DESIGN, VERIFICATION, AND VALIDATION PLAN

FOR THE

SOUTH TEXAS PROJECT

QUALIFIED DISPLAY PROCESSING SYSTEM

Design Specification Number 955842

Rev. 4

TABLE OF CONTENTS

process variable characteristics that must be displayed. Examples of variable characteristics determined by the review of the guidelines include the following: value, range, prediction, trend, and pattern recognition. Grouping the variables and factoring in the location of the QDPS displays on the control board determined the structure of the displays that needed to be developed. Many iterations occurred with the South Texas Operations Department during this development process. Utilizing the plant ERGs and the input from operations personnel ensures that the QDPS will present the data in a clear and concise form.

1.2.2 Qualified Control

The second subsystem, Qualified Control, performs the valve control functions, including control for the steam generator Power Operated Relief Valves (PORV), the Auxiliary Feedwater Throttle Valves, the Essential Cooling Water Flow, and the Reactor Vessel Head Vent Valves. The control units will be channel oriented in exactly the same way as the RPUs and will share the same cabinets with the RPUs. However, operation of the control unit will be independent of the RPU.

1.2.3 Steam Generator Water Level Compensation/Temperature Averaging Scheme

The third subsystem includes the Steam Generator Water Level Compensation Subsystem (SGWLCS) and the Temperature Averaging Scheme (TAS). The SGWLCS temperature compensates steam generator narrow range water level channels for level induced errors as a result of changes in reference leg temperature. The TAS calculates the loop average hot leg temperature from the three narrow range fast response RTD's per loop. The SGWLCS/TAS subsystems are housed in the same cabinets with the RPUs and the Qualified Control units. Like the Qualified Control subsystem, the operation of SGWLCS/TAS are independent of the RPU.

1.3 System Architecture

A block diagram of the QDPS system is shown in Figure 1. Each of the hardware units depicted contains one or more microprocessors. Each Auxiliary Process Cabinet (APC) shown on the left of Figure 1 contains an RPU, a Qualified Control unit and a SGWLCS/TAS unit as shown in Figure 2. Figures 1 and 2 illustrate the various inputs, outputs and data communication paths associated with the QDPS subsystems.

Response Facility computer System. RPU communication is via isolated, redundant RS-422 data links.

The Database Processing Unit (DPU) is an intelligent microprocessor based unit that receives isolated data link inputs from each RPU and uses the data to provide:

- Analog outputs to drive conventional indicators and recorders

- Contact outputs to provide qualified status information

- Datalink outputs to drive the display modules and recorder demultiplexer

The DPU performs redundant sensor algorithms and other necessary calculations. Each RPU transmits its data to both DPUs in the system. Therefore, each DPU maintains the entire database. Either DPU can provide the operator with the best information available from the entire data base. Comparison of the data between DPUs prevents the possibility of erroneous display information.

The display modules are qualified graphic/alphanumeric devices which provide comprehensive displays without requiring large amounts of control board space. Each display module interfaces to both Database Processing Units to provide full redundancy and meet single failure requirements. The display itself is a 512 by 512 dot matrix plasma display (with an active area of approximately 8.5 inches square) providing a flat screen, extreme ruggedness, and easy to read orange-on- black images that are flicker free.

The display module is human engineered for ease of operation and provides functional pushbuttons for individual display selections. The functional keys allow the operator to move easily from one page to another to display specific information.

During the design of the graphic display pages, South Texas Nuclear Plant Operations personnel have been involved in the establishment of criteria and mimic design. A checklist was developed that was based upon a review of the Westinghouse Owner's Group Emergency Response Guidelines (ERG's) which stipulated the plant process variable characteristics that must be displayed. Examples of variable characteristics determined by the review of the guidelines include the following: value, range, prediction, trend, and pattern recognition. Grouping the variables and factoring in the location of the QDPS displays on the control board determined the structure of the displays that needed to be developed. Many iterations occurred with the South Texas Operations Department during this development process. Utilizing the plant ERGs and the input from operations personnel ensures that the QDPS will present the data in a clear and concise form.

1.2.2 Qualified Control

The second subsystem, Qualified Control, performs the valve control functions, including control for the steam generator Power Operated Relief Valves (PORV), the Auxiliary Feedwater Throttle Valves, and the Reactor Vessel Head Vent Valves. The control units will be channel oriented in exactly the same way as the RPUs and will share the same cabinets with the RPUs. However, operation of the control unit will be independent of the RPU.

1.2.3 Steam Generator Water Level Compensation/Temperature Averaging Scheme

The third subsystem includes the Steam Generator Water Level Compensation Subsystem (SGWLCS) and the Temperature Averaging Scheme (TAS). The SGWLCS temperature compensates steam generator narrow range water level channels for level induced errors as a result of changes in reference leg temperature. The TAS calculates the loop average hot leg temperature from the three narrow range fast response RTD's per loop. The SGWLCS/TAS subsystems are housed in the same cabinets with the RPUs and the Qualified Control units. Like the Qualified Control subsystem, the operation of SGWLCS/TAS are independent of the RPU.

1.3 System Architecture

A block diagram of the QDPS system is shown in Figure 1. Each of the hardware units depicted contains one or more microprocessors. Each Auxiliary Process Cabinet (APC) shown on the left of Figure 1 contains an RPU, a Qualified Control unit and a SGWLCS/TAS unit as shown in Figure 2. Figures 1 and 2 illustrate the various inputs, outputs and data communication paths associated with the QDPS subsystems.

