

ENCLOSURE 1

WATTS BAR NUCLEAR PLANT
VERIFICATION AND VALIDATION PLAN
FOR
EAGLE-21 DIGITAL INSTRUMENTATION

8701050197 861229
PDR ADCK 05000390
A PDR

WATTS BAR NUCLEAR PLANT

VERIFICATION AND VALIDATION PLAN

FOR

EAGLE-21 DIGITAL INSTRUMENTATION



TABLE OF CONTENTS

<u>Title</u>	<u>Page</u>
1.0 Introduction	4
1.1 Purpose	4
1.2 System Functions	4
1.3 System Architecture	4
2.0 References	6
3.0 Definitions	8
4.0 System Development	10
5.0 System Verification	11
5.1 Introduction	11
5.2 Verification Philosophy	11
5.3 Verification Techniques	12
5.3.1 Reviews	13
5.3.1.1 Design Documentation Review	13
5.3.1.2 Source Code Review	13
5.3.1.3 Functional Test Review	13
5.3.2 Software Testing	13
5.3.2.1 Structural Testing	13
5.3.2.2 Functional Testing	14
5.4 Verification Matrix	15
5.4.1 Safety Classification	15
5.4.2 Hierarchical Level of Software Components	15
5.4.3 Justification of Matrix Elements	15
5.4.3.1 Class 1E Associated Software	16
5.4.3.2 Non-Class 1E Associated Software	16

TABLE OF CONTENTS

<u>Title</u>	<u>Page</u>
5.4.4 Application of the Verification Matrix and Criteria Utilized for Software Testing for the Eagle-21 Replacement Hardware	16
5.4.4.1 Application of the Verification Matrix	16
5.4.4.2 Criteria Utilized for Software Testing	17
6.0 System Validation	18
6.1 Validation Philosophy	18
6.2 Validation Testing Overview	18
6.2.1 General Description	19
6.2.2 Functional Requirements Testing	20
6.2.3 Abnormal-Mode Testing	20
6.2.4 System Prudency Review Testing	21
7.0 Development, Verification and Validation Team Organization	22
7.1 Development Team	22
7.1.1 Chief Programmer	22
7.1.2 Programmers	23
7.2 Verification Team	23
7.2.1 Chief Verifier	23
7.2.2 Verifiers	23
7.2.3 Librarian	24
7.3 Validation Team	24
7.3.1 Chief Verifier	24
7.3.2 Functional Requirements Decomposer	24
7.3.3 Lead Validator	24
7.3.4 Test Engineer	25
7.3.5 Librarian	25
7.3.6 Test Technician	25

1.0 INTRODUCTION

1.1 Purpose

The purpose of this plan is to provide a description of the design, verification, and validation process and the general organization of activities that are being used in these areas on the Eagle-21 Process Protection System replacement hardware. The material contained herein is modeled after the guidance provided in (a) the 414 Integrated Protection System Prototype Verification Program, which was presented to the NRC in 1977 as part of the Westinghouse RESAR 414 system, (b) ANSI/IEEE-ANS-7-4.3.2-1982 and (c) Regulatory Guide 1.152, and (d) the Design, Verification, and Validation Plan implemented for the South Texas Qualified Display Processing System (QDPS).

1.2 System Functions

The Eagle-21 Process Protection System replacement hardware performs the following major functions:

1. Reactor Trip Protection (Channel Trip to Voting Logic)
2. Engineered Safeguard Features (ESF) Actuations.
3. Isolated Outputs to Control Systems, Control Panels, and Plant Computers.
4. Isolated Outputs to information displays for Post Accident Monitoring (PAM) indication.
5. Automatic Surveillance Testing to verify channel performance.

1.3 System Architecture

The Eagle-21 System Architecture is shown in Figure 1. The basic subsystems are:

1. Loop Processor Subsystem

The Loop Processor Subsystem receives a subset of the process signals, performs one or more of the protection algorithms, and drives the appropriate channel trip (or partial engineered safeguards actuation) signals. It also drives the required isolated outputs.

2. Tester Subsystem

The Tester Subsystem serves as the focal point of the human interaction with the channel set. It provides a user-friendly interface that permits test personnel to configure (adjust setpoints and tuning constants), test, and maintain the system.

3. Input/Output (I/O)

The microprocessor based system interfaces with the field signals through various input/output (I/O) modules. These modules accommodate the plant signals and test inputs from the Tester Subsystem, which periodically monitors the integrity of the Loop Processor Subsystem.

2.0 REFERENCES

The following is a list of relevant industrial standards which were considered in the development of this plan:

1. ANSI/IEEE-ANS-7-4.3.2.-1982, "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations"
2. IEEE Std. 279-1971, "Criteria for Protection Systems for Nuclear Power Generating Stations"
3. IEEE Std. 603-1980, "Criteria for Safety Systems for Nuclear Power Generating Stations"
4. WCAP 9153, "414 Integrated Protection System Prototype Verification Program," Westinghouse Electric Corp., August 1977.
5. WCAP 9740, "Summary of the Westinghouse Integrated Protection System Verification and Validation Program," Westinghouse Electric Corp., September 1984.
6. Regulatory Guide 1.97, Rev. 2, "Instrumentation for Light-Water-Cooled Nuclear Power Plants to Assess Plant and Environs Conditions During and Following an Accident," December 1980
7. ANSI/ASME NQA-1-1983, "Quality Assurance Program Requirements for Nuclear Power Plants"
8. IEEE Std 729-1983, "Standard Glossary of Software Engineering Terminology"
9. IEEE Std 730-1981, "Standard for Software Quality Assurance Plans"
10. IEEE Std 828-1983, "Standard for Software Configuration Management Plans"
11. IEEE Std 829-1983, "Standard for Software Test Documentation"
12. IEEE Std 830-1984, "Guide to Software Requirements Specifications"
13. NBS Special Publication 500-75 (February 1981), "Validation, Verification and Testing of Computer Software"
14. NBS Special Publication 500-93 (September 1982), "Software Validation, Verification, Testing Technique and Tool Reference Guide"
15. NBS Special Publication 500-98 (November 1982), "Planning for Software Validation, Verification and Testing"
16. IEC SC 45A/WG-A3 (January 1984), "Draft: Software for computer in the Safety System of Nuclear Power Stations"

17. Regulatory Guide 1.152, "Criteria for Programmable Digital Computer System Software in Safety-Related Systems of Nuclear Power Plants"
18. Regulatory Guide 1.153, "Criteria for Power, Instrumentation, and Control Portions of Safety Systems"
19. Design, Verification and Validation Plan for the South Texas Project - Qualified Display Processing System. Design Specification Number 955842, Revision 3, July 1985.

