deficiencies and repeat this activity
if REJECT	→	correct deficiencies, hold review with program manager to determine course of action
if INTERRUPT	→	develop correction plan and reenter procedure if possible

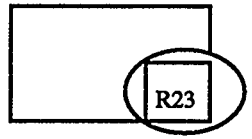
Terminal, lowest level activity, is not discussed on subsequent pages

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

M102 - Number of unfixable errors is determined

M103 - Number of unresolved anomalies is determined

R29 - Miller, L.A., J.E. Hayes, and Steven M. Mirsky. Guidelines for Verification and Validation of Expert Systems. Volume 4 of this report. Science Applications International Corporation for U.S. Nuclear Regulatory Commission and Electric Power Research Institute. September 1993.

GUIDELINE PROCEDURE P11.0

P11.0 FORMAL CUSTOMER REVIEW

WHEN TO USE THIS GUIDELINE:

The goal of this guideline is to perform formal customer review of the requirements and/or design for the system. For a Formal Customer Review of the requirements, a requirements specification has been received. For a Formal Customer Review of the design, a requirements specification has been received and has been the subject of requirements analysis.

Pre-Conditions/Trigger Conditions

- Requirements specification has been delivered
- 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).
- Prepare informal regression testing plan showing tasks, schedules, resources, etc.
- Select tools to be used if any (e.g., word processing package, software for scanner, etc.)

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 hard copy (and if possible softcopy) of the source code, a computer, and word processing software

EXECUTION

- 1) Identify desired participants for Formal Customer Review. At least one representative from each of the following groups should be included:

  System Users
  Program Manager
  Acquisition Manager
  System Maintainers
  Customers
  Consultants
  Developers
  Subejct Matter Experts
  Technical Editors
  V&V

  This group of individuals will be referred to as the customer review panel.

o 2) Prepare a requirements (or design) package for each participant and distribute the package at least one week prior to the review.

- 3) Assign a narrator, a scribe, and an action item gatherer. The roles for each follow:

Narrator	runs the review, "reads" each requirements and facilitates the group's critique of each requirement.
Scribe	maintains the master "marked-up" list of requirements
Action Item Gatherer	maintains the list of action items, assignees, assigned by, and due dates