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"

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.

DEVELOPMENT TEAM - A team of individuals or an individual assigned to design, develop and document software as outlined in this plan. Generally it comprises team coordinator, programmers and a librarian.

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 as specified in the design requirements.

FUNCTIONAL TEST REVIEW (FTR) - A review which is performed on the documented functional tests that were run by the developer 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, violations of development standards, and other problems.

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

MODULE (M) - Refers to a significant partial functional capability of a subprogram. 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.

SOFTWARE DESIGN SPECIFICATION (SDS) - A document which represents the designers' 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 TEST SPECIFICATION (STS) - A document detailing the tests to be performed, test environment, acceptance criteria and the test methodology. Approved receipt of the SDS document by the chief verifier 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.

VERIFICATION TEAM - A team of individuals or an individual assigned to review source code, generate test plans, run tests and document the test results for a computer system. It is comprised of verifiers led by a chief verifier.

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 QDPS, as shown in Figure 3, involves four stages:

1. Definition
2. Design
3. Implementation
4. System Integration and Testing

A brief description of each stage is given below:

The definition stage is characterized by the statement of the problem to to be solved, 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 Specification.

The design stage is characterized by the decomposition of these System Design Specifications into Hardware Design Specifications 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.

The implementation stage is characterized by the actual construction of the hardware and the coding of the various software entities. 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 officially turned over to the verification team for final independent review and/or testing as specified in Section 7.0.

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 the 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 engineer 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 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.

The system integration and testing stage is where the various hardware components and software entities are assembled in a stepwise manner with additional testing at each step to ensure that each component performs its required function when integrated with its associated components.

The final activity associated with the system integration and testing stage is the testing of each subsystem and final testing of the QDPS. A system test plan is derived from the system functional requirements and System Design Specifications to confirm that the QDPS 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 multidisciplineary engineers to ensure that all aspects of the design are reviewed.

During the implementation 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 conduct 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 commensurate 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 team uses for determining the level of verification that should be applied to each software entity. The section concludes by defining the application of the verification matrix to the QDPS.

5.2 Verification Philosophy

Figure 4 illustrates the integration of the system verification and validation process with the system design process. As discussed in Section 4.0 during the implementation 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 officially turned over to the verification team. At this point the verification team performs an independent review and/or test of the software entities to verify that the functionality of the software entities meet the applicable Software Design Specifications. After the verification team is satisfied that all requirements are met, the software is configured for use in the final system and subsequent system validation process.

Figure 5 illustrates the philosophy utilized in conducting the 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 verification team proceeds to verify that module. Not only is the software module verified to meet the module Software Design Specification, but the verification team 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 verification team proceeds to verify that subprogram. As in the case of the software module, the verification team not only verifies that the subprogram meets the applicable Software Design Specifications, but the team verifies that the appropriate software modules were utilized in generating the subprogram entity. This verification philosophy ensures that the verification team tests and/or reviews the interface between the software unit, module and subprogram entities.

Depending upon the hardware implementation, the verification process may utilize the actual target hardware or a test jig that accepts target software in the verification of the software modules and subsystems.

5.3 Verification

Verification techniques used in software development fall into two basic categories: review and testing.

5.3.1 Review

There are three types of reviews used in the verification of software: Design Specification reviews, code reviews and functional test reviews.

5.3.1.1 Design Specification Review

This activity involves the comparison of a Software Design Specification for a subsystem, module, or unit to the Design Specification of the component above it to ensure that all of the performance requirements stated in the higher level document are met.

5.3.1.2 Source Code Review

Source code review, as opposed to code testing, is a verification method in which the software 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. The expectation is that most of the errors will be discovered and corrected at this level, where the cost of doing so will be minimal.

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 branches within the software entity. If this is not possible, the test inputs should be chosen to exercise every statement within the software entity. 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 some 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.

5.4 Verification Matrix

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

1. The safety classification of the system

2. The demonstrability of the system functions : Visual or non-Visual

3. 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 Demonstrability of System Functions

The method of testing for a software item depends upon the visual demonstrability of its function within the system. If the system function can be illustrated by a simple model as in Figure 6, then the method of testing required need not be as comprehensive as that for software items whose functions cannot be visually recognized. The verification team will be responsible for determining whether a component has a visual or non-visual function.

5.4.3 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 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.

At the system level, subtle errors resulting from the complex interaction between pieces of software never before interconnected may be exposed. The deterrent to any major problems in this phase of testing is thorough and comprehensive verification at lower levels.

5.4.4 Justification of Matrix Elements

Considering the parameters detailed above, different verification methods are required for different subsystems and software components. Figure 7 illustrates, in tabular form, the levels of verification being proposed for various systems. The safety classification column identifies different categories of subsystems in relation to their safety classification. The software component columns identify the levels of software found within each system. Each element of the matrix specifies the type of testing or review that will be performed on the software component within that safety classification. The justification of each matrix element follows.

The software associated with actuation and/or implementation of reactor trip and engineered safeguards (IEEE-279-1971) must receive the highest level of verification identified. As such, all software at the unit level must be structurally tested to ensure that all lines of the unit indeed meet the intended design specification. Since the control room operators rely upon the automatic actuation of the plant reactor trips and/or engineered safeguards actuations, the highest level of confidence must be afforded the operator. Examples of the South Texas Project system application that meets this definition are the SGWLCS/TAS, the Essential Cooling Water Flow control loop, and the auxiliary feedwater throttle valve qualified control loop.