3.0 DEFINITIONS

The definitions in this section establish the meaning of words in the context of their use in this plan.

COMPUTER SOFTWARE BASELINE - The computer program, computer data and computer program documentation which comprises the complete representation of the computer software system at a specific stage of its development.

DESIGN REVIEW - A meeting or similar communication process in which the requirements, design, code, or other products of a development project are presented to a selected individual or group of personnel for critique.

FUNCTIONAL TESTING (FT) - Exercise of the functional properties of the program to the design requirements.

FUNCTIONAL TEST REVIEW (FTR) - A review which is performed on the documented functional tests that were run by the programmer on his code.

INSPECTION - An evaluation technique in which software requirements, design, code, or other products are examined by a person or group other than the designer to detect faults, differences between development standards, and other problems.

INTEGRATION TESTS - Tests performed during the hardware-software integration process prior to microprocessor system validation to verify compatibility of the software and the microprocessor system hardware.

MODULE (M) - Refers to a significant partial functional capability of a subprogram and consists of more than one unit. Modules are usually stand-alone procedures or routines which may call other lower level modules or units.

PEER REVIEW - An evaluation technique in which software requirements, design, code, or other products are examined by persons whose rank, responsibility, experience, and skill are comparable to that of the designer.

PROGRAM - Totality of software in a system or one independent part of software of a distributed system implemented by a particular CPU.

SOFTWARE DESIGN SPECIFICATION (SDS) - A document which represents the designer's definition of the way the software is designed and implemented to accomplish the functional requirements, specifying the expected performance. An SDS can be for a system, subsystem, module, or unit.

SOFTWARE DEVELOPMENT PERSONNEL - A team of individuals or an individual assigned to design, develop and document software.

SOFTWARE TEST SPECIFICATION (STS) - A document detailing the tests to be performed, test environment, acceptance criteria and the test methodology. An Approved SDS document forms the basis for the STS.

SOURCE CODE REVIEW (SCR) - A review which is performed on the source code.

SUBPROGRAM (SP) - Refers to a major functional subset of a program and is made up of one or more modules. A subprogram is typically represented by the software executed by a single processor.

STRUCTURAL TESTING (ST) - Comprehensive exercise of the software program code and its component logic structures.

UNIT (U) - The smallest component in the system software architecture, consisting of a sequence of program statements that in aggregate perform an identifiable service.

VALIDATION - The test and evaluation of the integrated computer system to ensure compliance with the functional, performance and interface requirements

VERIFICATION - The process of determining whether or not the product of each phase of the digital computer system development process fulfills all the requirements imposed by the previous phase.

VERIFIER(S) - An individual or group of individuals assigned to review source code, generate test plans, perform tests, and document the test results for a microprocessor system. If the activity is extensive, a chief verifier will be appointed to guide and lead the Verification and Validation personnel.

VERIFICATION TEST REPORT (VTR) - A document containing the test results. In conjunction with the Software Test Specification it contains enough information to enable an independent party to repeat the test and understand it.

4.0 SYSTEM DEVELOPMENT

The development of the Eagle 21 System, as shown in Figure 2, involves three stages:

1. Definition
2. Design
3. Implementation and Test

A brief description of each stage is given below:

- 1) The definition stage is characterized by the statement of the objective to be achieved, the construction of an initial project plan, and a high-level definition of the system. During this stage, the overall functional requirements of the system are identified. Within Westinghouse, these requirements are brought together in a System Design Requirements document.
- 2) The design stage is characterized by the decomposition of these System Design Requirements into System Design Specifications and Hardware and Software Design Specifications of sufficient detail to enable the implementation of the system. The Software Design Specifications for the system are then further decomposed into subsystem, module and unit specifications.
- 3) The implementation and test stage is characterized by the actual construction of the hardware, coding of the various software entities, and testing. The software development team is responsible for the writing, assembling, testing, and documenting the computer code. As the software entities are completed, beginning at the unit level, they are formally turned over to the verifiers for final independent review and/or testing as specified in Section 5.0.

Software development can be viewed as a sequence of well-defined steps similar to system development. The System Design Specification is used to generate Software Design Specifications which in turn are used to develop high level language programs. These programs are converted by a compiler into assembly language, then by the assembler into machine code. The linker combines groups of assembled code with the library to produce relocatable object code for input to the loader. The loader generates the absolute code which is then burned into read only memory (ROM).

The use of a high level language allows the designer to express his ideas in a form that is more natural to him. The computer adjusts to his language and not he to the language of the computer. Software written in a high level language is more readily reviewed by an independent party who may not be familiar with the computer assembly language instruction set. Some features of the high level language aid the development of reliable software. For example, block structuring helps identify and reduce the number of possible execution paths.

As part of testing, the various hardware components and software entities are assembled in a stepwise manner. Additional testing at each step ensures that each component performs its required function when integrated with its associated components.

The final activity associated with the system implementation and testing stage is the testing of the system. A system test plan is derived from the system functional requirements and system design specifications to confirm that the system exhibits a level of functionality and performance which meets or exceeds the stated requirements. This final system test is referred to as the Factory Acceptance Test.

Several design assurance techniques are utilized throughout all stages of the development process to ensure that the hardware and software components meet the required specifications.

