Digital Instrumentation & Control Training

Module 5.0

Software/Firmware
Lifecycle Concepts

TABLE OF CONTENTS

LIST OF FIGURES

5.0 SOFTWARE/FIRMWARE LIFECYCLE CONCEPTS

Module Introduction:

Welcome to Module 5.0 of the Digital and Microprocessor Control Systems Course! This is the fifth of five modules available in the Digital Instrumentation & Control Training Course. The purpose of this module is to assist the trainee in recognizing basic software and firmware lifecycle information and terminology. This module is designed to assist you in accomplishing the learning objectives listed at the beginning of the module.

Learning Objectives

After studying this chapter, you should be able to:

1. Explain in very general terms what "Software Lifecycle" means
2. Explain in general terms what takes place in each of the six lifecycle phases defined in IEEE Std 1012-1998, "Standard for Software Verification and Validation Plans":
   a. Management
   b. Acquisition
   c. Supply
   d. Development
   e. Operation
   f. Maintenance
3. Explain in general terms the processes that comprise the Development Phase:
   a. Concept
   b. Requirements
   c. Design
   d. Implementation
   e. Testing
4. Explain the "Waterfall" software development concept.
5. Describe the most common reason software developments run into trouble
6. Explain the relationship between the following requirements specifications:
   a. Functional
   b. Software
   c. Hardware
7. In general terms, describe testing activities during these phases:
   a. Development
   b. Implementation
   c. Acceptance
   d. Operation and Maintenance

5.1 Software Lifecycle

Each individual digital I&C project is completed through multiple processes. These processes take the project from conceptual design through initiation of system operation. The processes are associated with the six Life Cycle Phases identified in IEEE Std 1012-1998:

- Management Phase
- Acquisition Phase
- Supply Phase
- Development Phase
- Operation Phase
- Maintenance Phase

For the purpose of this course, it is assumed that the Acquisition and Supply phases have been successfully completed with the selection of a digital system supplier whose system has received a Safety Evaluation Report (SER) for nuclear power plant safety related applications. This course will focus on digital I&C project development, including design, implementation and testing. It will also discuss testing as related to the operation and maintenance phases.

A simplified overview of a typical digital I&C project lifecycle is provided in Figure 5-15. This figure was adapted from EPRI TR-102348, Rev 1, "Guideline on Licensing Digital Upgrades," and illustrates the life cycle of a typical digital I&C system upgrade. The utility design and Verification & Validation (V&V) engineers should read and understand EPRI TR-102348 before proceeding with any significant design, procurement, or implementation activities involving a digital I&C project

5.2 Concept Process

The Concept Process involves the identification of the Project objectives and performance goals. During this phase, the Functional Requirements Specification (FRS) will be created. This document becomes the basis for all downstream hardware and software requirements specifications and application development. The steps in this process are discussed below:

5.2.1 Project Definition

Overview

A more detailed overview of the digital I&C upgrade project is provided in Figure 5-1 (Slide 5.2.1-5). The main flowpath down the left side of the figure shows the key steps in the modification process, starting with a change proposal and proceeding thru installation, operation and maintenance. The process has been simplified for this figure. For example, the administrative and contractual steps involved in an upgrade process (e.g. forming the project team, selecting vendors, etc) are not shown.

The upper right portion of the diagram shows activities associated with evaluation of potential system failures. In order to assess the impact of changes on plant design functions and safety, as well as on plant availability and investment protection, it is necessary to understand the potential failures (and other undesirable behaviors) of the system being modified and the effect that the modification will have on the likelihood and consequences of such failures. These activities will be referred to collectively as failure analysis in this module. Consideration of potential system failures should be an integral part of the design and implementation process for digital upgrades, interacting with all of the key design, specification and implementation activities, as shown in Figure 5-1. Although it is singled out on the diagram for emphasis, failure analysis is not a stand-alone activity or one that operates outside the design process.

Engineering evaluations are shown in the middle of the right side of Figure 5-1. Like failure analysis, engineering evaluations are activities that are performed as part of the design process, but are highlighted on Figure 5-1 for emphasis. Engineering evaluations include the collection of activities that are performed to demonstrate reasonable assurance that the system is safe and satisfies the specified requirements (e.g., for quality, dependability and performance). This may include evaluating and interpreting the results of the failure analysis, design verifications, software V&V, and review of vendor software design and development processes. Where appropriate and required by NUREG 0800, "Standard Review Plan," Chapter 7, analysis of overall defense-in-depth and diversity of the plant may be warranted to demonstrate the ability of the modified system to cope with common cause failures.

Licensing activities are shown in the lower right of Figure 5-1, illustrating their interaction with the design and implementation activities. It is important to note that many of the questions raised in licensing can be resolved using information that comes out of the failure analysis and engineering evaluations.

Some of the key design issues for digital systems are addressed at a number of points in the process of specifying, designing and implementing a digital upgrade. For example, quality assurance processes, require verification and validation activities be carried out throughout the design and implementation of the new system or subsystem. Similarly, human-system interface (HSI) design requirements need to be specified, appropriate verification and validations performed, and necessary training and procedures changes made as part of the implementation of the modification.

Pre-Conceptual Design

The first step of the project definition phase (as shown in Figure 5-2) (Slide 5.2.1-6) is to clearly define the objective(s) of the modification or replacement and establish early design concept(s). This is referred to as "pre-conceptual" because in some organizations, conceptual design is a formally defined phase of the design effort. Here, we are talking about the early concepts that are formed as the objectives of the change are defined. EPRI TR-102348, points out that plant systems and associated components that will be involved in the upgrade should be clearly defined early in the process. Key activities at this stage of the design process involve defining:

- Objectives of the modification
- Systems to be modified
- Other systems effected
- Early design concept alternatives
- High risk areas in the change process

Design Bases and Licensing

Understanding the design basis and licensing basis requirements of the system and the equipment being modified or replaced is necessary in order to assess the safety significance and selected a design approach for the upgrade. This will help to grade the efforts applied in downstream activities.

Source documents include:

- Regulatory requirements
- Final Safety Analysis Report (FSAR)
- Technical Specifications
- Industry codes and standards, system descriptions
- Equipment design documents
- Calculations
- Specifications
- Design basis document packages
- Other documents as required

It is very important to understand the current configuration, operating modes, and operations usage before defining the specification for the new system. It is also critical to determine the impact on plant documentation from the operations and maintenance procedures all the way through the plant software and hardware control processes.

Safety Significance

The next step is to determine the safety significance of the plant system or component being replaced. For example, if an individual controller is being replaced in a plant control system, the safety significance of the control system needs to be determined. The plant's Quality Assurance (QA) program (including a graded QA program if one exists) is a primary input to the determination. The plant probabilistic risk assessment (PRA), NRC Standard Review Plan (SRP) guidance, Technical Specifications and Licensing commitments also provide input to this process.

The complexity of the equipment and application also needs to be assessed, so that the appropriate rigor

for activities likely to be needed for evaluation and acceptance (to obtain reasonable assurance) is determined. Guidance covered here is documented in EPRI TR-102348 and EPRI TR-107339, "Evaluating Commercial Digital Equipment for High Integrity Application." This will provide input to the definition of critical characteristics and specific verification methods to be applied.

As documented in NRC Inspection Procedure 38703, the following factors should be considered in determining the extent of QA to be applied:

- The importance of malfunction or failure
- The complexity or uniqueness of the item
- The need for special controls and surveillance over process and equipment
- The degree to which functional compliance can be demonstrated by inspection and test
- The quality history and degree of standardization of the item.

Screening of Products and Suppliers

Identifying available products that will meet the project needs is a normal part of project definition. This activity is especially important when commercial grade components are to be used. One advantage of using commercial grade components is that there are a large number of candidates available. However, the larger variety also creates the need for additional screening effort.

Examples of questions to be used during the screening of potential vendors include:

- Does the vendor have a written quality assurance program?
- What industry standards does the vendor conform to?
- What software V&V methods are used?
- What software development documentation is available for review?

It may benefit the plant implementation process to standardize the platform and integrate the modification in a process that emphasizes use of a single platform – this will help in maintenance and operational phases. But at the same time, be careful in committing too early to a single platform.

Complexity and Failure Analysis

At this stage in the process, as shown in Figure 5-2 (Slide 5.2.1-6), with some specific equipment/product options identified, the failure analysis can look at external failure modes of the equipment and related these to the system-level evaluations already performed. Please note that internal failure modes of the equipment are probably not likely to be known at this time – this will be revisited later when the chosen product is evaluated. Also, early failure analysis can identify important design criteria and possible mitigation strategies, which can effect the definition of critical characteristics and verification methods to be used and may affect the design.

Another appropriate activity at this stage is to take a first look at complexity of each design option. The focus should be to keep it simple and to review the impact on the whole plant for the single design change – including impact on:

- Design
- Licensing
- Simulator
- Operations training
- Testing
- Maintenance

Project Activities

As shown in Figure 5-3 (Slide 5.2.1-13), the specific types of activities that are expected to be required to support evaluation and acceptance of the product(s) should be defined at this time. The early determination of project-specific methods and activities will allow tailoring the project to effectively address project needs, including specific requirements and dedication activities. Examples of the activities that project team members will be required to perform include:

- Engineering
  - Vendor and product evaluation
  - Equipment qualification
  - Application software development
- Licensing impact
- Procurement issues related to vendor acceptance (are they on the approved vendor list?)
- Procedures, training and simulator impacts

Cost/Benefit Evaluation

Cost/benefit assessment is a key consideration in determining whether and how to use commercial grade equipment. The traditional engineering economics approach to cost/benefit assessment does not apply well to nuclear plant I&C upgrades evaluations. The benefits of I&C upgrades are known qualitatively, but quantifying these generally is difficult to do. These benefits may include (for example):

- Reduced maintenance
- Less testing required
- Improved reliability
- Enhanced human machine interface (HMI)
- East of modification
- Savings on training and spare parts

The costs are clearly understood to be related to the direct costs of implementing the modification including:

- Plant change requirements
- Contractor activities
- Procurement of the new equipment

Iteration

Based on all of the above considerations, the project team is now responsible to iterate and identify the best option based on all of the considerations. The underlying basis for selection should be to solve the initial problem or need statement which initiated this project definition phase.

5.2.2 Software Safety and Risk Concepts

The purpose of this section is to provide an overview of how a utility approaches the concept of assessing and controlling digital risk. The main points to be covered are:

- Review some basic “safety definitions” and concepts
- Discuss the role and context of the Software Safety Plan

The objectives are clearly set up to define the following:

- The relationship between safety and risk
- The relationship of the Software Safety Plan to overall System Safety
- Sources of “digital risk.”

EPRI TR-102348 and TR-107339 provide examples of the definitions related to safety and risk. Because many components purchased by utilities now are from international companies, IEC 61508-4, 1998, “Safety Standard for Safety Instrumented Systems

(SIS)", is quoted in many vendor standards. This standard has the following definition of safety:

"Safety – freedom from unacceptable risk"

It is important to recognize that use of general industry standards does not always apply directly or well to nuclear applications, so the following cautions are provided:

- Nuclear power has formal "definitions" of safety systems which differs from most other industries
- The use of "risk" terminology varies across standards, industries, and disciplines.

It is best to avoid casual use of these terms in discussions with all affected parties – unless the common understanding of the meaning is clearly understood by all; suppliers, regulators and the like.

Basic "System" Definitions and Concepts

IEC 61508 provides the following definitions, which are important to our discussion here:

- **Failure** – termination of the ability of a functional unit to perform a required function
- **Fault** – abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function
- **Harm** – physical injury or damage to the health of people either directly or indirectly as a result of damage to property or the environment.

It is critical to recognize that "failures" and "faults" do not necessarily lead to "harm."

Additional definitions from IEC 61508 include:

- **Hazard** – potential source of harm
- **Risk** – the combination of 1) the probability of occurrence of harm, and 2) the severity of that harm.
- **Safety** – freedom from unacceptable risk

Hazards clearly have the ability or potential to cause harm. Risk assessment attempts to determine the probability of harm. Examples are provided for each of these cases.

Digital System Definitions and Concepts

Figure 5-4 (Slide 5.2.2-13) provides and overview of the software safety concept – and how the software safety plan fits into the whole process.

The following definition is used for "safety characteristic" from NRC Branch Technical Position (BTP) HICB-14, "Guidance on Software Reviews for Digital Computer-Based I&C Systems":

"Those properties and characteristics of the software system that directly affect or interact with system safety considerations."

Additionally;

"The safety characteristic, however, is primarily concerned with the effect of the software on system hazards and the measures taken to control those hazards."

Nancy Leveson, in her book Safeware, identifies the concept of software as:

"Software itself does not fail; it is a design for a machine. Software-related computer failures are always systematic."

IEEE Std 982.1-1988, "Standard Dictionary of Measures to Produce Reliable Software," provides important input into the definitions of a fault, and associated other definitions as follows:

- **Fault –** (1) an accidental condition that causes a functional unit to fail to perform its required function and (2) A manifestation of an error in software.
- **Bug –** see error and fault.
- **Error (2)** An incorrect step, process or data definition.

It should be recognized that software "bugs/errors/faults" are neither "abnormal events" as defined in IEC 61508 nor "accidental" as defined in IEEE Std 982.1.

Sources and Levels of Digital Risk

Figure 5-5, Figure 5-6 and Figure 5-7, (Slides 5.2.2-21, 22 and 23) provide overview of the sources and levels of digital risk.

Figure 5-5 (Slide 5.2.2-21) provides an overview of how a sub-system fault may affect plant safety – and the various facets of the design that can cause faults.

Figure 5-6 (Slide 5.2.2-22), from the University of Virginia, provides an overview of the different critical applications that occur in each of the three phases of a digital modification:

- Development Phase
- Operations Phase
- Maintenance Phase

Figure 5-7 (Slide 5.2.2-23) addresses the project risks and importance levels of early software system lifecycle impacts – with four levels of importance assigned to the level or risk.

5.2.3 Software Safety Plan

The objectives of this discussion are:

- Describe how a Software Safety Plan relates to nuclear safety associated with an I&C modification
- Identify the main elements of a Software Safety Plan and
- Discuss whether and how a Software Safety Plan might be developed for an example controls upgrade scenario.