Depending upon the demonstrability of the identified module and subprogram segments, these segments must either be structurally or functionally tested. Generally, if the module level becomes too complicated, one or more additional unit segments will be identified, which incorporate many of the operations included in the original complicated module or subprogram segment. Therefore, the subdivision will be designed to maximize the demonstrability of the module and subprogram segments. If the verifiers are successful in this task, the module and subprogram tasks are only required to be functionally tested. If the module and/or subprogram segments are visual, a functional test provides essentially the same level of verification as a structural test. Furthermore, for the verifiers to make an appropriate decision concerning the demonstrability of the module and/or subprogram, a detailed source code review (SCR) must be conducted which necessitates a thorough review of each line of code.

Regulatory Guide 1.97, Rev. 2 Type A, B and C Category 1 variables are defined as those key variables which are necessary for the operator to: (1) diagnose the accident; (2) take the preplanned manually controlled actions for which no automatic control is provided; (3) take the plant to a safe shutdown condition; (4) monitor the status of the plant critical safety functions; and (5) monitor the potential for breach of a fission product barrier. These variables are identified as only those necessary for the operator to achieve and maintain the plant in a safe shutdown condition. Even though no automatic actuation of plant protective functions results from these variables, the control room operators use these variables to initiate the appropriate manual actions (based upon the Emergency Operating Procedures) in order to achieve and maintain a safe shutdown condition. For example, RWST to containment sump switchover; isolation of auxiliary feedwater flow to one or more steam generators following a secondary high energy line rupture; and switchover to alternate secondary source of water.

Because of the importance of these variables to the operator for taking appropriate manual actions, the same level of verification is required for R.G. 1.97 Type A, B and C Category 1 variables (IEEE-603 1980, Paragraph 5.8.1) as for those associated with automatic actuations. That is, the unit level must be structurally tested and the module and subprogram segments must be either structurally or functionally tested, dependent upon the level of demonstrability.

The next category of software that must be addressed is that associated with the qualified control functions of head vent valve control and steam generator PORV valve control. Even though the hardware associated with these functions is qualified, meets the single failure criteria and is Class 1E powered, the operator is not required to implement these functions to mitigate a design basis accident. The control room operators have the option of maintaining the plant in a hot shutdown or hot standby condition until conditions in the plant warrant a normal plant cooldown sequence. As such, the level of verification of the software associated with these functions is relaxed when compared to those discussed above. The unit level of the qualified control function must either be structurally or functionally tested, dependant upon the demonstrability of the software. If the software is not visual, the unit level software must be structurally tested. However, if the software is visual, functional testing will be adequate. Note again that a thorough SCR must be conducted on the software to determine the level of demonstrability. With a thorough review and/or testing of the unit level software, the verification required on the module and subprogram segments is merely functional testing.

Observe that for all three software functions associated with IEEE-279 1971; IEEE-603 1980, Paragraph 5.8.1; and qualified control,the level of verification at the software unit level is greater than or equal to the verification performed at the module level. Furthermore, the level of verification at the software module level is greater than or equal to the verification performed at the subprogram level. Because of the safety significance of these plant functions, the verifiers must ensure that the unit level software is tested thoroughly to provide the highest level of confidence to the plant operating staff that these safety grade systems will work as specified in the system design document.

The level of verification for the Regulatory Guide 1.97, Rev. 2 Type D Category 2 variables is also specified (IEEE-603 1980, Paragraph 5.8.2). These variables are defined as those that (1) monitor the status of plant safety systems and; (2) monitor the status of those plant systems normally used to achieve a safe plant shutdown. The Type D Category 2 variables are not required to be monitored by the plant operating staff to achieve a safe shutdown condition. Furthermore, these variables are not required to be redundant, or Class 1E powered. However, some of the Type D Category 2 variables are Class 1E powered, but not redundant. Hence, the level of verification required for the software associated with the

Type D Category 2 variables is made a function of the power supply pedigree, but for either case, significantly relaxed. For those Type D Category 2 variables that are Class 1E powered, the subprogram, module and unit software levels need only be functionally tested. However, for the Type D Category 2 variables that are non-Class 1E powered, all three software levels need only receive a separate review. Depending upon the demonstrability of the unit and module levels, the software must either be source code reviewed or functional test reviewed. The subprogram level must only have a functional test review.

5.4.5 Application of the Matrix to the QDPS

To understand the verification levels to be used on the QDPS, a discussion of the architecture and functions of the system is in order. An overall block diagram of the system is presented in Figure 1. Each box on the figure, except for the APC's, represents a digital subsystem containing one or more subprograms. Within each box is the indicated level of verification to be performed on that subsystem. The levels of verification have been assigned in accordance with the highest safety classification of the functions performed by each of the boxes. For example, a Database Processing Unit (DPU) is a Class 1E cabinet which processes information to meet the requirements of Regulatory Guide 1.97, Rev. 2 Categories 1 and 2. Since a DPU processes Category 1 variables, it has been assigned a verification level 1. Similarly, since Remote Processing Unit (RPU) N is a non-1E cabinet which processes only non-1E Category 2 variables, it has been assigned a verification level 4.