Formal design reviews are held within Westinghouse to ensure that the System Design Specifications meet the System Functional Requirements. The design review team consists of a group of knowledgeable multidisciplinary engineers to ensure that all aspects of the design are reviewed.

During the implementation and test stage, acceptance testing and review are conducted by the designers on the hardware components, circuit boards, and subsystems to ensure they exhibit a level of functionality consistent with the Hardware Design Specifications and Software Design Specifications.

The final design assurance technique utilized is the execution of the system Factory Acceptance Test to ensure the system performance meets the system functional requirements and system design specifications.

5.0 SYSTEM VERIFICATION

5.1 Introduction

With the application of programmable digital computer systems in safety systems of nuclear power generating stations, designers are obligated to conduct independent reviews of the software associated with the computer system to ensure the functionality of software to a level consistent with that described in the system requirements.

Section 5.2 provides an overview of the verification philosophy. Section 5.3 describes the verification techniques utilized in performing the verification process. Section 5.4 describes the matrix that the verification personnel use for determining the level of verification that should be applied to each software entity. Section 5 concludes by defining the application of the verification matrix to the Eagle-21 Replacement Hardware.

5.2 Verification Philosophy

Figure 2 illustrates the integration of the system verification and validation process with the system design process. The verification process may be divided into two distinct phases: verification of design documentation, and verification of software.

As shown on figure 2, independent verification is performed at each step of the definition and design stages. For example, independent verification will occur to ensure that the decomposition from the functional requirements and the software requirements to the system design requirements has been performed properly and thoroughly. Similarly, an independent review will be conducted in the decomposition from the system design requirements to the system design specification. Figure 2 illustrates where an independent review and signoff will be conducted during the design process. Verification of the design documentation will be completed prior to the implementation and test phase.

During the implementation and test stage, when the writing, testing, assembling, and documenting associated with each software entity (beginning at the unit level) is completed by the design team, the software entity is formally turned over to the verifier. At this point, an independent review and/or testing of the software entities is performed to verify that the functionality of the software entities meet the applicable Software Design Specifications. After the verifier is satisfied that all requirements are met, the software is configured for use in the final system and subsequent system validation process.

Figure 3 illustrates the philosophy utilized in conducting the software verification process. The verification process begins at the unit software level, i.e., the simplest building block in the software. After all software units that are utilized in a software module are verified, the verifier proceeds to verify that module. Not only is the software module verified to meet the module Software Design Specification, but the verifier ensures that the appropriate units are utilized in generating the software module.

After all software modules necessary to accomplish a software subprogram are verified to meet the applicable Software Design Specifications, the verifier proceeds to verify that subprogram. As in the case of the software module, the verifier not only verifies that the subprogram meets the applicable Software Design Specifications, but also verifies that the appropriate software modules were utilized in generating the subprogram entity. This verification philosophy ensures that the verifier tests and/or reviews the interface between the software unit, module and subprogram entities.

Depending upon the hardware implementation, the verification process may utilize system hardware in the verification of the software modules and subsystems.

5.3 Verification Techniques

Verification techniques used in microprocessor based systems development fall into two basic categories: review and testing.

5.3.1 Reviews

There are three types of reviews used in the independent verification process: Design documentation reviews, code reviews and functional test reviews.

5.3.1.1 Design Documentation Review

This activity involves an independent review of the System Design Requirements and Specifications to ensure that all of the appropriate Functional Requirements have been satisfied.

5.3.1.2 Source Code Review

Source code review, as opposed to code testing, is a verification method in which the software program is examined visually. The operation of the software is deduced and compared with the expected operation. In effect, the operation of the software is simulated mentally to confirm that it agrees with the specification.

Source code reviews will be used to verify the transformation from a Design Specification into high level code. High level code is easy to read and understand, and therefore full inspection at that level is feasible.

5.3.1.3 Functional Test Review

A functional test review is a review by the verifier of the documentation associated with the functional tests which were performed by the designer. This review will provide a high degree of assurance that the software performs the functions specified in the design requirements.

5.3.2 Software Testing

Software tests can be divided into two categories: structural and functional.

5.3.2.1 Structural Testing

Structural testing, which attempts to comprehensively exercise the software program code and its component logic structures, is usually applied at the unit level. The functionality of the program is verified along with the internal structure utilized within the program to implement the required function.

Structural testing requires that the verifier inspect the code and understand how it functions before selecting the test inputs. The test inputs should be chosen to exercise all the possible control paths within the software component. If this is not possible, the test inputs should be chosen to exercise every statement within the component. For example, if a trigonometric function is calculated in several different ways, depending on the range of the input argument, then the test inputs include tests for the argument in each of these ranges, as well as on the boundaries between ranges. In particular, they exercise the upper limit, the lower limit, and at least one intermediate value within each range.

5.3.2.2 Functional Testing

In the functional approach to program testing, the internal structure of the program is ignored during the test data selection. Tests are constructed from the functional properties of the program which are specified in the Design Specification. Functional testing is the method most frequently used at the module or subsystem level. Examples of functional testing include random testing and special cases by function.

Random testing is the method of applying a test input sequence chosen at random. The method can be used in the following circumstances: to simulate real time events that are indeed random; to increase the confidence level in the correctness of a very complex module; to test a subsystem or a system where it is not necessary to test all the possible paths; to get a quantitative measure on the accuracy of a numeric calculation; or to get a measure of the average time required by some calculation.

Special cases by function can be deduced from the Design Specification of the module and will determine some test cases. For example, a subroutine for matrix inversion should be tested using almost-singular and ill-conditioned matrices. Subroutines which accept arguments from a specified range should be tested with these arguments at the extreme points of the range. An arithmetic package should be tested with variables which have the largest and smallest mantissa, largest and smallest exponent, all zeroes, and all ones and negative variables.