Using development of a control room heating, ventilation and air conditioning (HVAC) modification as an example, a number of key items need to be planned:

- Preparations for NRC review
- Setup to be consistent with BTP HICB-14
- Formalization of the "Software Safety Plan"

The concept "Software Safety Plan" is misleading for nuclear applications, because the term is better suited to other industries, based on functional requirements and safety implications. The nuclear plant "safety model" separates safety from non-safety clearly and distinctly in the design and licensing process. This is not done in most other industries.

In development of the "Software Safety Plan" the utility should consider the following:

- Safety systems have safety functions
- Safety functions are critical to the analysis in Chapter 15 of the FSAR

The priority must be to establish reasonable assurance that the safety functions will be met. However, in the final analysis, we really want coverage beyond simply meeting the safety function of the system. We want to meet reliability and performance objectives as part of the modification as well.

The main elements of the Safety Plan are:

- Purpose
- Organization
- Responsibilities
- Risks
- Measurement
- Procedures
- Methods
- Standards
- Documentation

The development of the Safety Plan should start early in the process and be endorsed at the highest level possible in the company. Periodic updates are necessary to ensure the Safety Plan represents the modification as it evolves thru the different phases of the project.

5.2.4 Diversity and Defense in Depth

The objectives of this section are to:

- Identify the relevant regulatory documents
- Understand the importance of development of a process to minimize common mode failure.
- Understand the scope of defense-in-depth and diversity analysis and when it is required.
- Understand how to use alternate plant capability to provide diversity.

The regulatory requirements for defense-in-depth and diversity (called D-cubed) analyses are from:

SECY-93-087, "Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs," which led to Branch Technical Position (BTP) HICB-19, "Guidance for Evaluation of Defense-in-Depth and Diversity in Digital-Based I&C Systems"

- Applies to digital Reactor Trip and Engineered Safety Features Actuation System (ESFAS) functions only – as described in BTP HICB-19 and the SRP.
- Maintains four separate and independent levels echelons for defense-in-depth:
  - Control systems
  - Reactor trip
  - ESFAS
  - Monitoring, indication, manual controls and Emergency Operations Procedures

The above is all based on the guidance in NSAC-125, "Guidelines for 10 CFR 50.59 Safety Evaluations."

The NRC position on Common cause Failure is as follows:

- Software cannot be proven to be error free
- High quality design reduces the likelihood of common mode failures
- For RPS and ESFAS, the defense against common mode failure must be demonstrated.

The guidance in BTP HICB-19 and the SRP depend on the concept of "A Graded Approach." The need for level of analysis is based on whether the modification applies to the RPS and ESFAS and depends on the multiple-echelon levels of defense-in-depth.

The process to be followed by the utility for modification involving RPS/ESFAS includes the following major points:

- Perform D-cubed analysis to show the software common mode failure is addressed – following NUREG/CR-6303," Method for Performing Diversity and Defense-in-Depth Analyses of Reac-

tor Protection Systems," and EPRI TR-1007997, "Guideline for Performing Defense-in-Depth and Diversity Assessments for Digital I&C Upgrades: Applying Risk-Informed and Deterministic Methods," guidance.

- Analyze the effects of failures on mitigation of Chapter 15 accidents using "best-estimate" methods.

The four main points in requirements for operating reactors and ALWRs include:

- Licensee should assess defense-in-depth and diversity of the proposed system to demonstrate that vulnerabilities have been addressed
- Demonstrate that each postulated common-mode failure analyzed for each event in the FSAR
- If a postulated common-mode could disable the safety function, then a diverse means to accomplish the same function or a different function may be required
- Provide set of displays and controls in the main control room for manual system-level actuation and monitoring.

NOTE: For digital system modifications to operating plants, retention of existing displays and controls in the main control room may satisfy point 4.

Figure 5-8 provides an overview of the echelons of defense and the protection provided. Figure 5-9 provides the overview of the listing of dversity attributes and criteria that can be used for a D3 analysis.

Next, we review the status of the D3 Technical Working Group on cyber security and the results of meetings including both NRC and industry representatives.

The modification needs to provide diverse means of performing safety functions affected by common mode failures. It needs to provide diverse displays, following BTP HICB-19 for manual controls. Additional acceptance criteria are defined in BTP HICB-19.

The D-cubed analysis must do the following:

- Demonstrate minimal risk of software common mode failure
- Evaluate safety functions, system design and failure consequences – per BTP HICB-19 and NUREG/CR-6303.
- Evaluate accident scenarios using "best estimate" accident analysis assumptions (not 10CFR50 Appendix K)

The practical goals of this analysis include:

- Limiting the scope of new safety analyses required based on engineering evaluations
- Identify the most limiting best-estimate accident in conjunction with common-mode failure.
- Use analysis results to determine required operator reactor time to maintain the safety function(s).

In typical plant design, non-safety plant control systems usually utilize diverse hardware and software controls and are part of the analysis or engineering evaluation. The extent of the diverse indication and control is important and must be determined as part of the D-cubed analysis.

The need for additional common mode failure (CMF) protection must be defined as part of the D-cubed analysis. BTP HICB-19 states:

"Vulnerability to CMF affecting response to Loss of Coolant Accident (LOCA) has been accepted… based on provisions of leak detection and pre-defined

operating procedures that together enable operators to detect small leaks and take corrective action before a large break occurs."

5.2.5 Software Risk

The purpose of this section is to perform the following:

- Define "software defect"
- Discuss the effectiveness of software testing
- Define "software safety."

Nancy Leveson in her book, Safeware, provides the following definition:

"Software does not exhibit random wearout failures as does hardware; it is an abstraction, and its "failures" are therefore due to logic or design errors."

In the reference, Introduction to the Team Software Process, Software Engineering Institute (SEI) series in Software Engineering, page 103-104 provides the following:

"Injection rates of two defects/hour are common during detailed design and six defects/hour are normal during coding. …Defect injection rates vary considerably among engineers and even among programs written by the same engineer."

In an example case, documented in Applied Software Measurement, the following is noted:

"The average 1,000 function point system – not a large system – will be delivered with 556 defects of which 6 are critical and 78 are significant."

The roles and responsibilities of testing are documented in detail in IEEE Std 1012 and covered in the testing phase of this module. This section is intended to provide an overview of testing and recognize that testing cannot catch all errors or even most of the errors. Many examples are available in computer engineering literature to document the process and experience of testing in finding software faults. An example is documented in the Introduction to the Team Software Process, SEI series as follows:

"The Magellan system had only 22 lines of code of software. It has a total of 186 defects in system test, 42 of them critical and only one critical defect was found in the first year of system testing."

NOTE: Some digital indicators have 20,000 or more lines of code.

Risk is also related to a series of software severity levels for defects as noted in Applied Software Measurement:

- Severity 1: System or program inoperable
- Severity 2: Major functions disabled or incorrect
- Severity 3: Minor functions disabled or incorrect
- Severity 4: Superficial errors

Nancy Leveson, in Safeware, documents the following:

"Software system safety implies that the software will execute within a system context without contributing to hazards." Also, in this text, software is noted to affect system safety in two ways:

- It can exhibit behavior in terms of output values and timing that contribute to the system reaching a hazardous state, or
- It can fail to recognize or handle hardware failures that it is required to control or to respond to in some way.

In conclusion:

- Software "bugs" are design defects
- Software testing is not enough.

5.2.6 Software Fault Prevention

Software fault prevention is addressed formally in the regulations in BTP HICB-14, IEEE Std 1012 and Regulatory Guide 1.168, "Verification, Validation, Reviews, and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." This is addressed in an upcoming section, but the concept of fault prevention can best be described in an example covered in this section, from the Bell Labs Technical Journal, dated April-June 1998.

Overall the goal of fault prevention is to help programmers avoid injecting the most frequent faults into a product. The referenced article introduces the coding fault prevention process and the technical guidelines used to prevent coding faults. It also explains the results achieved and the metrics used to measure them.

The team at Bell Labs completed a study of the switch fault removal effectiveness and fault flow by interviewing developers on the nature of the more than 600 faults noted. Performing an extensive root cause analysis of the faults found and fixed enabled the team to establish a baseline of the software faults life cycle.

As shown in Figure 5-10 (Slide 5.2.6-6 (Table 1)), a crucial finding determined from the analysis was that nearly half of the faults were coding faults, and the majority of them could have been prevented. The team closely examined the faults to understand the types of coding problems that existed. Figure 5-11 (Slide 5.2.6-8 (Table 2)) shows a list of the major coding faults found. The results of the analysis also showed that three types of faults – logic, interface and maintainability – account for more than 50% of the total coding faults.

The faults discussed in the Bell Labs article were actual faults found in a very large software project environment. The average programming ability and experience of developers connected with the project were high compared to the norm in the industry. The team extensively interviewed developers who produced the code and the faults found in it to validate the faults found and their causes.

Using a structure developed from software quality perspectives to describe various categories of coding faults, the team developed the faults in each category based on actual error information and considerations of how program components were used.

The team used a fishbone analysis to identify the root causes of the coding faults. The top three root causes of the faults were executions/oversights (38%), resource/planning (19%) and education and training (11%).

Based on these root causes, the team identified several solutions or countermeasures and then rated each countermeasure numerically using two process characteristics: effectiveness and feasibility. The paper and our associated slides provide an overview of the faults documented and the solutions and countermeasures recommended for each.

5.2.7 Quality Assurance (QA) Plan Development

The purpose of this section is to provide an overview of the major aspects of utility software Quality Assurance requirements that apply to all software usage onsite with the primary focus on quality affecting software and related usage.

In meeting 10 CFR 50 Appendix B, the utility has the responsibility to provide a set of controls over the entire software life cycle at a nuclear plant. The utility personnel are trained to be cognizant of the following three elements:

- Recognize the procedure that establishes the basis for the SQA (Software Quality Assurance) program
- Recognize the administrative procedure and requirements for implementing each element of the program
- Understand the major reasons why software controls are necessary at nuclear power stations.

Software plays a key role in all aspects of work at nuclear facilities. It enables personnel and processes to perform effectively and efficiently to ensure the safe, reliable and economic operation of the plant. The software must be trusted to work correctly every time it is used.

Within the scope of the utility specific programs, the term "software" includes the following:

- Software code
- Firmware
- Other programmable digital devices
- Configuration information
- Quality data

The Software Quality Assurance Program must apply to all software, quality data, and digital systems utilized with the nuclear program. The degree to which controls are applied to software, quality data and digital systems should be commensurate with the importance of the results or the consequences of an error.

The objective of the utility program is to apply sufficient controls to ensure the proper acquisition, development, testing, use and maintenance of software, quality data, and digital systems while not inhibiting the effective utilization of the technology.

The most basic need for SQA originates from the potential for latent defects or errors in software. One of the main objectives of the utility SQA program is to minimize the likelihood of defects being introduced into executable software code or programmable devices by applying appropriate systematic control techniques throughout the software life cycle.

Although the analysis and removal of defects is an important function of SQA, it is the prevention of defects that is the primary focus of SQA and related activities.

The basis for each utility SQA program is required to be established in the corporate nuclear Quality Assurance Program – usually documented in the Topical Quality Assurance Manual (TQAM). A specific section addresses the requirements for SQA and where the controls of SQA are provided. This is the top tier procedure applicable to the individual nuclear program or a group of nuclear sites, if controlled by the same design authority. A specific Software Quality Assurance Procedure usually provides more detail on the specifics of the program in meeting the requirements of the TQAM. In program implementation, the following major areas are usually addressed in specific guidance:

- Control of software design
- Software configuration management
- Software testing guidelines
- Control of software maintenance
- Electronic data/document quality

There are five specific reasons for imposing controls on software design, development, use and maintenance at a nuclear plant:

- Protection against conditions adverse to quality
- Ensure regulatory compliance
- Protect company investment

- Prevent violation of software laws
- Maintain Y2K clean management

The program implementation usually involves the explicit definition of responsibilities to individuals and organizations and the specific levels of classification or importance to software and applications.

5.2.8 Software Verification & Validation (V&V)

The purpose of this section is the following:

- Discuss software verification and validation
- Discuss IEEE Std 1012-1984, endorsed by Regulatory Guide 1.168 compared to IEEE Std 1012-1998
- Address the requirements for the Software V&V Plan by review of an example outline

The objective of this lecture is to address the phases of a V&V program that is based on IEEE Std 1012 and BTP HICB-14. Figure 5-12 (Slide 5.2.8-3) provides an overview of the software life cycle process and shows each of the following major phases:

- Management Phase
- Concept Phase
- Requirements Phase
- Design Phase
- Implementation Phase
- Test Phase

Each of these major sections has key component and focus areas that address the V&V requirements.

The management phase is not really a phase – but an ongoing process. Management phase refers to initial management plan production, organizational structure definition and management processes. These include management reviews, as well as issue reporting, tracking and trending.

The concept phase is the real initial phase of the software development process, in which user needs are described and evaluated.

In the requirements phase, the functional and performance capabilities for a software product are defined and documented.

The design phase is the period of time during which the design for the architecture, software components, interfaces and data are created, documented and verified to satisfy the requirements. The object of V&V during the design phase is to demonstrate that the design is correct, accurate, and complete in the transformation of the software requirements and that no unintended features are introduced.

The implementation phase is the period of time in the software life cycle during which a software product is created from design documentation and debugged. The implementation V&V activity addresses software coding and testing. The objectives of V&V are to verify and validate that these transformations are correct, accurate and complete.

The testing phase is the period of time in the software life cycle in which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied. The objective of V&V in the test phase is to ensure that the software requirements and system requirements allocated to software are satisfied by the execution of integration, system and acceptance testing.

IEEE Std 1012 provides guidance on software integrity levels and guidance on the associated level of rigor required to complete the test phase V&V.

Table 2 provides an outline for an example Software V&V Plan that meets IEEE Std 1012 guidance.

5.3 Requirements Process

The objective of this lesson is to understand the process by which the functional requirements developed in the Concept Process (Section 5.2) are transformed into detailed software requirements that become the basis for the software design.

Figure 5-13 illustrates a software development plan for the typical digital I&C upgrade project shown in Figure 5-14, (with more detail shown in Figure 5-15) and the relationship among the design processes.

- The functional requirements specify what the software is to do. (Concept Process output)
- The software requirements specify how the software accomplishes required functions. (Requirements Process output)
- The software design description (discussed in Section 5.4) lists what was done to accomplish the software requirements (Implementation Process output)

The remainder of this lesson is devoted to discussing how components in the plan are developed.