Since each Auxiliary Process Cabinet (APC) contains three subsystems, an additional block diagram of an APC has been included in Figure 2. The RPU, which processes Regulatory Guide 1.97 Category 1 variables, has been assigned a verification level 1. Likewise, SGWLCS/TAS, which must meet the requirements of IEEE-279 1971, must also be verified to the level 1 requirements. The qualified control system, which processes three control loops, has also been assigned a verification level of 1 since one of the loops it controls (Auxiliary Feedwater Control) must also meet the requirements of IEEE-279 1971.

6.0 SYSTEM VALIDATION

6.1 Validation Philosophy

Whereas the system verification process verifies the functionality of the software entities 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 testing results demonstrate that the system design meets the system functional requirements. Hence, any inconsistencies that occurred during the systems development in this area that were not discovered during the software verification activities discussed in section 4 would be identified through the validation process.

6.2 Validation Testing Overview

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

Validation compliments the verification process by ensuring that the system meets it functional requirements by conducting top- down testing, first from the subsystem level and then for the integrated system. This is illustrated in figure 5.

The major phases of the validation process include the following:

a. Top-down functional requirements testing
b. Prudency review of the design and implementation
c. 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 the dual approach, validation testing provides a level of thoroughness and testing accuracy which enhances the possibility of detection of any deficiencies that occurred during the design process but not discovered during verification Validation is performed on verified software residing within the final target hardware or test jig as shown in figure 4.

The validation process utilizes a methodology that defines a series of top-down functional requirement reviews and tests which compliment the bottom-up approach utilized during the verification testing phase.

a. Functional requirements testing - ensures that the final system meets the functional requirements. A comprehensive functional requirements decomposition is conducted on all system functional requirements from which the validation test requirements originated.

b. Abnormal mode testing - ensures that the design operates properly under abnormal operating conditions.

c. System Prudency Review/Testing - ensures that good design practice was utilized in the design and implementation of critical design areas of the system. These tests require that the internals of the system design and implementation be analyzed in detail.

d. Specific Man-Machine Interface testing - ensures that the operator interface utilized to modify the systems 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 a man-machine interface this interface.

Following specification of the above defined tests, detailed validation test procedures are written which specify the detailed engineering tests and required system responses. The tests are then conducted on the system target hardware or a test jig which incorporates the system verified software. Deviations between the system response and the desired result are identified and officially recorded as a Validation Trouble Report. These trouble reports are then returned to the design group for resolution by one or more of the following techniques: software change; hardware change; validation test modification; functional requirements/decomposition modification; or no problem identified. If a software or hardware change is required, any revised software must be reverified followed by a validation retest.

6.3 Validation Testing of Software Modifications

At their discretion, the Validators may choose to perform the validation testing on a test jig, rather than upon the QDPS itself. The basis for the use of a test jig lies in the fact that the QDPS is made up of multiple independent microprocessor based subsystems. The independence of these subsystems allows a test jig to be made up of a partitioned set of the entire set of subsystems, as long as the data flowing into the subset exactly simulates the data flow in the actual QDPS with respect to timing, content, and organization.

The rational for doing the validation on a test jig for software modifications is that some software modifications only affect one or more of the QDPS subsystems. So long as the interactions with the other portions of the system are well understood, and are determined to have no impact on previously performed testing, validation testing on a subset of the QDPS will be as effective in assuring correct system performance as such testing on the integrated system.

In order that the independence of the validation test be maintained, and the traceability of the validation test to the actual system be assured, the following is required of a test jig:

a. The configuration of the test jig will be defined by the validation group. This avoids the conflict of the testing being constrained by a hardware configuration specified by system designers.

b. The hardware environment under which the software under test must operate shall be configuration tracable to the original system. This means that the microcomputer board set, the PROMs, the bus environment, etc. shall be the same as the in the original QDPS. Input signals to the subsystem shall be defined so as to appear to the software to be actual QDPS signals. Any use of hardware boards (CPU, memory) different from the actual QDPS shall be analyzed to demonstrate that their use does not affect the validity of the testing.

c. A documented configuration of the test jig shall be maintained and referenced in the validation test plan to provide an audit trail for the testing performed.

7.0 DEVELOPMENT AND VERIFICATION TEAM ORGANIZATION

During the system development process, two independent software teams will be utilized, one for development and one for verification. The software development team receives the System Design Specification, generates the Software Design Specifications, and then designs, develops, tests, and documents the code. The verification team receives the released code and its documentation from the design team, performs the required reviews and tests dictated by the Software Verification Level within the South Texas Verification Matrix and produces a Verification Test Report (VTR).

This kind of organization has several advantages. The use of two independent teams 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.

Team independence is essential to achieve these goals. In particular, the two teams will have separate team leaders. The chief verifier will report to a different supervisor than the developer. The separation guarantees that the verification team is free from pressure by the development team to compromise on testing. Note that the development team submits the code for verification only after the development team has confirmed the code to its satisfaction. Errors discovered during the development phase testing are not required to be documented by the verification team.

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 he is verifying.

7.1 Development Team

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 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 Team

The functions of the verification team are as follows:

7.2.1 Chief Verifier

Team leader who is responsible for all technical matters. His main duties are:

a. Check the Software Design Specifications received from the development team for completeness and unambiguity.

b. Check verifiers' Software Test Specifications for completeness.

c. Oversee verification of critical sections in the software.

d. Supervise and consult with the verification team.

e. Review Test Reports

f. Write the system validation test plan.

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.

e. Perform system validation test.