5.4 Verification Matrix

The choice of particular verification techniques to be utilized on a system component is a function of the following parameters:

- A. The safety classification of the system
- B. The hierarchical level of the software component (unit, module or subprogram)

5.4.1 Safety Classification

The safety classification of an item is defined according to IEEE-279-1971 and IEEE Std 603-1980. In general, the safety classification of the system establishes the verification requirements for the system. However, since all the components contained in the system do not necessarily perform equal safety functions, a higher or lower level of verification may be assigned to specific system components depending on the exact functions performed. If a different level of verification is assigned to a component, the interactions between that component and the other components in the system must be carefully considered and reviewed.

5.4.2 Hierarchical Level of Software Components

For software that is organized in a hierarchical structure, the intricacies of the actual code can not be easily grasped at the upper levels. For all but simple systems it is prudent to approach verification in a progressive manner, beginning at the unit level. It is at the unit level that the code can be most easily inspected or comprehensively tested as necessary.

As the software is built up into higher level components during the integration stage, it becomes possible to demonstrate complete processing functions. This process allows the validation of functional performance requirements. Thus, validation testing assumes a functional theme, with the main emphasis on the interaction between subsystems and their interfaces.

5.4.3 Justification of Matrix Elements

Considering the parameters detailed above, different verification methods are required for different subsystems and software components. Figure 4 illustrates, in tabular form, the levels of verification. The software component columns identify the levels of software. Each element of the matrix specifies the type of testing or review that will be performed on the software component within that classification. The justification of each matrix element follows.

5.4.3.1 Class 1E Associated Software

The software associated with actuation and/or implementation of reactor trip, engineered safety features, and information displays for manually controlled actions (as defined by IEEE Std. 279-1971 and IEEE Std. 603-1980) must receive the highest level (level 1) of verification identified. As such, all software must be structurally tested to ensure that all lines indeed meet the intended design specification. Since the plant operators rely upon the automatic actuation of the reactor trips and/or engineered safeguards actuations, as well as information displays for manually controlled actions, the highest level of confidence must be afforded.

5.4.3.2 Non-Class 1E Associated Software

Any associated software that is not directly related to Class 1E variables will receive level 2 verification. This software has the following criteria:

1. Does not generate any Class 1E information.
2. Has no impact on the Class 1E function.
3. Has no direct electrical path to erroneously alter a Class 1E function or its data.

5.4.4. Application of the Verification Matrix and Criteria Utilized for Software Testing for the Eagle-21 Replacement Hardware

- 5.4.4.1 The Eagle-21 Replacement system can be divided into two groups: 1) that which performs Class 1E protection functions, has impact on Class 1E functions, and which tests Class 1E functions and 2) that which monitors the system and provides non-class 1E information to the user.

The first group consists of the following (Reference Figure 1):

1. All of the Loop Processor Subsystem
2. The portion of the Tester Subsystem that runs surveillance tests and therefore, has an impact on the I/O modules
3. That portion of the Tester Subsystem which controls communication to the Loop Processor for parameter update.

4. That portion of the MMI cart which allows the operator to input new parameters and which does the limit checking on those inputs.

This group, which meets the criteria for Section 5.4.3.1, will be verified at level 1 to give the highest degree of confidence to this code.

The second group consists of the following (Reference Figure 1):

1. That portion of the Tester Subsystem which has no direct link to the Loop Processor other than a read-only datalink. This includes the software which updates the test panel lights and outputs analog trend points.
2. All of the MMI software except that listed in 4) above.

This group will be verified at level 2 since it meets the criteria of section 5.4.3.2.

5.4.4.2 Criteria Utilized for Software Testing

Past experience has demonstrated that emulation testing of very simple procedures is not necessary and that the resources spent testing these procedures could be better applied to the larger, more error-prone code. The following are the criteria used to determine if a procedure can be classified as "simple" and subject to a strict source code review as opposed to testing. If any one of the following statements is true, testing will be performed as defined in Section 5.3.2.1.

1. The verifier determines that this particular procedure is a unique case and, while all other conditions are satisfied, a code and documentation review is not adequate and that testing should still be performed.
2. Math operations (+, -, /) are done by this procedure and involve at least one variable that is not ROM based and is not a data constant.
3. Logical operations are done by this procedure and the result is used in a manner other than as an ordinary TRUE/FALSE or where the resulting logical byte is NOT accessed according to the definitions:
TRUE equates to (0=0)
FALSE equates to (0=1)

4. Logical operations are done by this procedure for the purpose of setting or clearing (masking) status or control bits.
5. There is more than one path to the procedure due to the use of one or more of the following PLM control statements:
DO-CASE
DO-WHILE
ITERATIVE DO BLOCKS (DO counter = start TO end)
IF-THEN
GO TO
6. The procedure consists of more than twenty executable statements. The term "executable statement" is defined as any statement other than the procedure declare, procedure end, or comments.
7. The procedure includes one or more internal procedures.

6.0 SYSTEM VALIDATION

6.1 Validation Philosophy

Whereas the system verification process verifies the decomposition of the system requirement documents in the definition and design stage and also verifies the functionality of the software entities (unit, module, and subprogram) beginning from the smallest software entity and progressing to the program level, the system validation process is performed to demonstrate the system functionality. By conducting the system validation test, the results demonstrate that the system design meets the system functional requirements. Hence, any inconsistencies that occurred during the system development, in this area, that were not discovered during the various design verification activities discussed in Section 5.0, would indeed be reviewed, identified, and tracked by the verifiers through resolution by the design team.

Following completion of the system validation test, the user can indeed have a high degree of confidence that the system functional requirements are met.

6.2 Validation Testing Overview

During verification, a bottom-up microscopic approach is utilized to thoroughly and individually review and/or test each piece of software within the total system. This requires a significant effort and verifies that each software element operates properly as a stand-alone entity.