The requirements process transforms the functional system requirements into individual requirements documents for both hardware and software. This process also initiates verification and validation activities for the project. The inputs and outputs for the Requirements Process are shown in Figure 5-16.

Any upgrade to an existing system must meet the functional, safety and reliability requirements of the existing system without introducing new, unintended functions that could have an adverse impact on plant reliability and operation. The major benefit of an organized development of functional requirements is the reduction of risk associated with unintended functions in the upgrade.

5.3.1 Requirements Characteristics

The Functional Requirements Specification (FRS) should not only specify what the upgrade system should do, but also specify what the upgrade system should NOT do under specified circumstances. In industry history, a significant number of problems that have been identified as “software error” or “software failure” are actually due to inadequate or incomplete specification of functional requirements.

Problems that are determined early in the upgrade life cycle may be corrected more easily (and economically) the earlier they are detected. If problems are left undetected and uncorrected until testing or later, the cost to correct the problem may be orders of magnitude greater than if corrected during requirements development.

EPRI TR-108831, “Requirements Engineering for Digital Upgrades; Specification, Analysis, and Tracking,” combined the results from several earlier EPRI projects to develop an approach for functional requirements specifications that combines the use of early conceptual designs and consideration of risk. The approach facilitates the use of vendor specifications when commercial grade equipment will be used in the upgrade. This section summarizes the design philosophy described in EPRI TR-108831.

Figure 5-17 illustrates a model for requirements specification development that is consistent with engineering practices in the nuclear industry. The model readily lends itself to adaptation to the scope, complexity, cost, and risk associated with digital upgrades, whether devices or systems, safety-related or non-safety related.

The lead engineer should determine if the complexity or potential risk of the proposed upgrade justifies the formal approach and format recommended by EPRI TR-108331. As guidance for this determination, the engineer may consider the following items to determine the level of effort appropriate for functional requirements specification activities:

- Project risk associated with the upgrade,
- Type of the upgrade (Device, System, Safety-related),
- Quality assurance activities to ensure that correct requirements are stated and that the installed system meets the stated requirements.

5.3.1.1 Project Definition (Problem Analysis)

In the problem analysis phase of the requirements specification model, the engineer should:

- Identify what the upgrade must (shall) do;
- Identify what the upgrade must not (shall not) do;
- Identify constraints and interfaces;
- Assess financial and regulatory risk.

5.3.1.2 Composition of Requirements

Composition of requirements is the process by which the written Functional Requirements Specification (FRS) is created. The FRS is the top-most document in the life cycle of a project, as it defines what the project is to do; that is, the functions that will be performed. All other documentation is based upon the Functional Requirements Specification. Currently, there is no accepted industry standard for preparation of an overall project Functional Requirements Specification. IEEE Std 830, "Recommended Practice for Software Requirements Specifications," provides a format for a typical software requirements specification. EPRI TR-108831 builds on and extends the outline provided in IEEE Std 830, and presents a suggested format that can be adapted for a particular use. Systematically addressing the outline will assist the project engineer in developing a requirements specification that is reasonably complete.

The project engineer should discuss most of the requirements in natural language; that is, a verbal description of what is desired. For more critical requirements, the engineer may use other descriptive methods to ensure clarity, such as described in EPRI TR-108831, page 3-8, "Requirements Statements." If a requirement may be specified more precisely (and confidently) with one of the other methods, the other method should be used.

The engineer should be careful not to over-specify. Digital I&C upgrades should use commercially available equipment as much as possible to control costs. Specification of requirements that commercial equipment cannot meet will limit the choice of suppliers.

5.3.1.3 Requirements Analysis

Requirements Analysis is a process by which the engineer reviews the developing Functional Requirements Specification and ensures that the requirements are organized and coherent. The engineer should ensure that:

- The requirements define what needs to be done NOT how to do it.
- The requirements define what shall NOT be done; that is, unacceptable system behaviors.
- The requirements are abstract (independent of the implementation), unambiguous (can be interpreted in only one way), necessary (verifiable, or related to a specific statement of need), and validatable (testable).

The requirements analysis process recommended in EPRI TR-108831 consists of the following steps:

Concept Evaluation

The conceptual design is a tool that enables the engineer to understand the requirements of the upgrade, and to determine if the requirements can be met with existing hardware and software. The conceptual design should include, at the system level, all necessary interfaces and functional (operating) requirements. Any significant difference in function, operation, and interface or failure mode between the conceptual design and the functional requirements should be identified and evaluated for its impact on overall plant operation and safety at this early stage.

The complexity of the project determines the level of detail required in the conceptual design. EPRI TR-108831 describes a graphical conceptual design method by which the engineer may ensure that specified functional requirements are properly connected and may also identify missing functions and unnecessary complexity.

System Failure Evaluation

In performing a System Failure Analysis, the engineer identifies potential problems that may occur with the new system. It is not necessary to determine or evaluate detailed failures below the system level. However, the engineer should be aware of, and evaluate, new failure modes and effects that may be created as a result of the nature of the upgrade. Early in the requirements process, the informal hazard analysis identified in Section 5.3.2 is adequate.

EPRI TR-104595, "Abnormal Conditions and Events Analysis for Instrumentation and Control Systems," describes several methods that may be used to perform a failure analysis. If the upgrade is safety-related, the engineer should perform a single failure criterion analysis to ensure compliance with applicable sections of 10 CFR 50 Appendix A as interpreted by IEEE Std 279, "Criteria for Protection Systems for Nuclear Power Generating Stations," and IEEE Std 603, "Standard for Safety Systems for Nuclear Power Generating Stations." EPRI TR-102348 provides guidance for this analysis.

Completeness Evaluation

The evaluation of "completeness" is the process by which the engineer determines that the Functional Requirements Specification describes the black-box behavior of the upgrade system in sufficient detail to distinguish it from any undesired system that could be designed.

The specification is incomplete if the system or software behavior for some events or conditions is omitted or is subject to more than one interpretation.

The following list of 18 questions, adapted from EPRI TR-108831, Figure 3-9, may be used by the engineer to find omissions and inconsistencies in the Functional Requirements Specification, and thus determine if the specification is "sufficiently" complete. For more complex systems, the engineer should refer to EPRI TR-108831 for more rigorous methods.

1. Is the software's response to out-of-range values specified for every input?
2. Is the software's response to NOT receiving an expected input specified? Are timeouts provided? Does the software specify the length of the timeout, when to start counting the timeout, and the latency of the timeout (the point past which the receipt of new inputs cannot change the output result, even if they arrive before the actual output)?
3. If input arrives when it should not, is a response specified?
4. On a given input, will the software always follow the same path through the code? Is the

software's behavior deterministic?

5. Is each input bounded in time? Does the specification include the earliest time at which the input will be accepted and the latest time at which the input will be considered valid?
6. Is a minimum and maximum arrival rate specified for each input (for example, a capacity limit on interrupts signaling an input)? For each communication path? Are checks performed in the software to avoid signal saturation?
7. Is there a requirement for interrupts to be masked or disabled; can events be lost?
8. Can any output be produced faster than it can be used (absorbed) by the interfacing module? Is overload behavior specified?
9. Is all data output from the sensors to the buses used by the software? If not, it is likely that some required function has been omitted from the specification.
10. Can input that is received before start-up, while off-line, or after shutdown influence the software's start up behavior? For example, are the values of any counter, timers, or signals retained in software or hardware during shutdown? If so, is the earliest or most recent value retained?
11. In cases where performance degradation is the chosen error response, is the degradation predictable (for example, lower accuracy, longer response time)?
12. Are there sufficient delays incorporated into the error-recovery responses; e.g., to avoid returning to the normal state too quickly?
13. Are feedback loops (including echoes to the screen) specified, where appropriate, to compare the actual effects of outputs on the system with the predicted effects?
14. Are all modes and modules of the specified software reachable (used in some path through the code)? If not, the specification may include superfluous items.
15. If undesired system states have been identified, does every path from an undesired state lead to a low-risk state?
16. Are the inputs identified which, if not received (for example, due to sensor failure), can lead to an undesired state or can prevent recovery (single point failure)?
17. Do the hardware tolerances specified in the software (possible as boundary conditions for signaling a reactor trip) correspond to the actual hardware tolerances?
18. Is feedback provided to the user or on the operator's console that indicates the status of all critical hardware items in the system?

Testability Evaluation

Good requirements statements are verifiable, that is, related to a specific statement of need. However, not all requirements are readily validatable, or demonstrated to be implemented successfully in the design. The preferred means of validation is testing. For non-testable requirements, other means of validation such as analysis are acceptable.

The following set of test types, adapted from EPRI TR-108831, Table 3-10, may be used to evaluate testability. The test types are listed in order of increasing implementing expense:

- Independent Analysis or Review
- Static/Bench Test
- Special Test System or Simulator
- Factory Acceptance Test (FAT)
- Site Acceptance Test (SAT)
- Return to Service Surveillance Test (RTS Test)

The FAT, SAT, and RTS Test are described in more detail later.

The engineer should prepare a spreadsheet containing a matrix that lists each requirement in the specification. This spreadsheet becomes the Requirements Traceability Matrix (RTM) as used throughout the DCS Conversion Procedure, the SVVP and other documentation.

For each requirement, the engineer should note the test that most readily validates the requirement. If a requirement exists for which no test type is apparent, that requirement should be clarified such that it becomes testable, or it should be deleted.

The RTM should provide a clear tie between specifications and the implemented design. Once developed in the design phase, the RTM is used as input to the final hardware and software test plans and reports.

Plant Simulator Impact

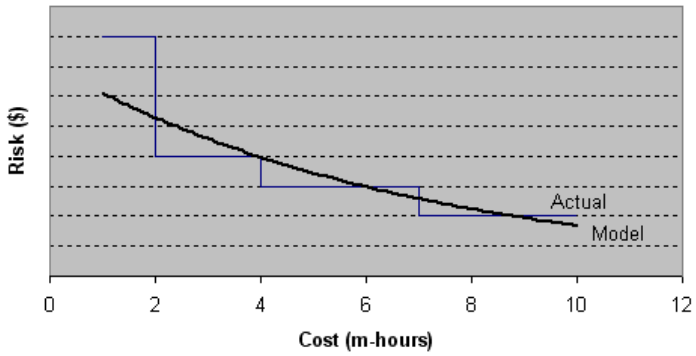
Because digital systems will potentially be used by control room personnel or feed systems used by control room personnel, an evaluation must be made to assess the impact on the plant simulator and the overall ability of the proposed design to be implemented in the plant simulator. Complex digital systems can pose a simulation problem due to the need for the plant simulator to replicate functionality or drive intermediate hardware and software to achieve adequate fidelity.

5.3.1.4 Risk Assessment

In the project risk assessment process, the engineer should identify areas where the Functional Requirements Specification is "soft" or incomplete. These areas represent project risk. The engineer should then evaluate the risks and identify whether they are acceptable, or if the design should be modified to reduce project vulnerability.

For example (per EPRI TR-108831), if the requirements analysis activities determine that the upgrade involves a License Amendment, it may be advisable to modify the design and eliminate the need for a License Amendment. Modifying the design may increase project cost, but it may be more economical, with less risk to the overall project schedule and the ultimate cost, than pursuing a License Amendment.

The figure below illustrates the relationship between project risk and the effort required to reduce risk using the iterative method described in TR-108831:



Project risk decreases as effort increases, but risk can never be reduced to zero. At the point where project risk becomes acceptable, the requirements specification effort should be stopped and the project should move on to the implementation phase.

5.3.2 Identifying Safety Requirements

For a safety-related system, identification of safety requirements should be done very early in the requirements development process. This lesson will examine the major steps in identifying hazards and safety requirements with additional details provided in Section 5.3.3. Emphasis will be placed on clearly defining that the system must do as well as what the

system must not do. See Figure 5-18. The reason for early consideration of safety requirements is that attention is frequently paid to *process* requirements, without consideration of the impact of unforeseen events on *safety.*

The first step is to prepare a table with the following columns:

- Hazard Description
- Hazard Cause
- Hazard Level (High/Medium/Low)

With the table constructed, the analyst will list hazards that can befall the system. It is helpful to perform this task together with system "experts" who are familiar with the application. For each hazard, the cause and level are identified. Severity of consequences determines the level.

Hazards are frequently dismissed as "not credible" without going through the process outlined above. Documenting the evaluation will help ensure that each hazard is evaluated as to its "credibility" and can be addressed accordingly.

Once hazards are identified, they can be managed. This can be done through:

- Elimination. The design is simplified; unnecessary "requirements" are removed, more appropriate requirements are substituted, or specific human errors are corrected.
- Reduction. Safety margins are increased, controllability is increased through improved design, and barriers to hazards are inserted.
- Control. Exposure to hazards is reduced, the hazards are contained or isolated to minimize adverse impact, and protection systems can be implemented to mitigate impact of hazards.

5.3.3 Preliminary Hazards Analysis

Prior to developing a conceptual design, the engineer should perform a Preliminary Hazards Analysis (PHA) that identifies the fundamental design and licensing issues posed by the upgrade. It is not necessary to formally document the PHA at this time, as it is primarily a vehicle to assist preparation of the Functional Requirements Specification. The engineer may attach the PHA to the FRS to document the PHA. As a minimum, the engineer should consider keeping the PHA as a set of desk notes to use later; e.g., when preparing the 10 CFR 50.59 evaluation.

The following information, adapted from EPRI TR-108831 may be used to perform the PHA:

1. Can the upgrade malfunction in such a way that a safety function is defeated?
2. Can the upgrade malfunction in such a way that could create the possibility of a new accident or unanalyzed plant condition, transient or equipment malfunction?
3. Can the upgrade malfunction in such a way that the probability (risk) of occurrence, or the consequences of a currently analyzed accident, plant condition, transient or equipment malfunction are increased as compared to the existing system?
4. What is the worst consequence if the upgrade fails?
   - How is the failure detected?
   - How is the failure mitigated? (How is recovery accomplished)?
   - Is a diverse backup means of control, protection or information display available?
   - Is operator action relied upon as the diverse backup? If so, is adequate time available for the operator to perform the required function within the existing licensing basis?