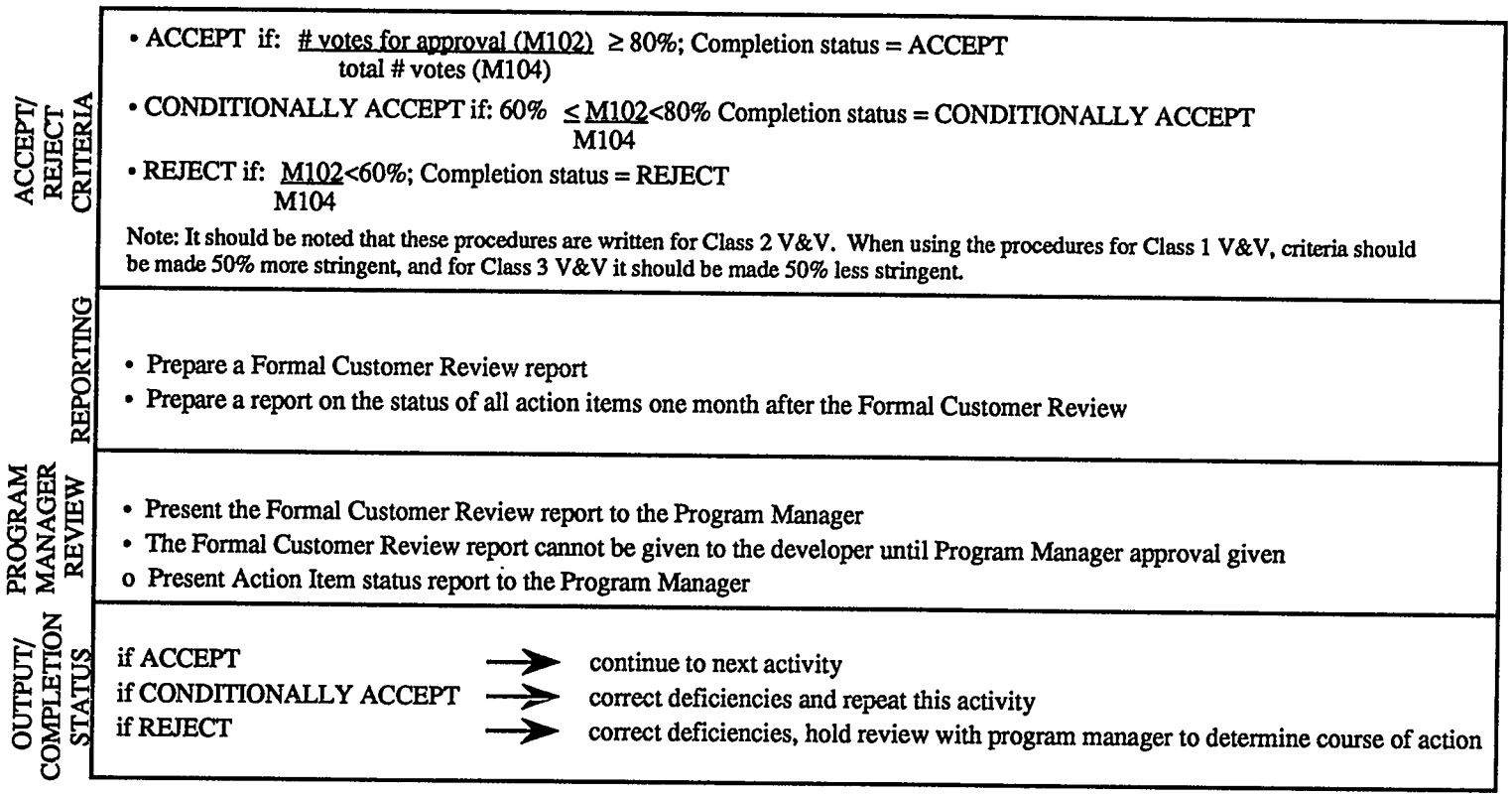
  NOTE: For a small review, the scribe and the action item gatherer could be the same individual.

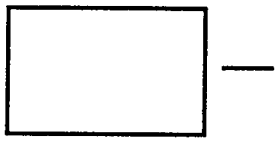
- 4) Step through each requirement and solicit feedback from the customer review panel, gathering action items and marking up the requirements as the meeting proceeds.

- 5) Prepare a list of action items from the review as well as a marked-up copy of the requirements and distribute to all the members of the customer review panel.

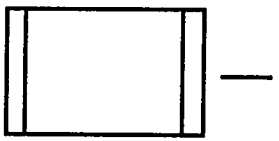
- 6) Request that each member of the customer review panel indicate approval (or disapproval) of the requirements specification.

- 7) Follow up on all open action items.





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

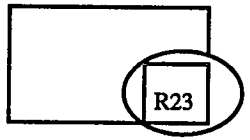


High level (non-terminal) activity (decomposes to lower level activities), discussed on subsequent pages

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

M102 - Number of unfixable errors is determined

M103 - Total number of Reviewer votes

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
SAIC-95/1028
Vol. 7

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

User's Manual

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 provides a step-by-step guide, or user manual, for personnel responsible for the planning and execution of the verification and validation (V&V), and also developmental testing, of expert systems, conventional software systems, and also various other types of artificial intelligence systems. While the guide was developed primarily for applications in the utility industry, it applies well to all industries. The user manual has three sections. In Section 1 the user assesses the stringency of V&V needed for the system under consideration, identifies the development stage the system is in, and identifies the component(s) of the system to be tested next. These three pieces of information determine which package of V&V methods, called a Guideline Package, is most appropriate for those conditions. The V&V Guidelines Packages are provided in Section 2. Each package consists of an ordered set of V&V techniques to be applied to the system, along withguides as to the review/evaluation team, and the measurement criteria. In Section 3, the details of 11 of the most important (or least-well explained in the literature) methods are presented to assist the user in the accurate application of these techniques.

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