Validation complements the verification process and not only ensures that the final implemented system satisfies the top-level

functional requirements but also that good engineering practice was utilized during the design and implementation of the system. Following are the major phases of validation:

- * Top-down functional requirements testing
- * Prudency review of the design and its implementation
- * Specific Man-Machine Interface (MMI) testing

The macroscopic top-down functional requirements phase of validation testing treats the system as a black box while the prudency review phase requires that the internal structure of the integrated software/hardware system be analyzed in great detail. Due to this dual approach, validation testing provides a level of thoroughness and testing accuracy which is at least equivalent to that which occurs during verification and insures detection of any deficiencies that occurred during the design process but not discovered during verification. Validation testing is performed on the verified software residing within the final target hardware.

6.2.1 General Description

The Validation plan defines a methodology that must be followed to perform a series of top-down functional requirement based reviews and tests which compliment the bottom-up approach utilized during the Verification testing phase.

Four independent types of reviews and/or tests are to be conducted to insure over-all system integrity:

1. Functional Requirements Testing - this insures that the design meets the functional requirements.
2. Abnormal-mode Testing - this insures that the design operates properly under abnormal-mode conditions.
3. System Prudency Review/Testing - this ensures that good design practice was utilized in the design and implementation of critical areas of the system. The items covered within this section require the internals of the system design and implementation to be analyzed in detail.
4. Specific Man-Machine Interface testing - this ensures that the operator interface utilized to modify the system's data-base performs properly under normal-mode and abnormal-mode data-entry sequences. This is a critical area requiring special attention due to the impact on the software of the system-level information which can be modified via this interface.

The functional requirements and abnormal-mode testing phases of Validation utilize a black-box systems approach while the System Prudency Review/Testing phase emphasizes the need to understand the internal operations and interactions within the system.

6.2.2 Functional Requirements Testing

The Validation functional requirements testing phase consists of the following steps:

1. Functional requirements decomposition

The top-level functional requirements must be decomposed into detailed sub-requirements. For each sub-requirement, a test or a series of tests must be identified and performed to insure that the specific sub-requirement is satisfied.

Some sub-requirements are fairly general so it is important that the same individual that performs the decomposition also provides the interpretation as to the type of test which must be executed to insure that the sub-requirement is met.

2. Validation test procedure generation

Once the decomposition has occurred, the specifics of the test(s) must be defined in test procedural form such that it (they) can be conducted during validation testing.

3. Validation test execution (Refer to Section 7.3)

The detailed tests per the Validation test procedures must be conducted by a Validation Test Technician and the results must be reviewed by the Validation Test Engineer.

Each functional sub-requirement must be uniquely identified. The test procedure generated to test each sub-requirement must be coorespondingly identified for ease of cross-referencing.

6.2.3 Abnormal-Mode Testing

During this phase of Validation the functional requirements are reviewed to define a series of abnormal conditions underwhich the system must operate properly, without resulting in, or causing, any inadvertent or detrimental actions.

The Validation abnormal-mode testing phase consists of the following steps:

1. Functional requirements decomposition

The top-level functional requirements must be reviewed to identify detailed abnormal-mode conditions. The type of test that must be conducted to exercise the system under each abnormal-mode condition must also be defined.

2. Validation test procedure generation

Once the decomposition has occurred, the specifics of the test(s) must be defined in test procedural form such that it (they) can be conducted during Validation testing.

3. Validation test execution (Refer to Section 7.3)

The detailed tests per the test procedures must be conducted by a Validation Test Technician and the results must be reviewed by the Validation Test Engineer.

Each abnormal-mode condition must be uniquely identified. The test procedure generated to test each sub-requirement must be correspondingly identified for ease of cross-referencing.

6.2.4 System Prudency Review/Testing

During this phase of Validation, the system design and implementation is analyzed and reviewed against the "System Prudency Checklist". The system must be evaluated against this checklist to insure that good engineering practice has been followed.

The System Prudency Checklist addresses the following critical design areas:

- * Firmware program storage
- * Data-base information storage
- * Multiple-processor shared memory architectures
- * Data-link oriented system architectures
- * Diagnostics
- * System time synchronization

Most of these items do not relate directly to a functional requirement or to a series of functional requirements but address the issue of integrated system integrity.

7.0 DEVELOPMENT, VERIFICATION AND VALIDATION ORGANIZATION

During the system design process, two independent functions will be utilized: one for development, and one for verification. The software development personnel receive the System Design Specification, generate the Software Design Specifications, and then designs, develops, tests, and documents the code. The verification personnel receive the released code and its documentation, performs the required reviews and tests as dictated by the Software Verification Level within the Verification Matrix and produces a Verification Test Report (VTR).

This type of organization has several advantages. The use of two independent entities introduces diversity to the process of software generation and reduces the probability of undetected errors. Another benefit is that such a scheme forces the designer to produce sufficient and unambiguous documentation before verification can take place.

Functional independence is essential to achieve these goals. In particular, the two functions will have separate lead engineers. Note that the development personnel submits the code for verification only after the development team has confirmed the code to its satisfaction. Errors discovered (debugging) during the development phase testing are not required to be documented by the verification engineers.

The use of the above procedures does not preclude the possibility that the developer of one module may be the verifier of a different module, as long as that person did not participate in the design or coding of the module being verified.

7.1 Development Activity

The composition of the development team is dependent upon the functions that are required to be performed by the team. Typical team functions include the following:

7.1.1 Chief Programmer

This is the team software leader who is responsible for the software technical matters. The duties of the Chief Programmer include:

a. Software Design Specification

The chief programmer has the responsibility for the development of the Software Design Specifications, which are based on the System Design Specification.

b. Architecture

Global decisions on the structure of the software, decomposition and data base are made by the chief programmer.

c. Coding

Some critical sections of the programs (both in terms of importance and complexity) can be coded by the chief programmer.

d. General

The chief programmer supervises the rest of the team in software technical matters.

7.1.2 Programmers

It is anticipated that there will be more than one programmer, and that at least one programmer will function as a back-up to the chief programmer. The programmers' tasks are to develop the code for modules and/or sub-systems as directed by the Software Design Specifications.