5. Does the end user interact with the upgrade in a manner that is significantly different from the existing system or device? What are the consequences if operations or maintenance personnel use existing (pre-upgrade) procedures for the upgrade?
6. Are the upgrade plant interfaces (electrical, mechanical, structural) compatible with similar interfaces of the existing system or device?
7. Does the upgrade alter any fundamental basis for the operating license as discussed in the plant Safety Evaluation Report (SER) or any supplement to the Safety Evaluation Report (SSER) for the plant or any other upgrades or modifications that have been performed since the operating license was approved?
8. What are the performance requirements for the upgrade? NRC Branch Technical Position (BTP) HICB-21, "Guidance on Digital Computer Real-Time Performance," provides assistance.

5.3.4 Development of the Software Requirements Specification (SRS)

The purpose of the SRS is to ensure that:

- The customer describes what he wishes to obtain;
- The supplier understands what the customer wants;
- The developer designs the software according to documented and accepted requirements.

Regulatory Perspective

Regulatory Guide 1.172, "Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," discusses the SRS in terms of meeting quality criteria of IEEE Std 279-1971. In addition, several of the General Design Criteria of 10 CFR 50 Appendix A describe functions that are part of the design bases and that would be included in the SRS of any software that is used to perform design basis functions. Appendix B, Criterion III, "Design Control," requires measures for design control and measures to verify design adequacy. Appendix B also contains general criteria for quality assurance practices.

Regulatory Guide 1.172 endorses IEEE Std 830, with a few exceptions, as a method for achieving high functional reliability and design quality for safety system software. Although the standard does not specifically address safety systems, it provides guidance on the development of software requirements specifications that will exhibit characteristics important for developing safety system software.

The Software Requirements Specification (SRS) addresses the software functional and performance requirements, including any resident or embedded operating system. From a regulatory standpoint, the SRS is the critical link between what the utility asks (FRS) for and what it gets (SDD). As discussed above, if the engineer does not specify what he wants, he is unlikely to get it. Likewise, if he does not specify what he does not want, he may be surprised by what he gets.

According to IEEE Std 830, the SRS is a specification for a particular software product or program that performs certain functions in a specified environment. The SRS addresses the following basic issues:

- Functionality (What is the software supposed to do?

- External Interfaces (How does the software interact with people, the hardware platform and other hardware and software?)
- Performance (Speed, availability, response time, recovery from errors)
- Attributes (Portability, correctness, maintainability

The SRS does not contain design requirements; those are contained in the SDD.

Characteristics

Characteristics of a good SRS are described in IEEE Std 830.

1. Accuracy

Software should accurately reflect every requirement derived from system requirements and safety analyses. To accomplish this, the SRS should be unambiguous; every requirement should have only one interpretation. The use of natural language is inherently ambiguous, and should be reviewed by an independent party. Specification languages or other representations can be used to reduce ambiguity.

2. Completeness

The SRS should specify how functions are initiated and terminated, as well as the status at termination. Accuracy requirements (units, error bounds, data type, data size) should be provided for each input and output variable. Physical variables that are controlled or monitored should be fully described. Prohibited functions should be described.

Timing requirements are especially important. Timing constraints and acceptance criteria should be identified for each mode. Timing requirements should be deterministic; that is, they should not vary with the process. Timing requirements for normal and abnormal operation should be specified.

A document containing "LATER" or "TBD" is not complete.

3. Consistency

As used in RG 1.172, the consistency should be internal and external. External consistency means the SRS is consistent with associated software and system products. Internal consistency means that no requirement in the specification conflicts with any other requirement in the specification. IEED Std 830 does not require external consistency, because it treats an external inconsistency as an error in requirements composition.

4. Ranking

The SRS should be ranked for importance; that is, critical and non-critical (necessary vs. desirable) requirements should be identified. Software requirements important to safety must be identified as such in the SRS. IEEE Std 830 suggests three degrees of necessity: essential, conditional and optional. Requirements that are conditional or optional are not necessary for the system to be considered acceptable.

For safety system software, unnecessary requirements should not be imposed.

5. Verifiable

The SRS should be verifiable; a requirement is verifiable only if some finite cost-effective process can be used to check that the requirement is met. Unverifiable requirements should be modified or restated so it is possible to verify them. If unverifiable requirements cannot be so modified or restated, they could be removed.

6. Modifiable

The SRS should be modifiable; changes in requirements can be made easily and consistently while retaining the structure and style of the document. This term is closely related to the style (form, structure and modularity), readability and understandability of the document.

7. Traceable

The SRS should be traceable. Forward Traceability means that each of its requirements can be referenced in subsequent documentation. Backwards Traceability means that each requirement can be traced back to its source in earlier documents; i.e., the FRS. The Requirements Traceability Matrix (RTM) is the tool used to trace requirements.

A forward trace should exist from each requirement in the SRS to the specific inspections, analyses or tests used to confirm that the requirement has been met.

Evolution

The SRS will usually evolve during the software development process. Details may not be available when the project is initiated, or changes may be required as deficiencies or shortcomings are discovered in the SRS. However, new requirements should not be introduced as part of the evolution process. All requirements in the SRS must be traceable to the FRS.

Prototyping

Prototyping may be used during the requirements portion of a project. Tools are available to allow a prototype to be developed that exhibits some of the characteristics of a system. A prototype is useful because it allows the customer to view the prototype and react to it more realistically than if he were to just read the SRS. The prototype may also display unexpected system behavior, so that the SRS and FRS, if necessary can be adjusted. Also, a SRS based on a prototype will undergo less change during development.

Embedded Design

The SRS should not be written to produce a specific design. Every requirement in the SRS will limit design alternatives to some extent. However, in some cases, it is necessary to limit the design. Examples include the need to keep certain functions in separate modules, to limit communications or to check data integrity for critical variables. Generally, the requirements should be stated from an external viewpoint. If models are used, it should be clarified that the model indicates the desired external behavior and does not specify a design.

Additional SRS preparation guidelines, including suggested format, are provided in IEEE Std 830.

Additional project documents developed during this phase will include:

- The System Interface Specification (SIS) defines the requirements of all interconnected systems, subsystems, and components interfacing with the project.
- The Requirements Traceability Matrix (RTM) provides a method of verifying and validating that all system requirements have been satisfied by the project. All system requirements should be depicted through all the development steps, then conclude with a verification or validation task.
- The Configuration Management Plan (CMP) is initiated in this process. The CMP identifies functional and physical characteristics to be controlled, specifies how changes to those characteristics are controlled, reports status of changes, and specifies means used to verify compliance with requirements

- The Software V&V Plan (SVVP) is initiated in this process. The SVVP describes conduct of software V&V during each lifecycle phase, defines reports to be generated, describes necessary administrative tasks such as error reporting, task iteration, etc., and defines documents to be produced in the V&V process. Format of the SVVP is prescribed by IEEE Std 1012.

5.4 Design Process

The objective of this session is to understand the design process, in which software requirements are transformed into an architecture and detailed design for each software component. The design includes databases and interfaces (external to the software, between the software components, and between software units). The design process, including inputs and outputs, is shown in Figure 5-19.

In the design process, the previously developed requirements specifications are interpreted into the software and hardware design of the upgrade project. Major decisions are made that determine the structure of the system.

The objective of the design process is to develop a coherent, well-organized representation of the system that meets the requirements specifications. This is accomplished primarily through the generation of the Software Design Description (SDD) and the Hardware Design Description (HDD). In order to meet the specified requirements, the design process must identify, evaluate and mitigate potential project risks.

5.4.1 Risk Assessment Methods and Techniques

During the design process, the risk presented by the design should be assessed periodically. This is done to ensure that the specified safety requirements have been met and that new risks have not been introduced. The goal is to identify the hazards of a system and to impose design requirements and management controls to prevent mishaps by eliminating hazards or reducing the associated risk to an acceptable level.

The risk management process is comprised of four activities:

- Assessment
- Elimination
- Mitigation
- Control

The assessment activity has three components:

- Identification
- Consequence
- Probability

Identifying hazards can be done in several ways. The most common are:

- Deductive (Probabilistic Risk Assessment)

The deductive method starts at the hazard and attempts to identify initiating events that can result in the hazard occurring. This path, from the general to the specific, is a Top-Down or Backward Search Approach. The Fault Tree Analysis or PRA is a typical example of this method.

The Fault Tree Analysis is a deductive method that searches from the front (Undesired effect) to the back (Initiating event). The method defines undesired states then determines what must occur for the undesired state to exist. The FTA examines the causes of events. The hazards must be identified prior to starting the FTA.

A simple Fault Tree Analysis, with the undesirable result (mission failure) at the top and potential causes of the event extending down, is shown in Figure 5-20.

- Inductive (Failure Modes and Effects Analysis)

The inductive method starts at initiating events and attempts to identify the hazards that can result. This path, from the specific to the general, is a Bottom-Up/Forward Search Approach. The Failure Modes and Effects Analysis (FMEA) and Preliminary Hazards Analysis are typical examples of this method.

The FMEA is an inductive method that searches from the back (initiating event) to the front (hazard). The FMEA is used primarily to assess reliability. The analysis looks at failure modes, then their effects. Component failure rates are gathered from databases then used to make predictions. Reliability is calculated by determining the combinations of events/failures that can render the system inoperative. Since software does not "fail," inclusion of predicted failure rates for software is questionable.

A simple Event Tree analysis, based on the Fault Tree analysis above, is shown in Figure 5-21. The Event Tree illustrates the inductive process used in a FMEA. In this example, the potential causes at the top of the diagram extend down to the undesirable event at the bottom.

Quantitative (Deductive) methods used by themselves to identify and classify hazards do not always yield a complete picture. For example, Nancy Leveson states in Safeware, that "35 percent of the actual in-flight malfunctions were not identified by the method [PRA] as 'credible'." Leveson further states, "Software itself does not fail, it is a design for a machine." Thus, methods used to quantify hazards resulting from software errors are questionable.

The priority for system safety is eliminating hazards by design. Therefore a qualitative (inductive) risk assessment procedure performed early in the design process that considers only hazard severity is generally sufficient to minimize risk. If reliable data are available, a combination of Deductive (top-down) and Inductive (bottom-up) will likely be the most successful in identifying hazards.

When hazards are not eliminated during the early design phase, a risk assessment procedure based upon the hazard probability, as well as hazard severity, is necessary to establish priorities for corrective action and resolution of identified hazards.

One method of classifying hazard severity and probability uses a matrix derived from MIL-STD 882B. The matrix provides a systematic method for assigning a hazard level to an identified failure event based on the severity and frequency of the event.

As shown in Figure 5-22, the hazard level consists of one number and one letter. The number represents the Severity of the Event. The letter of the hazard level represents the Frequency of Occurrence. As can be seen from the table, each hazard level is associated with a risk category. Risk categories assist risk-management team members in differentiating credible high-hazard threats that may result in loss of life and property from less probable risks, therefore aiding management in risk vs. cost decisions.

IEEE Std 1012-1998 describes a similar method to identify Software Integrity Level (SIL). This classification is shown in Figure 5-23.

5.4.2 Critical Digital Review (CDR)

As described in EPRI TR-107339, Appendix A, the critical digital review is a process to provide assurance of the integrity of the overall digital system (including hardware and software). The following discussion is intended to provide an introduction to the CDR concept rather than provide sufficient informa-

tion to conduct a CDR. For a Regulator conducting an inspection, the findings of a previously performed CDR can provide valuable assistance in assessing the consequences of an unforeseen event. If a CDR was not previously performed, the methodology may be used to facilitate the inspection. In these situations, the Regulator should consult EPRI TR-107339, Appendix A. Depending on the severity of the consequences, it may be advisable to engage a consultant specializing in the CDR.

5.4.2.1 Basics

The CDR supplements the software V&V and hardware quality control normally implemented by a manufacturer. Software V&V can demonstrate that the software was developed in a defined, systematic fashion. However, all software is assumed to have design defects ("bugs") and all hardware will eventually fail. The CDR evaluates the potential consequences of digital device or system malfunction and relates the consequences to probable ways the malfunction can occur. A quality design will incorporate means to identify, prevent or mitigate undesirable consequences of unforeseen events.

Unforeseen events are not limited to the physical system. A common software failure in a redundant system or lack of proper documentation to configure a component required for continued plant operation when the vendor does not have support available are examples of unforeseen events.

The CDR itself is not an audit or survey. The CDR is a technical evaluation. The audit or survey is a Quality Assurance Activity. There is some overlap in the activities performed in the CDR, audit or survey. CDR findings may be used as input to an audit or survey.

If performed, the CDR is conducted normally during the design phase of the digital project, although planning for the CDR will take place during the concept phase. The Regulator will likely not be involved at this time. More likely, the Regulator will be conducting an inspection to determine the cause or consequences of an unforeseen event that has already happened. EPRI TR-107339, page A-7 provides a list of questions that may be used by the Regulator as a starting point to assess root causes for unforeseen events that may have occurred.

Deliverables of the CDR include:

- List of recommended action items outlining potential problems and mitigation options;
- Descriptions of the digital system architecture and functional characteristics, including development processes;
- Discussions of important technical details.

The primary objective of the CDR is to penetrate the "technical shell" of the digital portion of the system, as shown in Figure 5-24, adapted from EPRI TR-107339. The goal of this penetration is to gain a detailed knowledge of the core technical architecture. The knowledge of the core technical architecture forms the basis for questioning relevant requirements, expectations and potential for unforeseen events. Penetrating the technical shell requires technical expertise as well as judgment of what is relevant to the digital system.

The CDR can assist the engineer in identifying the potential for obscure behavior and unique failure modes in a digital upgrade, even if it is designed to provide exactly the same functionality as its analog predecessor. If an undetected or unrecognized failure in the same system can cause or enable significant consequences, the CDR can help identify potential

unforeseen events and mitigation strategies. If the potential consequences are not significant, an audit or survey of vendor practices may be more appropriate than a CDR.