7.2.3 Librarian

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

T23817-1

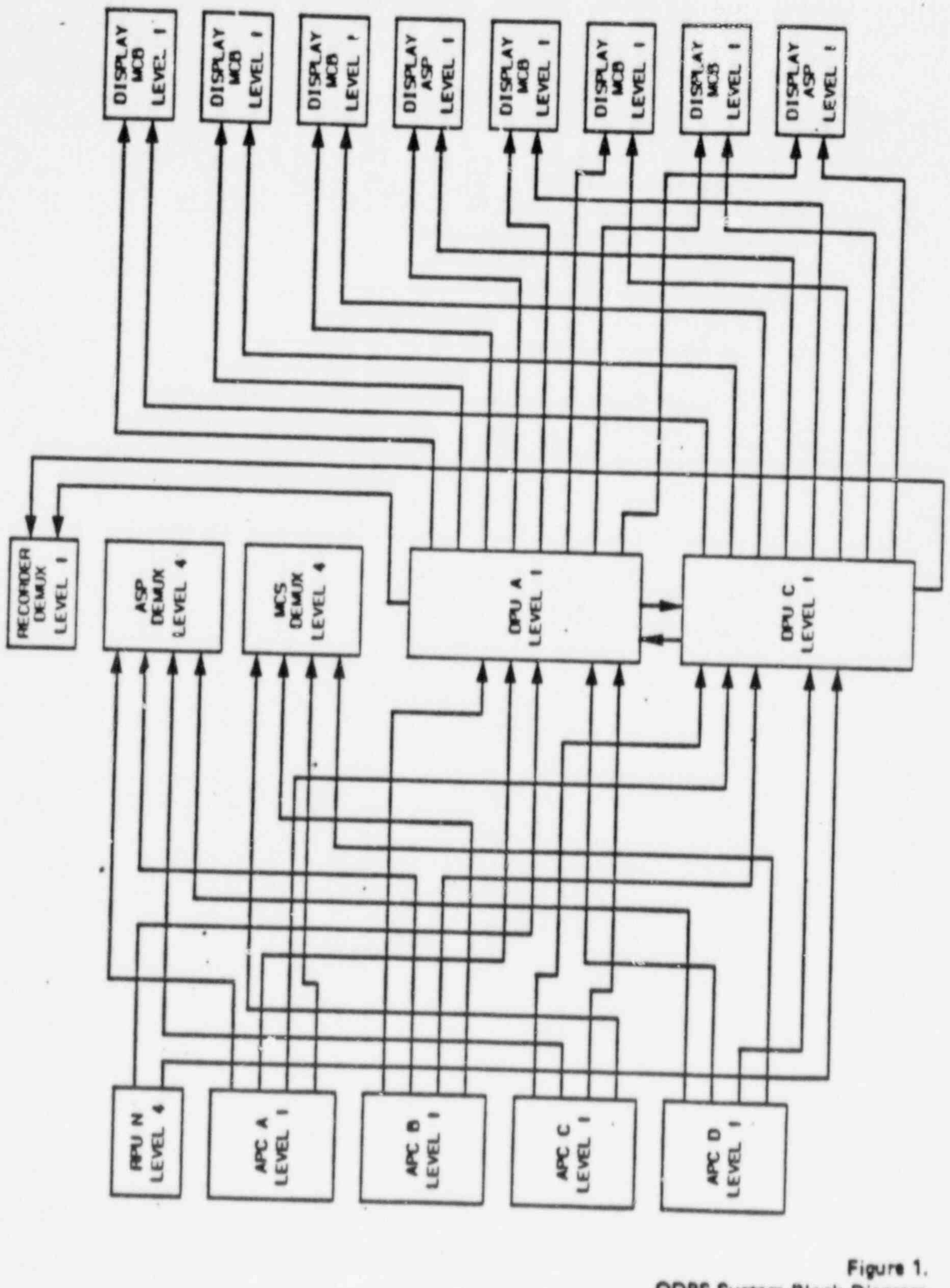