7.2 Verification Activity

The functions of the verification team are as follows:

7.2.1 Chief Verifier

Team leader who is responsible for all technical matters. The duties of the Chief Verifier include:

- a. Review System Design Requirements and Specifications received from the development engineer for completeness and unambiguity. (This review may be performed by another qualified individual who is independent of the design area being reviewed.)
- b. Review the Software Design Specifications received from the development engineer for completeness and unambiguity.
- c. Review verifier's Software Test Specifications for completeness.
- d. Oversee verification of critical sections in the software.
- e. Supervise and consult with the verification team.
- f. Review Test Reports

7.2.2 Verifiers

- a. Perform source code inspections and review Software Design Specifications.
- b. Write Software Test Specifications.
- c. Run tests on subprograms, modules and units.
- d. Write test reports.

7.2.3 Librarian Function

The Librarian performs the following duties in the maintenance of the Verification Software Library:

- a. Responsible for the storage and configuration control of the computer software being verified as follows:
 - (1) Establishes identification of each software element (i.e. unit, module, subprogram) within the Computer Software Baseline (CSB)
 - (2) Enforces procedures for software and documentation changes during reverification effort
 - (3) Maintains configuration control of the current CSB
- b. Controls the transmittal of computer software to authorized personnel only
- c. Ensures no unauthorized changes occur to the CSB

7.3 Validation Function

The functions of the Validators are as follows:

7.3.1 Chief Verifier

- a. Coordinate total Validation program
- b. Review Validation testing results and write final report
- c. Supervise and consult with the validators

7.3.2 Functional Requirements Decomposer (optional/Chief Verifier)

- a. Coordinate Validation of a specific area
- b. Review functional decomposition for completeness and accuracy (this review may be performed by another qualified individual who is independent of the design area being reviewed)

7.3.3 Lead Validator (optional/Chief Verifier)

- a. Coordinate Validation of a specific area
- b. Review functional decomposition for completeness and accuracy (this review may be performed by another qualified individual who is independent of the design area being reviewed)
- c. Review and approve test procedure vs functional requirement test specification to insure test procedure is adequate

- d. Along with the Librarian, insure that proper verified code is being validated

7.3.4 Validation Test Engineer

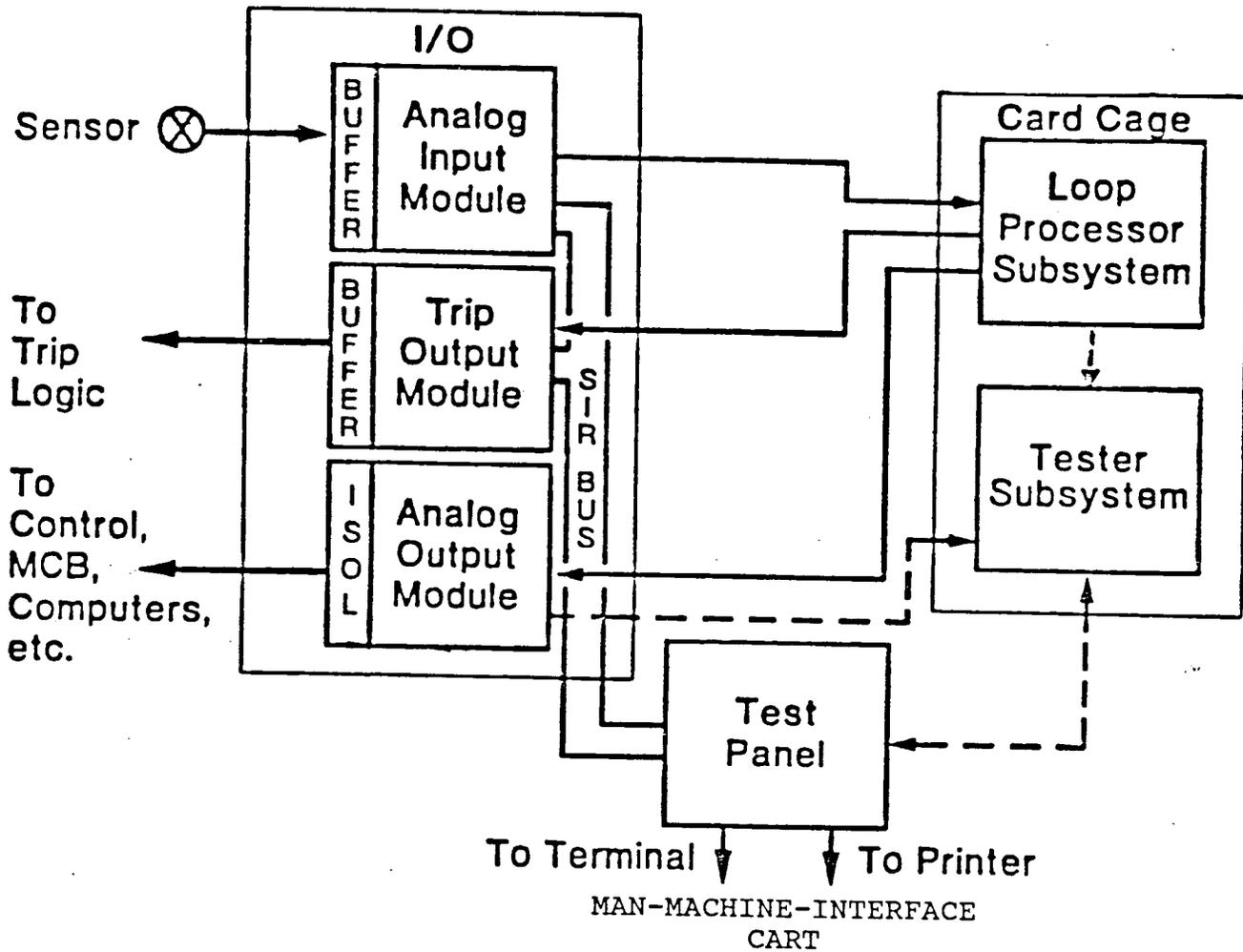
- a. Write Validation test procedures
- b. Oversee Validation testing and review test results
- c. Generate Validation Trouble Reports

7.3.5 Librarian

- a. Coordinate with the Chief Verifier/Lead Validator(s) and/or Validation test Engineers to insure that proper verified code is being validated.
- b. Coordinate dissemination of Validation trouble reports to the appropriate design engineer.