5.4.2.2 Assumptions

There are five underlying assumptions of the CDR process as described in EPRI TR-107339:

1. **A CDR is not self-contained.** Followup is required to ensure action items are resolved and mitigative measures are implemented.
2. **Almost all software has bugs; almost all hardware will fail.** In spite of best efforts, failures will occur. Some will be due to software errors. Others may be manifest by hardware failure, but the consequences are made worse by software errors. If there are no serious consequences, a CDR is probably not necessary. If serious consequences can occur, the CDR offers a way to examine the potential for bugs and failures and their mitigation.
3. **A digital system is comprised of a platform and an application.** The platform is the generic base used for application development. The application is the means by which the platform is adjusted or configured to meet specific project requirements. See Figure 5-25. The platform can be very simple, such as a single circuit board based microprocessor, or massive, such as a plant wide distributed control system. If the platform has received a SER, the CDR can focus primarily on the application. It is important to evaluate the differences in behavior between the existing analog or non-sequential discrete hardware application and the new digital platform/application. A non-sequential discrete hardware logic application will move smoothly between states unless there is a malfunction. A digital system using a PLC may require several scans to complete the transition, and may pass through invalid states. These invalid states may not affect the output device, but improper means of detecting the invalid states can lead to common-cause failures.
4. **The "System" consists of the digital system, vendor processes and the project perspective.** Vendor perspective is the scope, formal requirements and informal expectations of the project. Informal expectations are often unstated and simply assumed by project team members. The assumptions may be valid for those familiar with the plant systems where the system will be applied, but the software developer may not understand the subtleties of a particular application. The unstated project perspective can limit the effectiveness of a CDR. If:
   - There are high-consequence events that may be caused by an unusual failure
   - The review team does not have sufficient technical access
   - The platform will not be widely used in future projects

   then it is not likely that a CDR will rule out the possibility of a future event caused by an unusual failure in the system. If it is recognized early that the focus for mitigation is the project perspective (and not the system itself), the system may be more complex than necessary. A simple manual bypass may be the most cost-effective means of mitigation.
5. **Software development processes rarely**

function as described on paper. The system developer may have impressive written procedures, but the procedures may not be followed in practice. It is necessary to work with the people doing the actual development and V&V tasks to see how the plans are being used.

A good application resting on a bad platform will yield a poor system, as will a bad application implemented on a good platform.

It is necessary, particularly for a Regulator conducting an inspection where unforeseen circumstances have occurred, to look at the "real system," which consists of the platform, the application and the plant perspective.

The plant perspective includes scope, formal (written) requirements, informal expectations and unstated assumptions. Expectations and assumptions are sometimes considered "too obvious" to require formal documentation. Unfortunately, the expectations and assumptions may not be so obvious to the system developer, and the result can be vulnerability to unforeseen events.

5.4.2.3 Components

The critical digital review follows a four-step process:

1. **System Orientation.** The purpose is to gain an overview of the supplier's system architecture.
2. **Process Orientation.** This step examines the supplier's policies, procedures and standards for development, documentation testing and maintenance of the product. This includes record keeping, failure investigation and customer complaint handling.
3. **Thread Analysis.** Follows specific functions through documentation, testing and implementation. Thread analysis in I&C systems frequently includes tracing signals through data acquisition hardware, software and display. It is necessary to dig deep enough into the technical core of the system to get a clear understanding of potential failure modes. Typical questions include:
   - Are vendor procedures followed by the staff?
   - Are technical reviews performed regularly?
   - Is there a well maintained design basis?
   - Are customers notified of errors?
   - Does the digital system/software architecture support digital system integrity?
   - Does the digital system/hardware architecture support digital system integrity?
   - How does the digital system fail?
4. **Risk Analysis**. This is the final CDR phase. The analysis is done qualitatively, and will include a fault tree and failure modes and effects analysis. The qualitative failure modes analysis postulates failures of hardware modules, unintended software behavior, human errors and field device failures.

5.4.3 Development of the Software Design Description (SDD)

The objective of this lesson is to assist the Regulator in understanding how the SDD document identifies and details the methods of accomplishing the intended functions, and what to look for in performing an inspection. The SDD is an output of the design activity

shown in Figure 5-19. As described in IEEE Std 1016, "Recommended Practice for Software Design Descriptions," the SDD enumerates how the software system will be structured to satisfy the requirements specified in the software specification in accordance with IEEE Std 830. It is a translation of requirements into a description of the software structure, software components, interfaces and data necessary for the implementation phase.

The SDD represents a partitioning of the system into design entities and describes the properties and relationships of the entities as a standard set of attributes. Design requirements are satisfied by identification of the entities and their attributes.

The formal SDD outline in IEEE 1016 is well suited to traditional programming methods and languages (such as FORTRAN, C++, etc.). It is less well suited to the graphical (object-oriented) methods described in IEC 61131, such as ladder diagram, function block diagram and sequential function chart. The SDD for an object-oriented application program should contain all the information prescribed in IEEE-1016, but need not conform to the format.

5.4.3.1 Design Entities

A design entity is an element or component of a design that is structurally and functionally different from other elements and is separately named and referenced.

Design entities result from decomposition of the system software requirements. The objective is to divide the system into separate components that can be considered, implemented, changed and tested with minimal effects on other entities.

Entities can exist as a system, data stores, program, module, or process that possess common characteristics such as interfaces or shared data. The common characteristics are described by design entity attributes.

5.4.3.2 Design Entity Attributes

The design entity attribute is a named characteristic or property of a design entity. Attributes are questions about design entities. Answers to the questions are the values of the attributes. The collection of answers provides a complete description of the entity.

Attributes are selected according to the following criteria:

- The attribute is necessary for all software design projects
- An incorrect specification of the attribute could result in a fault in the software system to be developed
- The attribute describes intrinsic design information and not information related to the design process such as designer name, design status or revision history.

IEEE Std 1016 requires the following ten attributes to be specified for each entity. The attribute description should include assumptions and tradeoffs. If an attribute is not applicable, its description may state "none."

1. Identification

This is the unique name for the entity. The name may be chosen to characterize the function of the entity.

2. Type

This is the nature of the entity, such as subprogram, module, and process or data store. It may also designate a major class of entities to assist in locating

an entity that deals with a particular type of information.

3. Purpose

This attribute describes the specific purpose for creating the entity. It describes the functional and performance characteristics, which are satisfied by the entity. If there are special requirements for the entity that are not listed in the SRS, they are stated in this attribute.

4. Function

This attribute describes what the entity does. That is, it states the transformation that is made on the inputs to produce the desired outputs.

5. Subordinates

If there are other entities that compose this entity, they are listed in this attribute. This information is used to trace requirements to design entities and to identify parent/child relationships. The subordinates attribute identifies the "composed of" relationship with another entity.

6. Dependencies

This attribute describes the relationships of this entity with other entities by identifying the "uses" or "requires the presence of" relationship with other entities. This attribute describes the nature of each interaction including timing and conditions required for interaction such as initiation, order of execution, data sharing, etc.

7. Interface

This attribute describes how other entities interact with this entity, including methods and rules for the interaction. Methods of interaction include how the entity is invoked or interrupted and means by which it communicates with other entities such as common data areas. The attribute describes input ranges, definitions of inputs and outputs and output error codes. For information systems, the attribute includes inputs, screen formats, and a description of the interactive language used to develop the interface.

8. Resources

These are the physical resources, external to the software design, that are used by the entity. This includes devices such as printers, disk drives, math libraries, and processing resources such as CPU cycles and memory allocation.

9. Processing

Processing is a description of the rules used by the entity to perform its function. This attribute describes the algorithm used by the entity and includes contingencies in the event of overflow conditions or validation check failure. It should include timing, sequencing of events or processes, process steps, etc. It is the most detailed attribute for this entity.

10. Data

This attribute describes the data elements internal to the entity. It lists method of representation, use, initial values, format and acceptable values. The description should include whether the data elements are static or dynamic, whether it is shared by other transactions, if it is a control parameter or used as a value, loop iteration counter or pointer.

The description may be in the form of a data dictionary that describes the content, structure and use of all data elements used by the entity.

5.5 Implementation Process/Performance Issues

5.5.1 Implementation Process

The objective of this session is to understand how the Implementation Process transforms the project design into actual hardware and software code, and

develops site-specific detailed design documentation. In this process, the project hardware is assembled and integrated with the software for acceptance testing. Project design drawings are finalized.

The following representative documents and activities, illustrated in Figure 5-26, are developed during this phase to support system hardware/software integration and testing:

- Software source code listings are created and base lined for all application software developed for the Project.
- Hardware related drawings
- Any required Operation and Maintenance manuals for the final functioning system will be generated during this process.
- System surveillance requirements and methods will be identified and documented during this process.
- All training requirements for the installation, operation, calibration, or maintenance of the final functioning system will be identified and documented during this process.

The following are representative documents and activities developed during this phase to support the preparation of the modification package utilized for the system installation:

- Design Input Report (DIR)
- Engineering Change Notices (ECNS)
- Calculation updates for Electrical, HVAC and Combustible Loadings,
- Updated equipment database, circuit schedule and other applicable databases and lists as required
- 10 CFR 50.59 Screening

5.5.2 System Performance Issues

Digital sampling of analog signals introduces two types of error, aliasing and finite wordlength, to the sampled version of the signal. NUREG-1709,"Selection of Sample Rate and Computer Wordlength in Digital Instrumentation and Control Systems," provides regulatory background, theoretical and practical information and best engineering practices associates with sample rate and wordlength selection. This lesson section summarizes NUREG 1709.

Digital I&C systems manipulate binary numbers. Therefore, they must convert analog signals to binary numbers, and in some cases, convert binary results back to analog signals. There are two key factors that are important to the performance, reliability, and safety of digital I&C systems, and that are not issues in analog systems:

1. **Sample rate**: The rate at which an analog signal is measured (sampled) its instantaneous value is converted to the binary equivalent; and
2. **Computer wordlength**: The number of discrete binary bits used to represent a numerical value of the sampled signal or internal variable

When a digital I&C system samples an analog signal, it gets a snapshot of the signal at discrete time intervals. If the analog signal changes rapidly and the sample time intervals are not small enough, the sampled version of the signal will misrepresent high frequency components of the original signal as low frequency components. This type of signal corruption is called aliasing. Refer to Figure 5-27 and Figure 5-28

Due to finite wordlengths, mathematical operations such as addition and multiplication introduce

round off and/or truncation errors. If finite wordlength errors are not properly addressed in digital I&C systems, they may cause unexpected behavior. Refer to Figure 5-29.

In control systems, aliasing or severe finite wordlength errors may cause instability. In monitoring, alarm, and protection systems, such conditions may degrade performance. In protection systems, aliasing and finite wordlength errors may adversely affect setpoint accuracy and response time requirements. However, through proper sample rate and computer wordlength selection, these error sources can be minimized to a point where they have an insignificant effect on the system.

The rate selected for a particular signal depends upon its rate of change (frequency content of the signal). Shannon's Sampling Theorem states that a signal must be sampled two times faster than the signal's highest frequency component to reconstruct the signal in the time domain. This theorem defines a theoretical minimum sample rate to prevent aliasing.

$$If:$$
$$f_o = original\_signal\_frequency$$
$$f_s = sample\_frequency = 2 f_o$$
$$f_a = alias\_frequency = f_s - f_o$$
$$Then:$$
$$f_a = f_s - f_o = 2 f_o - f_o$$
$$f_a = f_o$$

When practical issues, such as signal noise, are considered, the sample frequency is greater than two times the highest frequency of the analog signal. Calculation of the minimum sample rate depends upon the following:

1. **Application:** control, monitoring, protection, or indication
2. **Environment:** signal noise
3. **I&C equipment:** input signal filters, A/D converters, and other interfacing computer equipment
4. **Interfacing systems:** actuators and dynamics of the plant process

Sample rate selection methods vary, depending upon the type of digital I&C system. Types of systems include discrete, open loop, and closed loop I&C systems.

Discrete I&C systems deal with input signals taking on one of two values, and the input signals often come from discrete devices such as relays, bistables, etc. Discrete I&C systems do not have aliasing problems, but the system response time depends on the sample rate.

Open loop I&C systems do not have feedback signals and include protection, monitoring, alarm, and some control systems. Three sample rate selection methods are commonly used with open loop I&C applications. Two of the methods, the sampling ratio method and the oversampling method, are concerned with meeting a maximum allowable aliasing level, and they are best suited for those systems requiring signal accuracy. A third method, the rise time method, is suitable for open loop I&C systems that do not have stringent signal accuracy requirements.

The sampling ratio method uses an analog low-pass filter placed in front of the A/D converter. The filter prevents high frequency signal components or noise from reaching the A/D converter. The sampling rate is selected using a relationship between the analog signal bandwidth and the frequency of required attenuation. For additional information, refer to NUREG-1709.

The oversampling method samples the analog input signal at a rate much higher than the Shannon

theorem would suggest. Antialiasing filters are used frequently with this method to prevent high frequency signal components or noise from affecting the required accuracy.

The rise time method samples the analog input signal at a rate sufficient to evaluate the signal a fixed number of times during the rise time of the signal. The rise time is defined as the time required for a step change in a signal to transition from 10% to 90% of its final value. In Figure 5-30 the rise time from signal value = 0.1 to 0.9 is 2 seconds. If it is desired to sample 5 times in this interval, the sample rate is 2/5 = 0.4 sec = 2.5 Hz.

Closed loop I&C systems have at least one feedback signal. There are three sample rate selection methods available to closed loop I&C systems: the phase/gain margin method, the closed loop bandwidth method, and the rise time method. These methods are mainly concerned with system stability.

The phase/gain margin method may be used to evaluate a selected sample time for control system stability where there is a mathematical model for the closed-loop control system. Using the control loop's open-loop frequency response, the gain margin is the amount of gain added from a value of 1.0 before instability occurs at a phase shift of -180°. Phase margin is the amount of phase shift added between its value at a gain =1 before instability occurs at phase shift =-180°. See Figure 5-31. If the phase/gain margin meets control system requirements, then the sample rate is sufficient for stability.

The closed loop bandwidth method may be used where the system does not have phase/gain margin requirements. Although Shannon's theorem would suggest a sampling rate greater than twice the signal bandwidth, good engineering practice is to use at least six times the bandwidth. This practice will enable the digital control system response to approach that of an analog control system. As before, it is good practice to precede the A/D converter with a low pass filter to prevent high frequency process signal components or noise from being introduced to the control loop.

Finite wordlength errors occur when real number data are represented by a finite number of bits in a computer system. These errors occur at input signal acquisition, intermediate calculations, the output signal, and algorithm coefficients. For example, finite wordlength errors are introduced at A/D conversion. The accuracy of the conversion is impacted by the A/D converter's dynamic range. As the converter covers a wider range of input values and resolves to smaller voltage levels, the dynamic range increases. The dynamic range is affected by the wordlength of the A/D converter and its associated error specification.