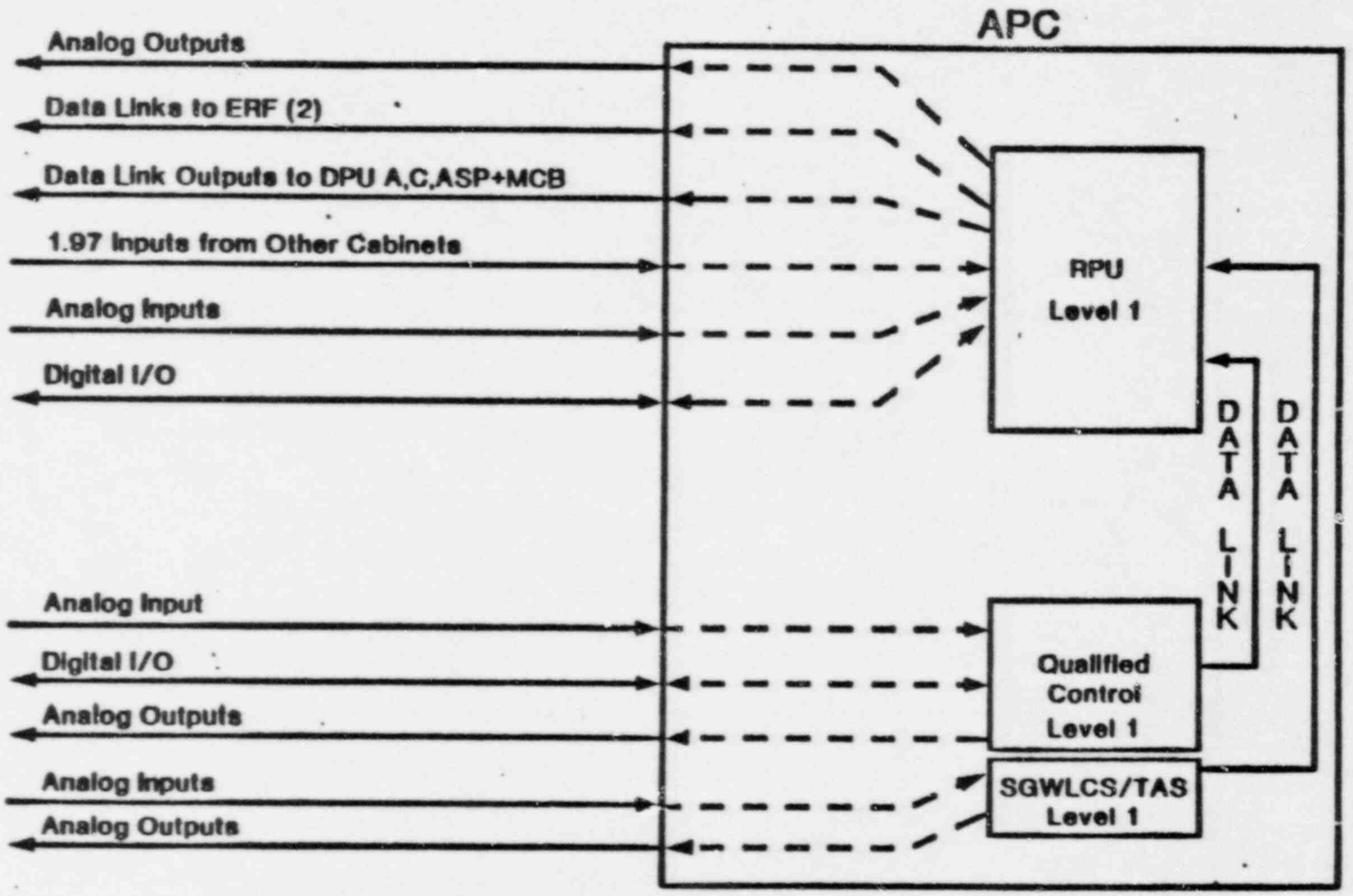
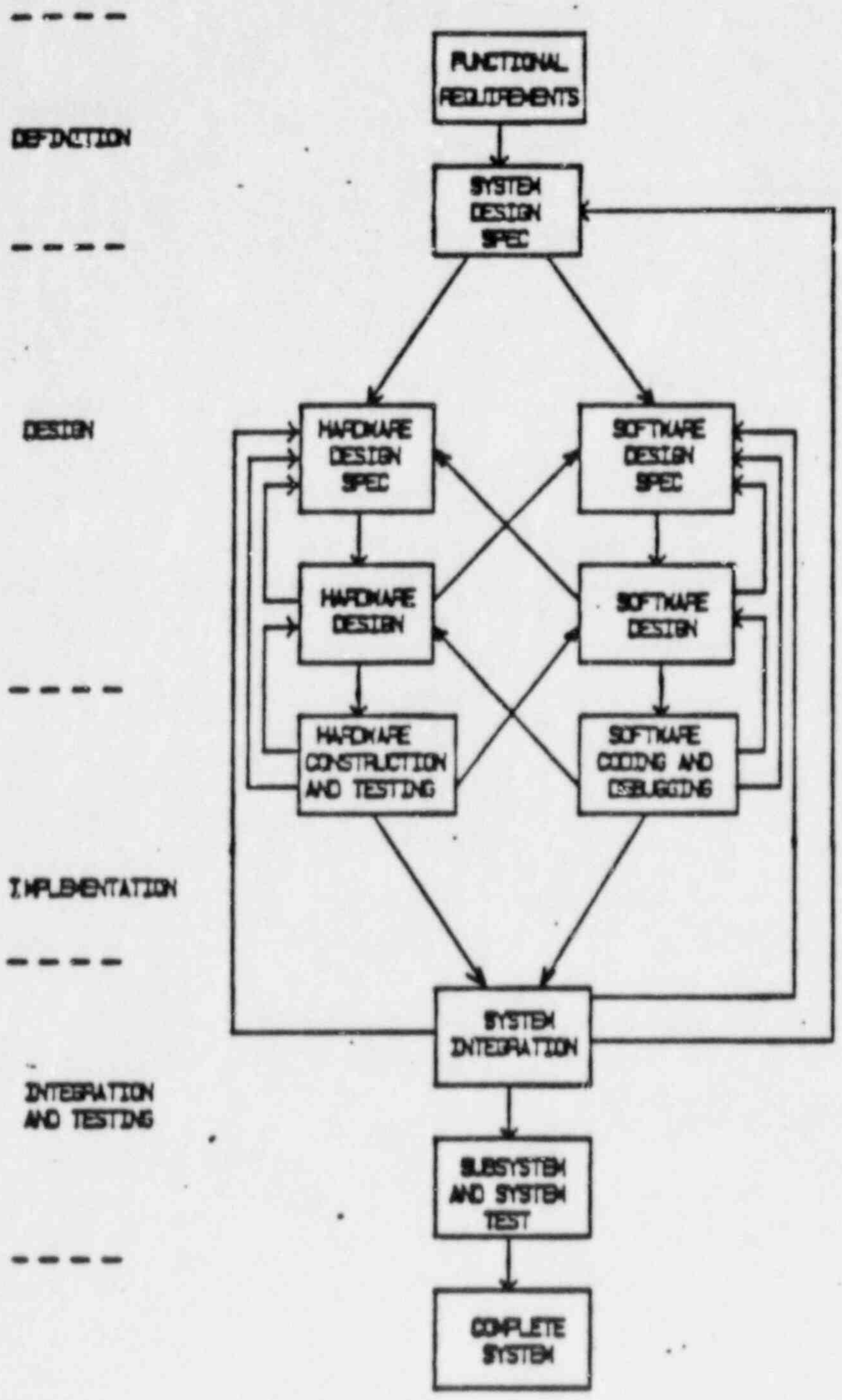