7.3.6 Validation Test Technician

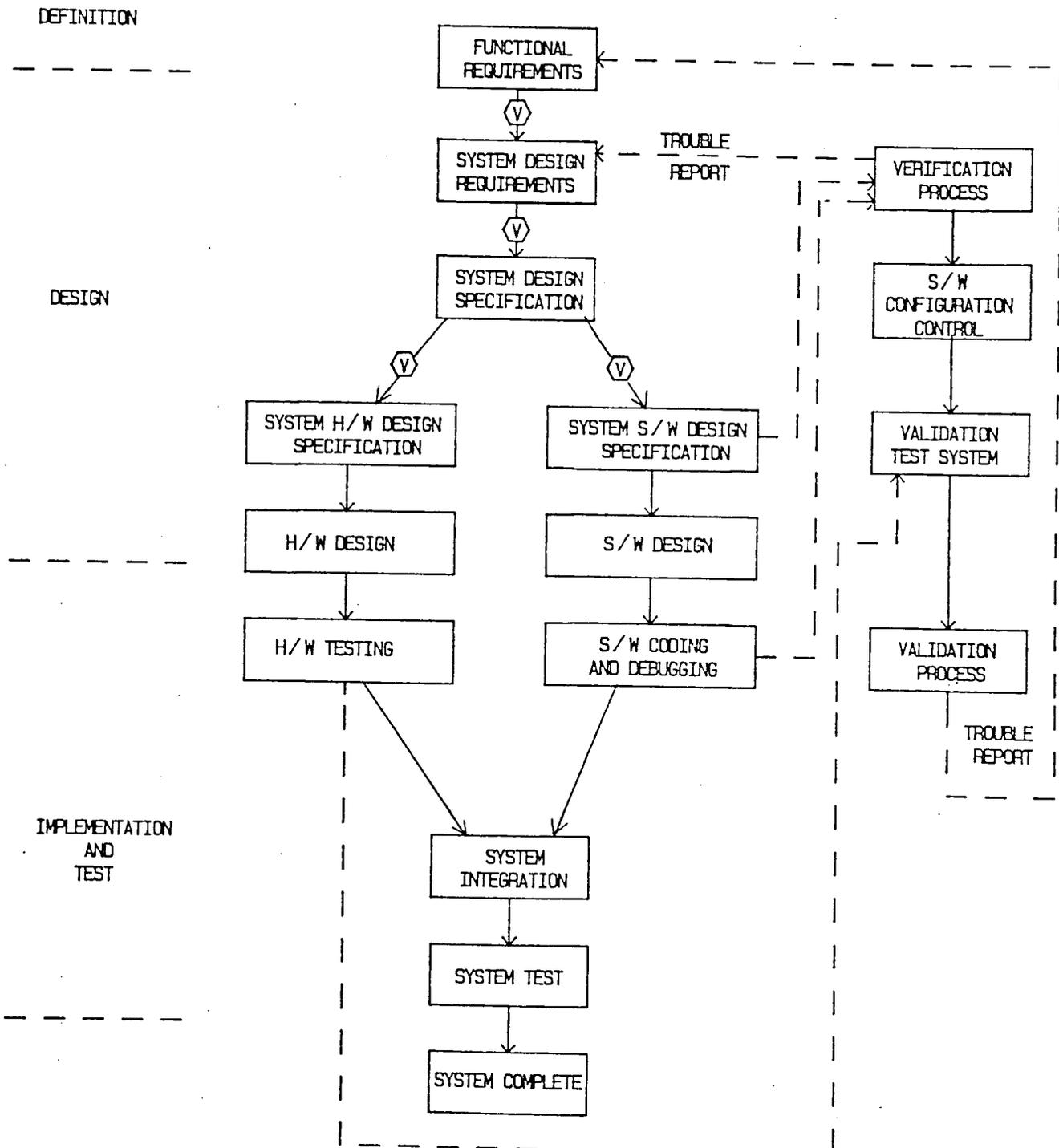
- a. Perform Validation tests under direction of the Validation Test Engineer
- b. Document test results



EAGLE - 21
 PROCESS PROTECTION SYSTEM
 ARCHITECTURE

FIGURE 1

FIGURE 2
DESIGN, VERIFICATION AND VALIDATION PROCESS



⬡ DENOTES INDEPENDENT VERIFICATION REVIEW

DESIGN DOCUMENT	COMPONENT	VERIFICATION		
		UNIT	MODULE	SUBPROGRAM
SOFTWARE DESIGN SPEC	SUBPROGRAM			↕
	----- MODULE		↕	↕
	----- UNIT	↕	↕	

SOFTWARE VERIFICATION PROCESS

FIGURE 3

SOFTWARE VERIFICATION LEVEL	SOFTWARE COMPONENT			
	UNIT	MODULE	SUBPROGRAM	
* NOTE 1 1	ST	ST	ST	INDEPENDENT TESTING
2	FTR	FTR	FTR	↑ SEPARATE REVIEW ↓

ST = STRUCTURAL TESTING
 FTR = FUNCTIONAL TEST REVIEW

* NOTE 1: REFERENCE SUPPLEMENTAL CRITERIA IN SECTION 5.4.4.2

SOFTWARE VERIFICATION MATRIX
 FIGURE 4

ENCLOSURE 2

V&V PLAN CLARIFICATION

EAGLE 21 DESIGN, VERIFICATION AND VALIDATION PLAN

OVERVIEW

This document is intended to clarify the EAGLE 21, Design Verification and Validation Plan; reference Design Specification # 408A47. Specifically, the objective of this document is to expand and clarify section 5.4.4.2 "Criteria Utilized for Software Testing," within the draft document. Upon review, the NRC has taken exception to this section of the document. Specific reference has been made that section 5.4.4.2 of the V&V Plan is not in compliance with ANSI Standard 7-4.3.2., Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations.