Intermediate calculations are affected by finite wordlengths of computer memory. Errors associated with intermediate calculations include round off/ truncation error, overflow, and incorrect type conversion. Overflow occurs during addition when the result occupies one more bit than the available storage space. Incorrect type conversions may occur when numbers are converted between two different numbering conventions (i.e., fixed- and floating- point notation).

In many applications, digital I&C systems convert digital results into analog signals. The conversion is carried out using a digital to analog (D/A) converter. Because the value of the digital signal is not known between updates, the D/A converter accepts a digital value and holds it at a representative DC voltage until the next update arrives. This creates a step-like analog signal that may not be acceptable to some plant systems.

Often, when algorithms are developed for digital I&C systems, real number coefficients are used in the design. When the coefficients are placed into the digital I&C system, truncation or round off of the

coefficients may occur, potentially degrading system performance.

Computer sample rate and wordlength selection is based upon design considerations that take their bases from fundamental engineering principles.

NUREG-1709 contains a list of design considerations that summarize the above discussions regarding sample rate and wordlength. The list is convenient for a Regulator to use in reviewing a design.

5.6 Testing Activities

The objective of this session is to understand the testing that is performed throughout the upgrade process to ensure the adequacy of the design and its implementation. The Software Verification and Validation Plan (SVVP), developed during the Concept process, identifies the testing activities that will be performed during the development phase. Thus, the SVVP serves as a roadmap through the testing activities.

5.6.1 Requirements Process Testing Activities

During the Requirements Process, the System V&V Test Plan is generated to include tracing system requirements to test designs, cases, procedures and results. The V&V Test Plan addresses compliance with all system requirements, adequacy of user documentation and performance at boundaries under normal and stressed conditions. The System V&V Test Plan provides test coverage of all system requirements, appropriateness of test methods and standards, conformance to expected results, feasibility of system qualification testing and feasibility and testability of operation and maintenance requirements.

The Acceptance Test Plan (ATP) is generated during the Requirements Process, as shown in Figure 5-16. The ATP is intended to ensure that developed software correctly implements system and software requirements in the operational environment, in contrast to the development environment of the System V&V Test Plan. The Acceptance Test Plan covers system requirements, documents conformance to expected results and demonstrates feasibility of operation and maintenance to meet user needs.

5.6.2 Design and Implementation Process Testing Activities

Testing activities in the design and implementation processes also involve subjecting the completed project software and hardware to testing both prior to and after shipment to the utility plant. Testing is necessary to verify that the hardware and software components developed for the project satisfy system requirements, and to verify that the requirements have been properly integrated. As shown in Figure 5-19, the outputs for Design Process test activities consist of plans to be executed in the Implementation Process.

Software Component (Unit) Test Plans

Software Unit Test (SUT) Plans describe the testing of individual project software units prior to full system integration.

Component (Unit) testing is conducted to verify the correct implementation of the design and compliance with program requirements for one software element (e.g. unit, module) or a collection of software elements.

System Integration Test Plan

The System Integration Plan (SIP) formalizes the order and requirements for the correct integration of all

project hardware and software components into the final functioning system.

Integration testing is an orderly progression of testing of incremental pieces of the software program in which software elements, hardware elements, or both are combined and tested until the entire system has been integrated to show compliance with the program design, and capabilities and requirements of the system. System Integration Test(s) may be performed by the supplier to verify that individual software units satisfy their intended functions as an integrated software system. This test may or may not be performed with the software loaded on the actual plant equipment.

Acceptance Test Plans

Hardware and Software Acceptance Test Plans outline the acceptance criteria for project hardware and software. These plans are generated after base lining the HDD and SDD against their requirements specifications.

Acceptance phase testing is conducted in an operational environment according to the Acceptance Test Plan to determine whether a system satisfies its acceptance criteria (i.e., initial requirements and current needs of its user) and to enable the customer to determine whether to accept the system. Factory Acceptance Testing (FAT) will usually be performed at the supplier's facility and will be conducted in accordance with an approved FAT Procedure generated by the supplier. The FAT is designed to demonstrate system conformance to the requirements as established by the Hardware and Software Acceptance Test Plans. The FAT should be performed in a configuration that emulates (as closely as possible) the actual plant configuration.

During the Implementation Activity, test cases and procedures are developed for the previously generated plans, as shown in Figure 5-26. Component testing as also conducted at this time.

Finally, the integration and acceptance tests are conducted during the test phase activity as shown in Figure 5-32.

Following completion of FAT, a FAT Report is generated by the supplier to document the results of all testing activities performed. Any testing failures or anomalies observed during testing are described in the report along with the resolution to the identified failure or anomaly. If the FAT is performed at the supplier's test facility, the FAT Report must be approved by the utility representative prior to shipment of the upgrade/conversion system to the plant site. If the FAT is performed at the plant site, the project representative must approve the FAT Report prior to additional testing or installation work. The FAT Report is included (attached to) the final Software V&V Report (SVVR).

5.6.3 Installation and Commissioning Testing Activities

These activities involve the installation of the project equipment at the plant site and performance of subsequent post-installation testing activities necessary to declare the system operational.

Prior to installation, Site Acceptance Testing (SAT) is performed by plant personnel using an approved SAT Procedure that demonstrates system conformance to all requirements as specified in the various specification documents (FRS/SRD, HRS, SRS). The SAT should be performed in a test bed configuration, rather than connected to actual plant equipment.

Following completion of SAT, a SAT Report is generated by the project Verification and Validation Engineer (VVE) to document the results of all testing

activities performed. Any testing failures or anomalies observed during testing are described in the report along with the resolution to the identified failure or anomaly. The SAT Report is included (attached to) the SVVR.

Following satisfactory completion of the SAT and after physical installation in the plant, a Post Installation/Modification Test (PIT/PMT) is conducted and documented according to approved procedures. The purpose of the PMT is to demonstrate that the system will perform its intended function in the real physical environment using real plant sensors and actuation devices. If the system is non-safety related, development is now complete, and the Operating and Maintenance lifecycle phases begin.

If the system is safety-related, the final step in the development phase is Return to Service (RTS) testing that verifies that the system meets its licensed requirements as described in the plant Technical Specifications. Following completion of RTS testing, the system is accepted for use by the plant Operations department and the Operating and Maintenance lifecycle phases begin with regularly scheduled Surveillance Testing.

5.7 Software Training Plan and Implementation

The objective of this session is to provide the Regulator with an understanding of the training plan characteristics discussed in NUREG 0800, to be used in evaluating plans developed by the utility for the software project.

The software training plan should exhibit the management, implementation, and resource characteristics listed below, as discussed in NUREG 0800, Section 7, Appendix 7-A (BTP HICB-14).

5.7.1 Management Characteristics

The management characteristics that the software training plan should exhibit include purpose, organization, and responsibilities.

Purpose requires a description of the means necessary to ensure that training needs of appropriate plant staff, including operators and I&C engineers and technicians, are fully achieved. The plan should include a general description of the training facilities.

Organization requires a description of the software training organization. The interfaces between the training organization and the project management organization should be described. Reporting channels should be described. Trainers should have the necessary knowledge of the software operation to ensure that trainees understand its operating and maintenance requirements.

Responsibilities requires a definition of the responsibilities and authority of the training organization and training by customers.

5.7.2 Implementation Characteristics

The implementation characteristics that the software training plan should exhibit include measurement and procedures.

Measurement requires a set of indicators used to determine the success or failure of the training effort. The plan should require that training data be collected and analyzed to determine the effectiveness of the training effort. The trainee error rate found at the end of training activities should be measured, recorded, analyzed and reported.

Procedures requires a description of the training procedures. The plan should list any documentation required to support the training effort. The training

program should be described. The plan should require that training be specific to different job functions. Training products and reports should be described. Reporting requirements should be specified.

5.7.3 Resource Characteristics

The resource characteristics that the software training plan should exhibit include methods/tools.

Methods/tools requires a description of the methods, techniques and tools that will be used to accomplish the training function. Training should be carried out on a training system that is equivalent to the actual hardware/software system.

5.8 Operations and Maintenance

The objective of this session is to understand how the software project is handled after it has been accepted for use and placed in operation in the utility's plant.

IEEE Std 610.12, "Standard Glossary of Software Engineering Terminology," defines the Operation and Maintenance Phase as "The period of time in the software lifecycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements."

5.8.1 Operations Support

The Operation Support activity, as described in IEEE Std 1074, "Standard for Developing Software Life Cycle Processes," covers the operation of the software product and operational support to users. This process also includes the performance of site surveillances as required. Support includes consulting with the user and maintaining a Support Request Log. If problems are determined or operating requirements change, the software maintenance organization initiates the required activity and the product re-enters the software lifecycle process.

The Operation V&V activity evaluates the impact of any changes in the intended operating environment, assesses the impact on the system of any proposed changes, evaluates operating procedures for compliance with the intended use, and analyzes risks affecting the user of the system.

5.8.2 Maintenance Support

Maintenance support covers modifications, migration and retirement of software. Migration is the movement of software to a new operational environment. Retirement of software is the withdrawal of active support by the operation and maintenance organization, partial or total replacement by a new system, or installation of an upgraded system. The Maintenance activity is activated when the software code and associated documentation are revised to correct a problem or address a need for improvement or adaptation. As discussed in IEEE Std 1074.1, "Guide for Developing Software Life Cycle Processes," software maintenance refers to all the technical and management activities related to modifying a software system after it has been placed in operation. Maintenance may involve any or all of the following:

- Corrective maintenance: Identification and correction of software errors, performance failures, and implementation problems.
- Adaptive maintenance: Modifications to permit the software system to run in a different operating environment, or with different types of data.
- Perfective maintenance: Maintenance to incorporate new requirements, enhance performance,

improve cost-effectiveness or otherwise improve the software system.

Any software maintenance effort involves, at a minimum, the following steps:

1. Understand the existing system
2. Understand the problem or the desired improvement
3. Modify the system in a specified manner
4. Revalidate the updated system
5. Update documentation.

Steps 2 through 4 are analogous to the requirement and design, implementation and validation processes of software development. Thus, software modifications are treated as development processes and are verified and validated as described in Sections 5.2 through 5.6.

This process includes the maintenance of a Software Support Log, which should be created for each digital development project that produces modifiable software and should be included in the Software Configuration Package.

The Software Support Log documents support activities that lead to the need to perform software maintenance:

- Identify software improvement needs. This activity identifies lessons learned and needs to improve the software. The output of this activity consists of recommendations. The recommendations include their impact on the quality of the software, as well as any tools or methods required to implement the recommendations.
- Implement a problem reporting method. This activity accepts anomalies and prepares a problem report, and may include possible solutions. Problems can be resolved through corrections or enhancements as discussed above. Corrections and enhancements are documented. Enhancements can be considered for future projects. This activity analyzes the problem and makes the following determination:

1. Nature of the anomaly
2. Source and cause of the problem
3. Product that contains the error
4. Severity
5. Corrective action
6. Impact on cost, schedule and risk

- Reapply the software life cycle as appropriate. Information developed above is used to generate recommendations for maintenance that reenter the software lifecycle, usually at the requirements phase, where existing requirements are modified. This activity then monitors correction of the problem through the software lifecycle and documents completion in the Software Support Log. If the impact is major, such as affecting the fundamental system concept or adding or deleting major requirements, a new project should be considered.

EPRI TR-102348 - Overall Approach

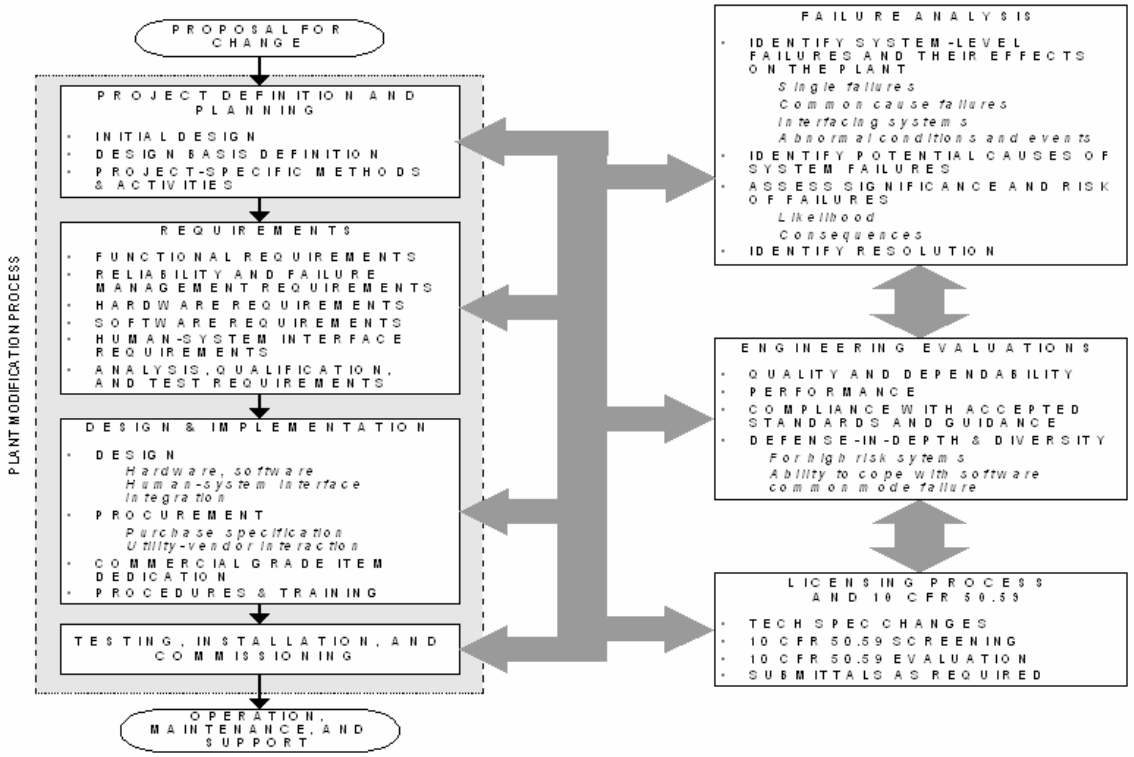


Figure 5-1 (Slide 5.2.1-5)