Figure 1.
QDPS System Block Diagram



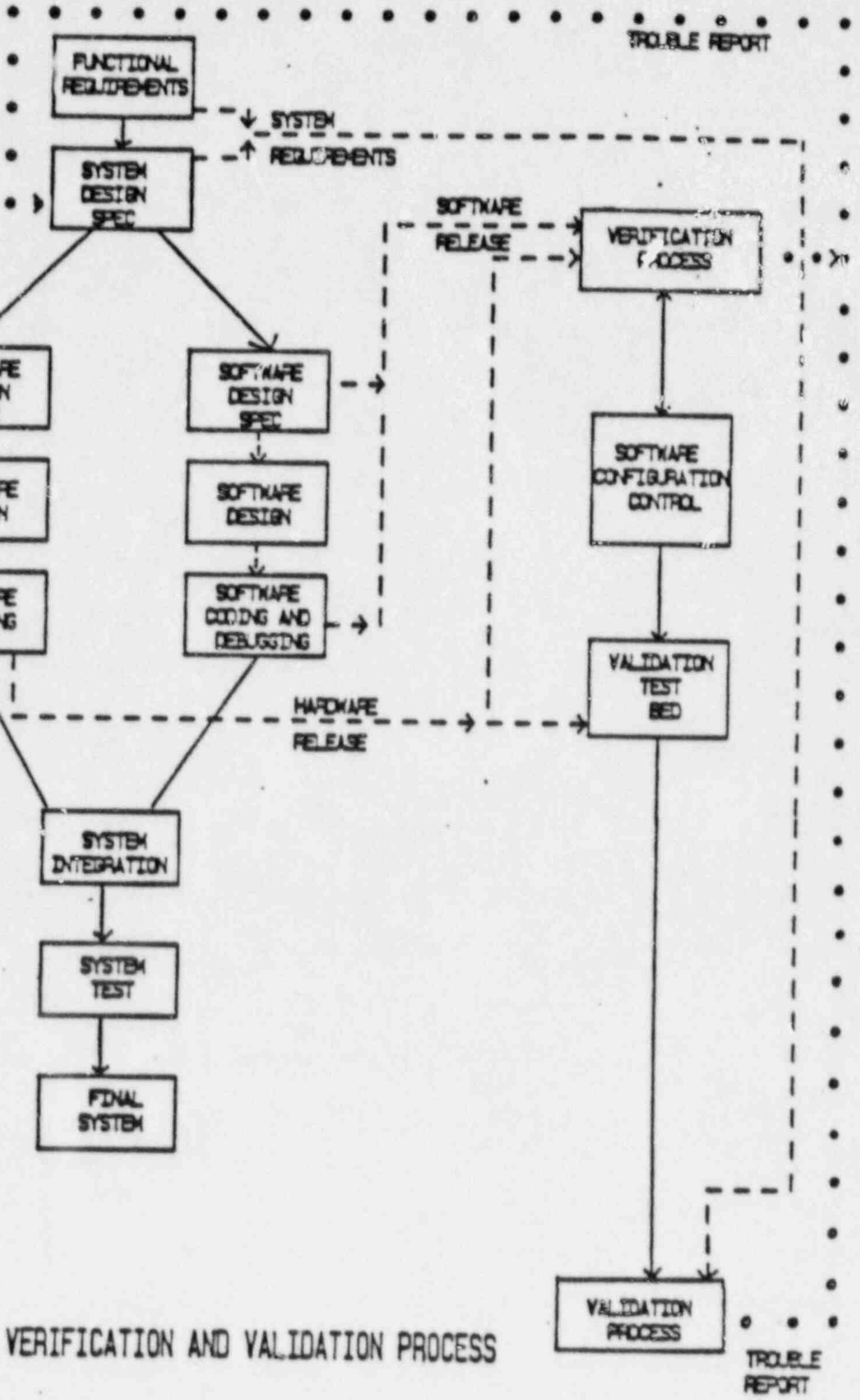
BASIC FEATURES AND I/O OF A CLASS IE AUXILIARY
PROCESS CABINET (APC)
Figure 2