The testing criteria, submitted within the document, are valid and fully meet the intent of ANSI Standard 7-4.3.2. Successful resolution of this item can be achieved through a more complete discussion relating to the original section. For the purpose of minimizing document cross-reference, section 5.4.4.2 of the draft has been reproduced in its entirety as Appendix A in this clarification.

BACKGROUND

Clarification of terms: Appendix A of this clarification reproduces the entire Section 5.4.4.2 from the draft document. This section begins with the text: "Past experience has demonstrated that actual testing of very simple procedures is not necessary.... Review of this text, based on subsequent comments, leads us to believe that the primary intent of this statement has not been effectively communicated. The phrase "actual testing" refers only to emulation of the program code on a VAX computer. With this in mind, the original statement can more clearly be stated as: "Past experience has demonstrated that emulation testing for very simple procedures is not necessary. Clarification of this phrase must also convey two very important points:

- Simple procedures are not being excluded from software verification testing. They will still receive both a strict documentation and source-code review.
- Verification testing will be performed on all software units, but some simple procedures [meeting a stated criteria] do not require emulation on a computer.

Previous verification experience: Over 2,000 software verification tests have been performed by Westinghouse in other activities similar to the EAGLE 21 V&V Plan. During this process it has become readily apparent that several types of "simple" procedures need not be emulated on a computer as part of their software verification testing. Several specific examples of these uncomplicated software procedures have been included for review as Appendix B to this clarification. In general, these procedures include those which initialize I/O ports, set variable(s) to a logical [i.e. TRUE or FALSE], or initialize variables to a fixed value.

After emulating many procedures on a VAX computer, it has become obvious that emulating code-function for procedures of these types is a meaningless exercise. Today's compilers certainly have the ability to set (initialize) a given variable to any value. Having to compile, link, locate, and emulate codes for procedures that only perform this function, adds no additional quality to the software verification process. Further, computer emulation of software units which initialize I/O ports has no meaning. The actual hardware port must be used. Proper functioning of all I/O ports is already thoroughly encompassed within the existing validation testing requirements.

Establishment of testing criteria: In order not to subjectively decide which software procedures may be considered "simple", a Software Testing Criteria was developed. This criteria is presented in section 5.4.4.2 of the V&V Plan submittal. Establishment of this testing criteria firmly provides strict guidelines which must be followed by every software verifier. In this way, a clear audit-trail is maintained within the entire software verification process. The criteria clearly indicates that any software procedure which performs math operations, uses logical operations in other than a pre-defined manner, performs any PLM control statement, has 20 or more lines of code, uses an internal procedure, or is judged as a unique case by the software verifier, must be emulated on a computer.

SUMMARY

Levels of verification testing: Verification will be performed on all software procedures. Some procedures which meet the previously stated software testing criteria will not require emulation on a computer due to their simplicity. However, these units must still undergo a strict review of their documentation and source code.

Compliance with existing standards: The EAGLE 21 Design, Verification and Validation Plan meets the intent of ANSI Standard 7-4.3.2. Section 7.3.1 of that document indicates that criteria for establishing test cases must be specified. This is done in section 5.4.4.2 of the submittal. In addition, the first paragraph from Section 7 (ANSI Standard 7-4.3.2) allows one to "...draw from previous experience and supplementary information.". The approach is consistent with these guidelines.

APPENDIX A

Text reproduced from:

EAGLE 21 DESIGN, VERIFICATION AND VALIDATION PLAN

5.4.4.2 Criteria Utilized for Software Testing

Past experience has demonstrated that actual testing of very simple procedures is not necessary and that the resources spent testing these procedures could better be applied to the larger, more error-prone code. The following are the criteria used to determine if a procedure can be classified as "simple" and subject to a strict source code review as apposed to testing. If any one of the following statements is true, testing will be performed as defined in Section 5.3.2.1.

1. The verifier determines that this particular procedure is a unique case and, while all other conditions are satisfied, a code and documentation review is not adequate and that testing should still be performed.
2. Math operations (+,-,/) are done by this procedure and involve at least one variable that is not ROM based and is not a constant.
3. Logical operations are done by this procedure and the result is used in a manner other than as an ordinary TRUE/FALSE or where the resulting logical byte is NOT accessed according to the definitions:

TRUE equates to (0=0)
FALSE equates to (0=1)
4. Logical operations are done by this procedure for the purpose of setting or clearing (masking) status or control bits.
5. There is more than one path to the procedure due to the use of one or more of the following PLM control statements: DO-CASE, DO-WHILE, INTERATIVE DO BLOCKS (DO counter = start To end), IF-THEN, GO TO.
6. The procedure consists of more than 20 executable statements. The term "executable statement" is defined as any statement other than the procedure declare, procedure end, or comments.
7. The procedure includes one or more internal procedures.

APPENDIX B

Examples of uncomplicated software procedures;

```
EU_TYPE_14:PROCEDURE(I) public;
  DECLARE I byte;
  I = 0;
END EU_TYPE_14;
```

```
copy_ai_app:PROCEDURE (lo,hi) PUBLIC;
  DECLARE (lo,hi) byte;
  return;
END copy_ai_app;
```

```
BEFORE$THE$DECIMAL:PROCEDURE(DIGIT$PLAC);
  DECLARE Digits$Plac byte;
  Digit$Plac = 0;
END BEFORE$THE$DECIMAL;
```

```
DEADMAN:PROCEDURE PUBLIC;
  /* FOR DEADMAN TIMER ON WESTINGHOUSE SELF HEALTH
  BOARD */
  DEADMAN_PORT = (11111111B);
END DEADMAN;
```

```
EXP$CHECK:PROCEDURE(POINTR);
  DECLARE POINTR BYTE;
  Pointr = 0;
END EXP$CHECK;
```