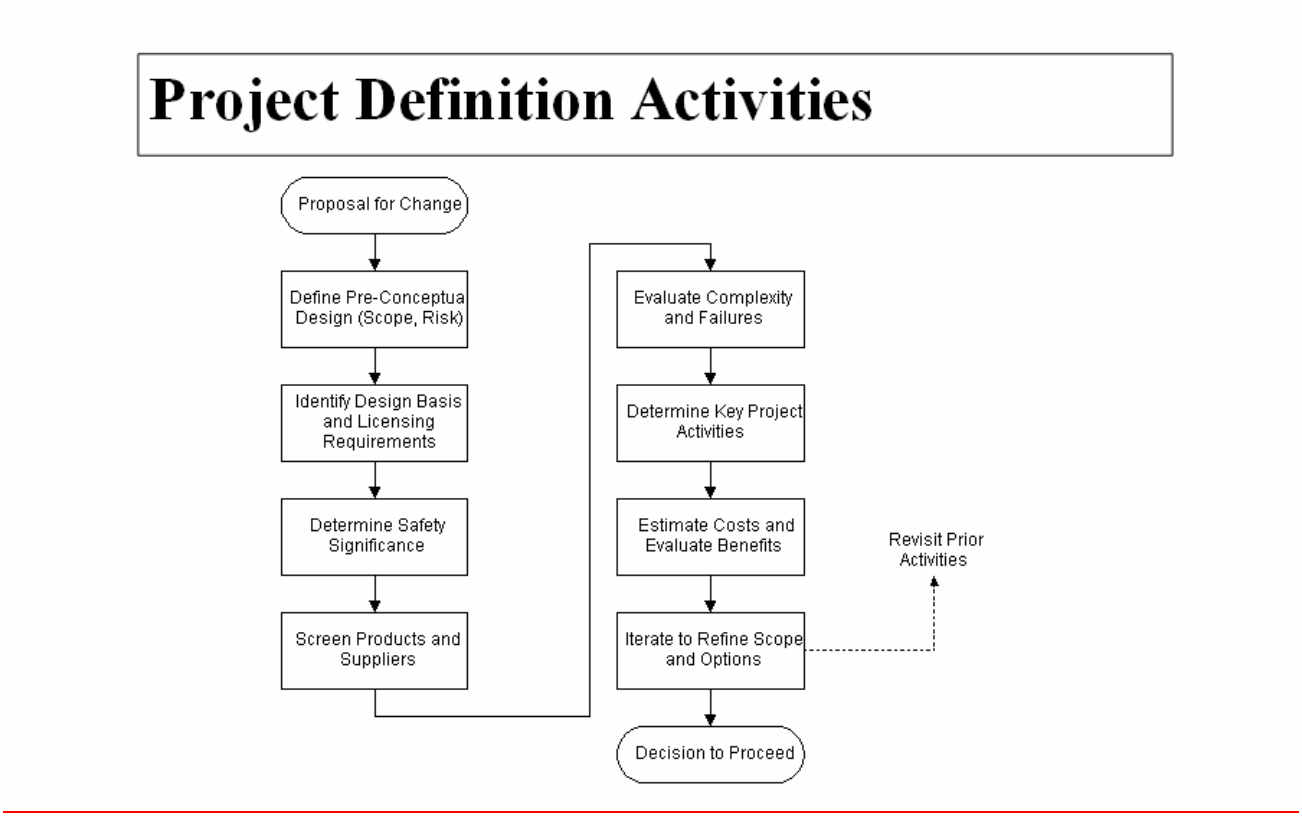


Figure 5-2 (Slide 5.2.1-6)

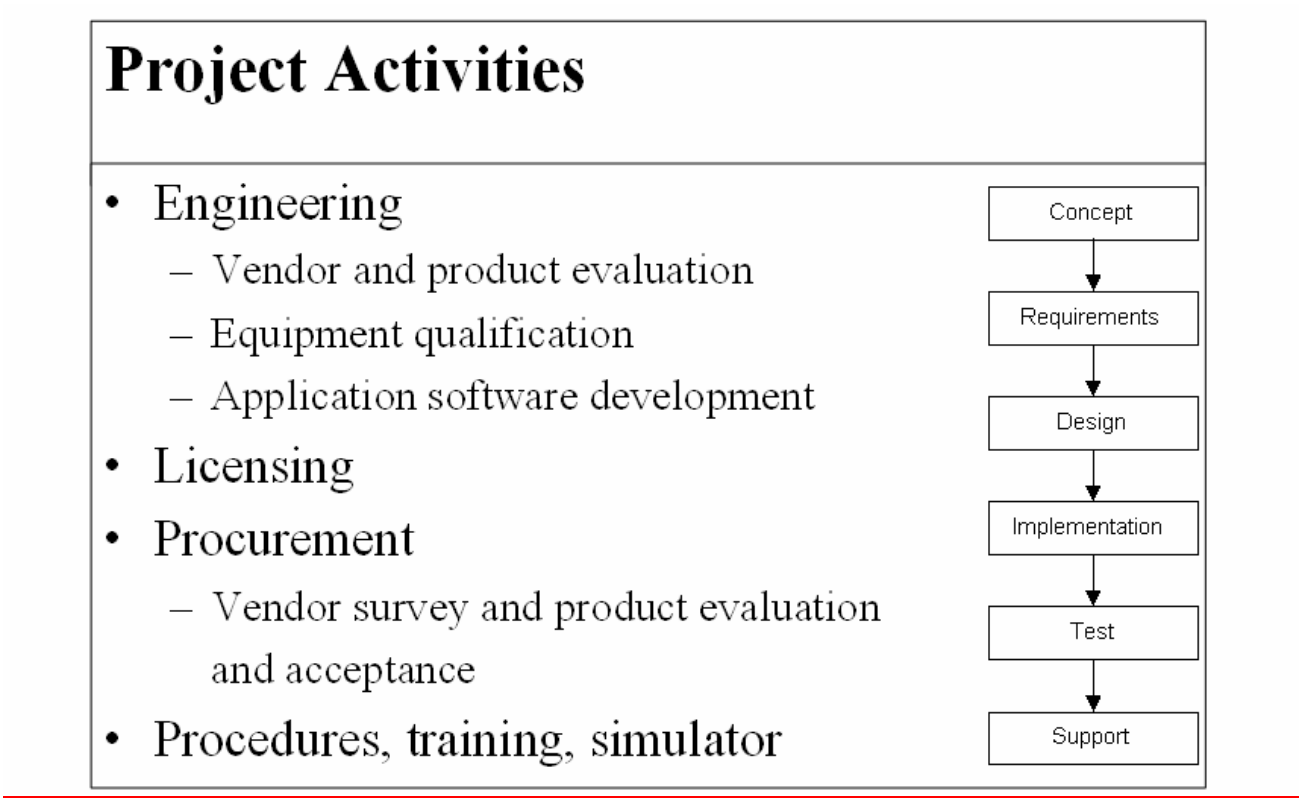


Figure 5-3 (Slide 5.2.1-13)

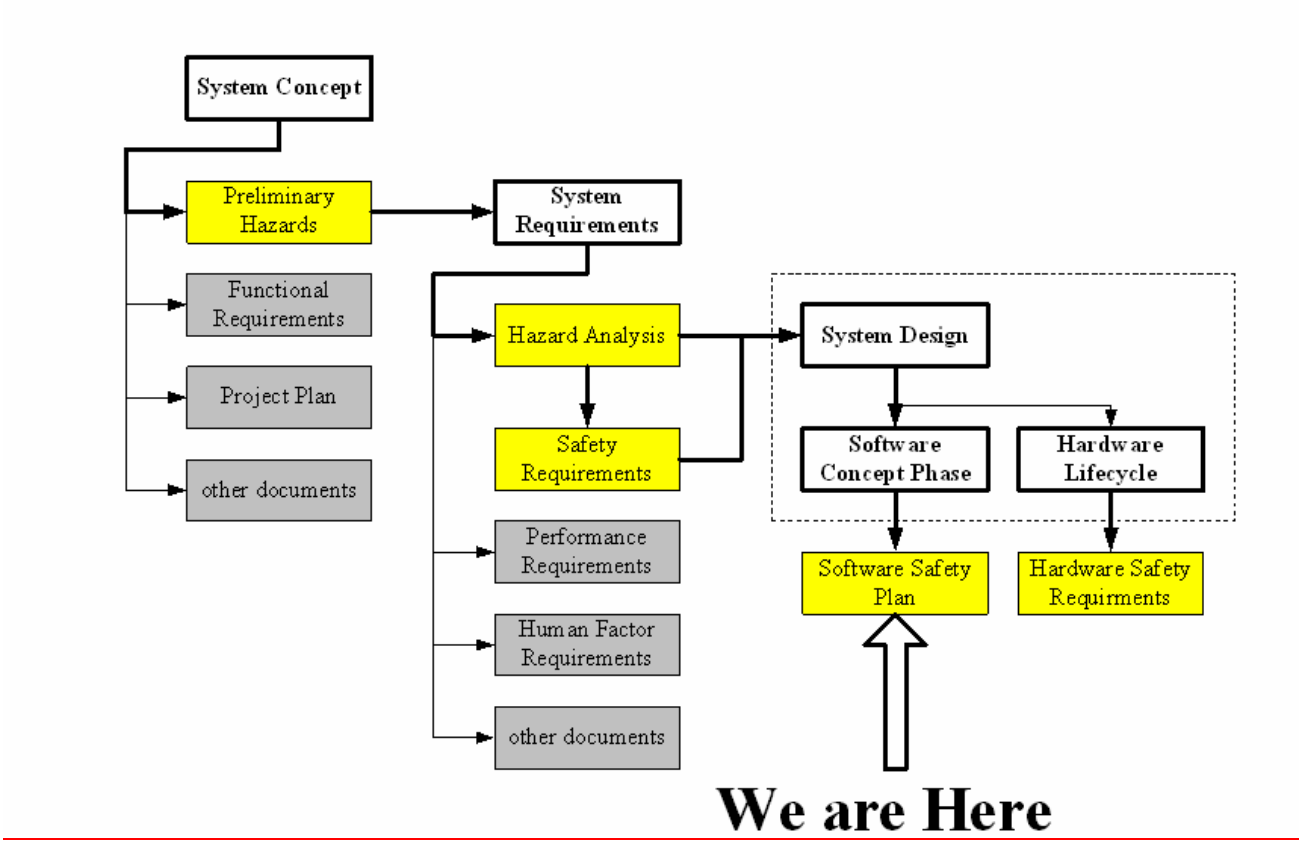


Figure 5-4 (Slide 5.2.2-13)

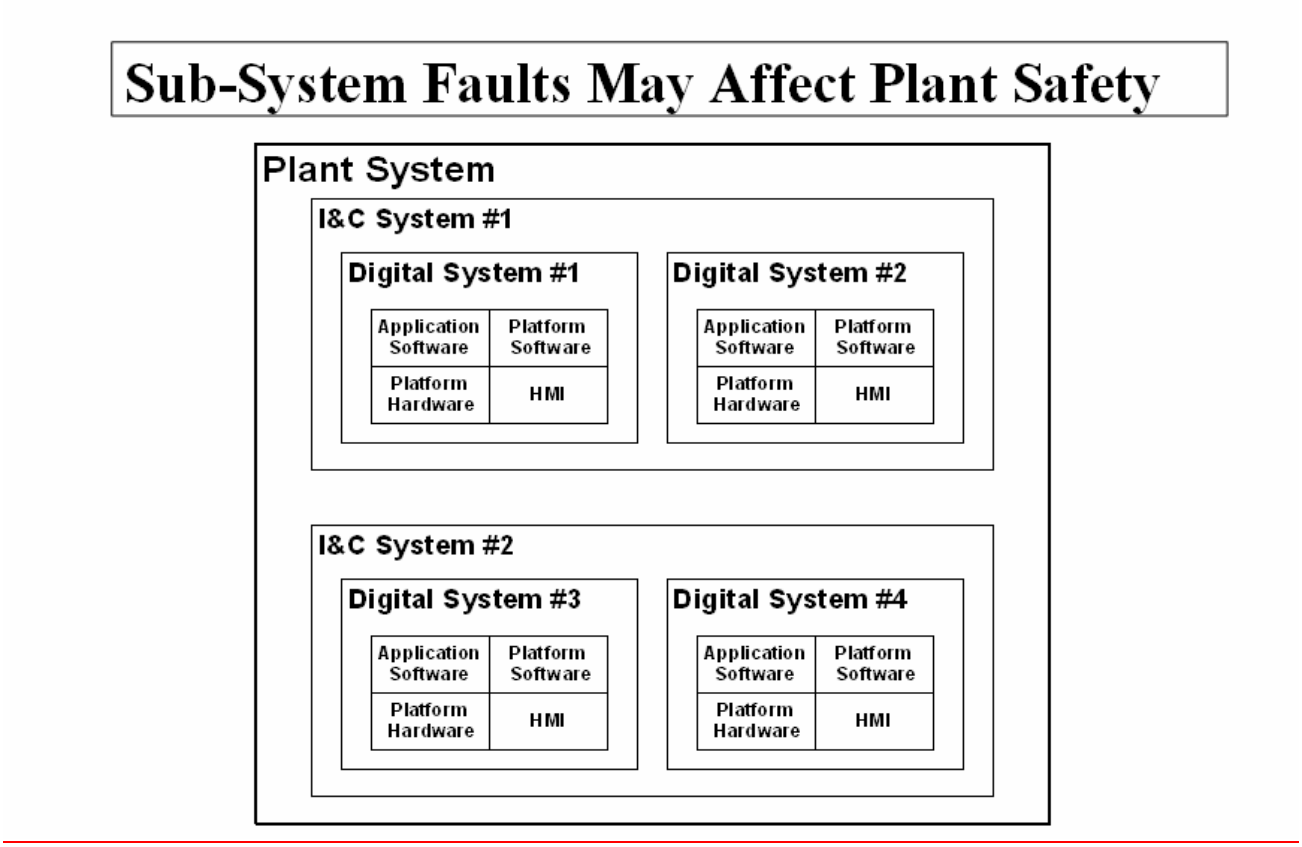


Figure 5-5 (Slide 5.2.2-21)