909 T29814.001



QDPS DESIGN PROCESS

FIGURE 3



QDPS DESIGN VERIFICATION AND VALIDATION PROCESS

FIGURE 4

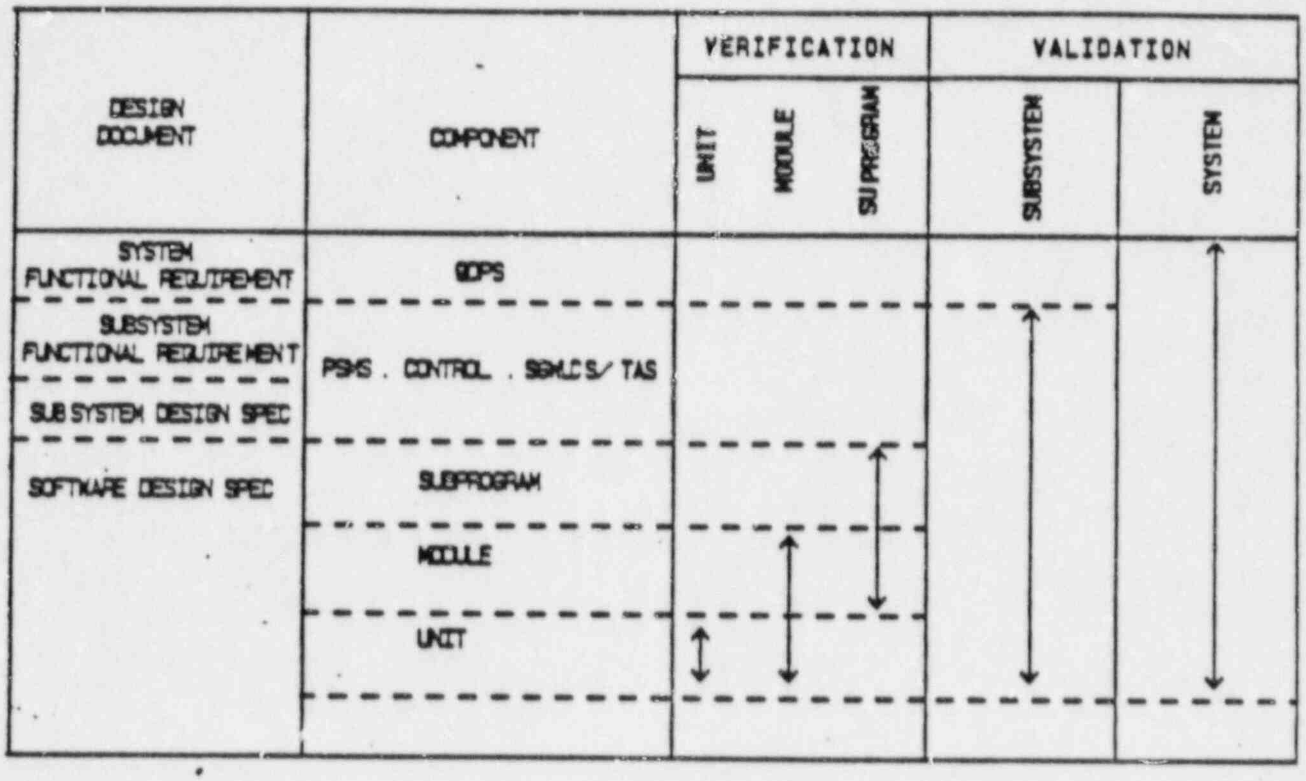
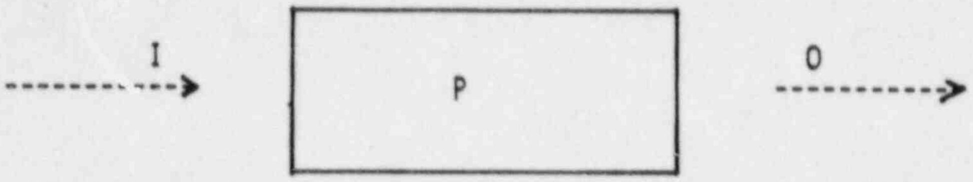

DESIGN DOCUMENT
COMPONENT
VERIFICATION
VALIDATION
UNIT
MODULE
SUBPROGRAM
SUBSYSTEM
SYSTEM
SYSTEM FUNCTIONAL REQUIREMENT
SUBSYSTEM FUNCTIONAL REQUIREMENT
SUB SYSTEM DESIGN SPEC
SOFTWARE DESIGN SPEC
QCPS
PSMS . CONTROL . SGWLCS/ TAS
SUBPROGRAM
MODULE
UNIT

FIGURE 5



Either (1) The system function has only one input and one output and for each input value/state there is one and only one expected output value or state. The transfer function in this case could be either linear or non-linear.

or (2) The system has more than one input and/or more than one output but they are disjunct and only specific inputs contribute to a specific output or a cluster of outputs and for each input cluster of values/states there is one and only one expected output value or state for a specific output or an output cluster. The inputs/outputs relationship can be represented in a simple tabular form and easily understood.

Figure 6. Model of a Visual Function

Software Verification Level	Safety Classification	Software Component			
		Unit	Module	Subprogram	
1	IEEE 279 ---------- IEEE 603 Para. 5.8.1	ST	ST/FT	ST/FT	
2	Qualified Controllers	ST/FT	FT	FT	Independent Testing
3	IEEE 603 Para. 5.8.2 Class 1E	FT	FT	FT	
4	IEEE 603 Para. 5.8.2 Non-Class 1E	FTR/SCR	FTR/SCR	FTR	Separate Review

FT	=	Functional Testing
ST	=	Structural Testing
SCR	=	Source Code Review
FTR	=	Functional Test Review
XX/XX	=	Non-Visual/Visual

Figure 7. South Texas Verification Matrix