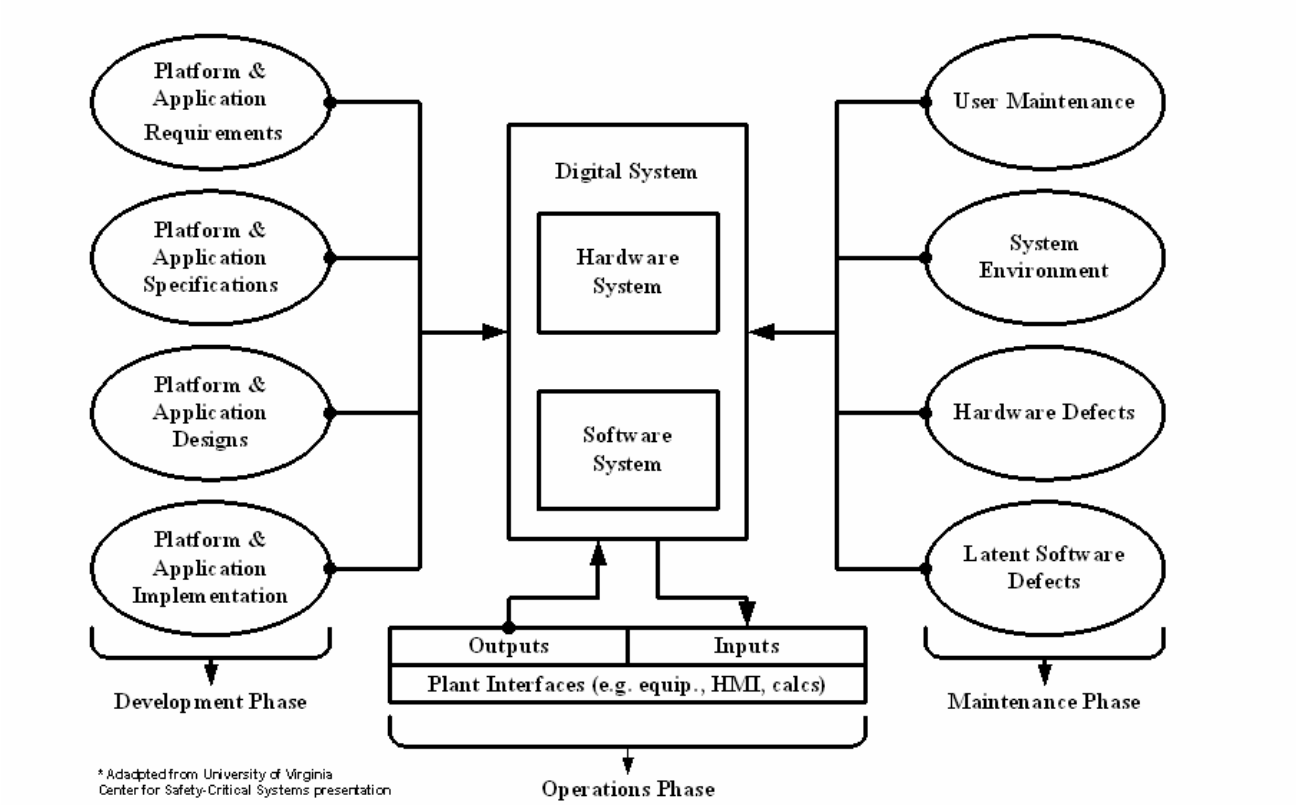


Figure 5-6 (Slide 5.2.2-22)

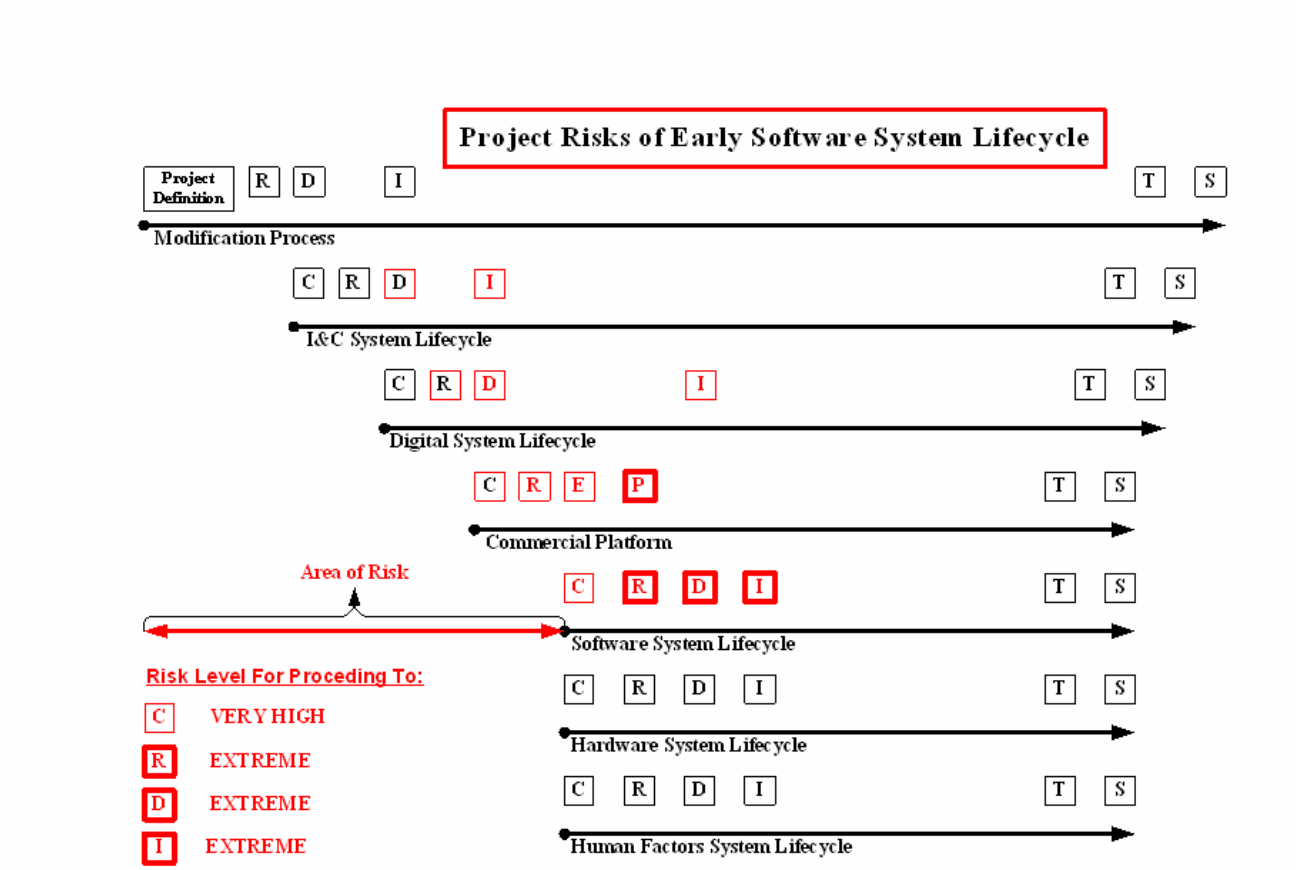


Figure 5-7 (Slide 5.2.2-23)

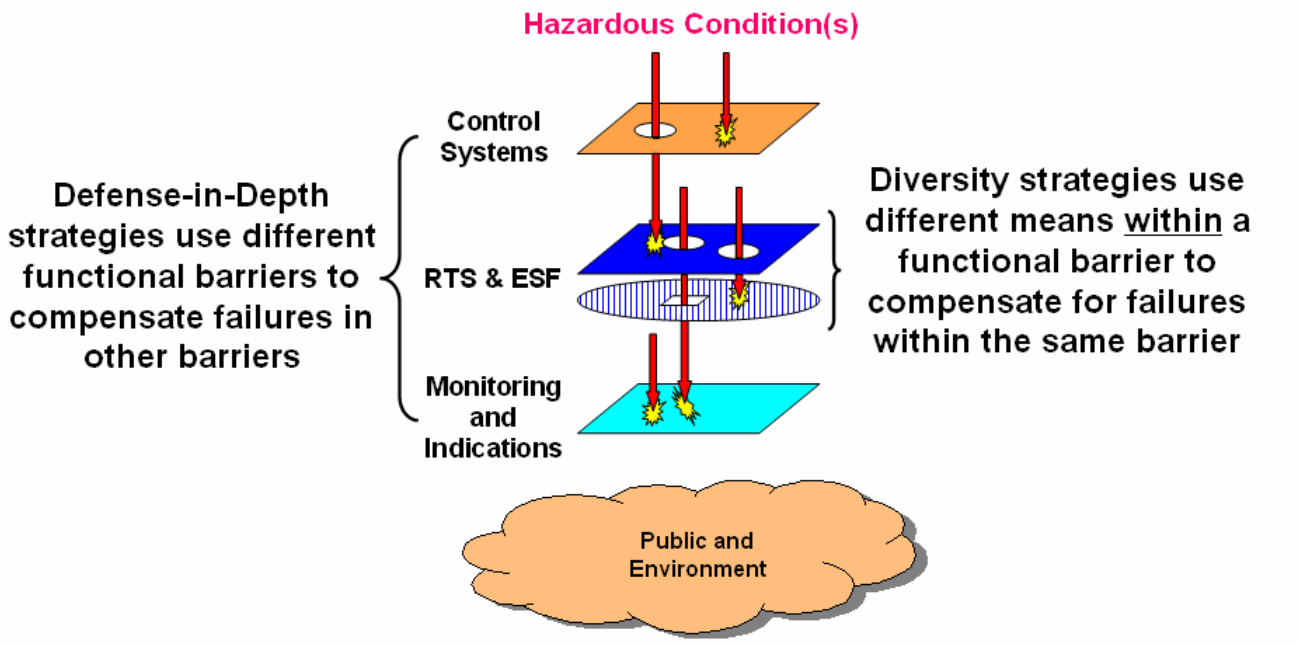


Figure 5-8 Defense-in-Depth and Diversity Strategies

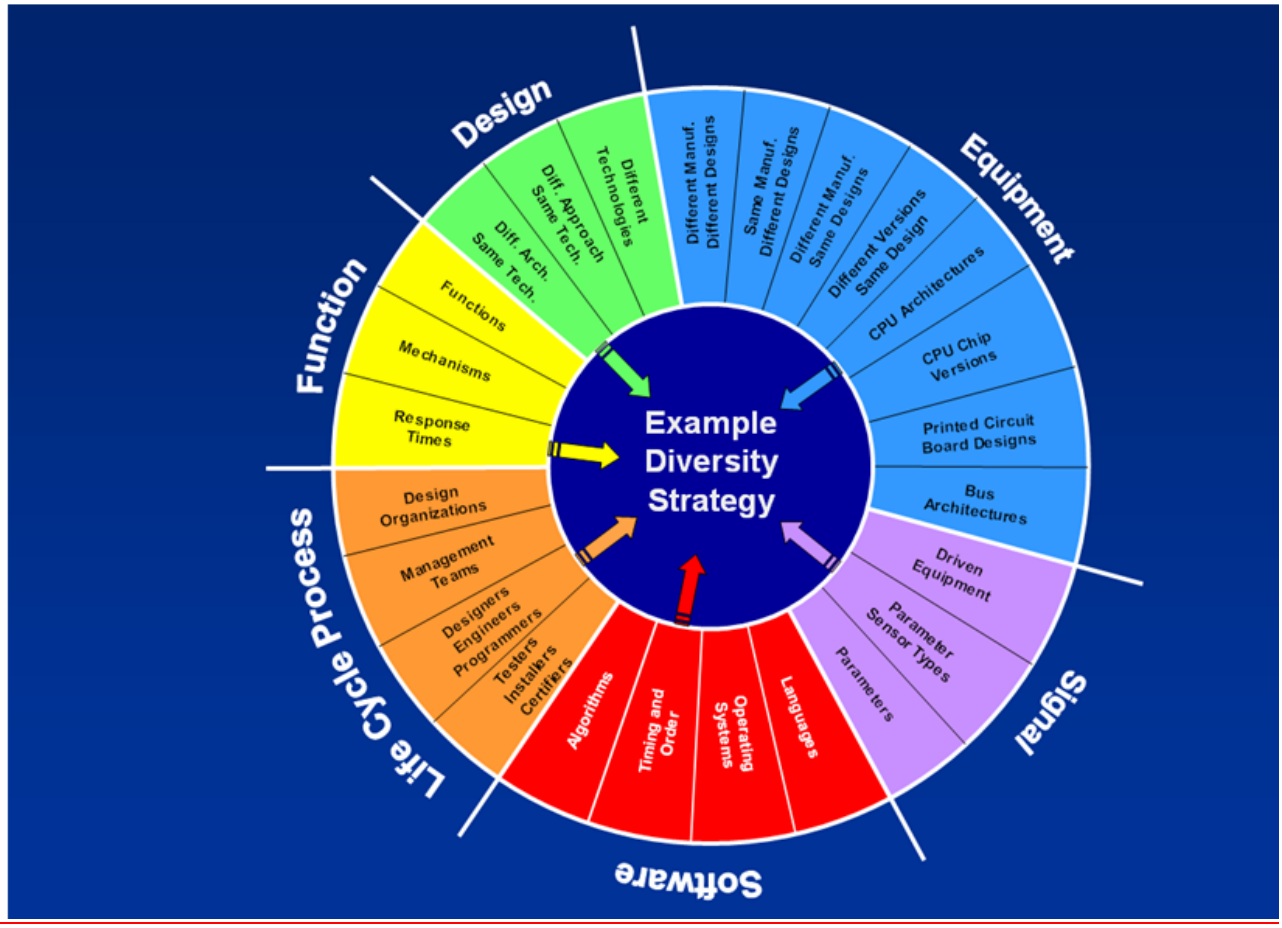


Figure 5-9 diversity Attributes and Criteria

Table 1: Where Faults Were Introduced

Phase	Injected Fault Density	Total (%)
Requirements	1.2	12.2
Data design	0.7	6.6
High level design	1.9	18.5
Low level design	1.5	15.3
Coding	4.7	47.4
Total	10.0	100.0

Figure 5-10 (Slide 5.2.6-6 (Table 1))

Table 2: Major Coding Faults

Type of Coding Defect	Total (%)
Logic	19.8
Standards	19.8
Maintainability	17.9
Interface	14.3
Performance	8.4
Functionality	4.0
Human Factors	3.5
Consistency	2.9
Data	2.4
Syntax	0.5
Other	6.6
Total	100

Figure 5-11 (Slide 5.2.6-8 (Table 2))

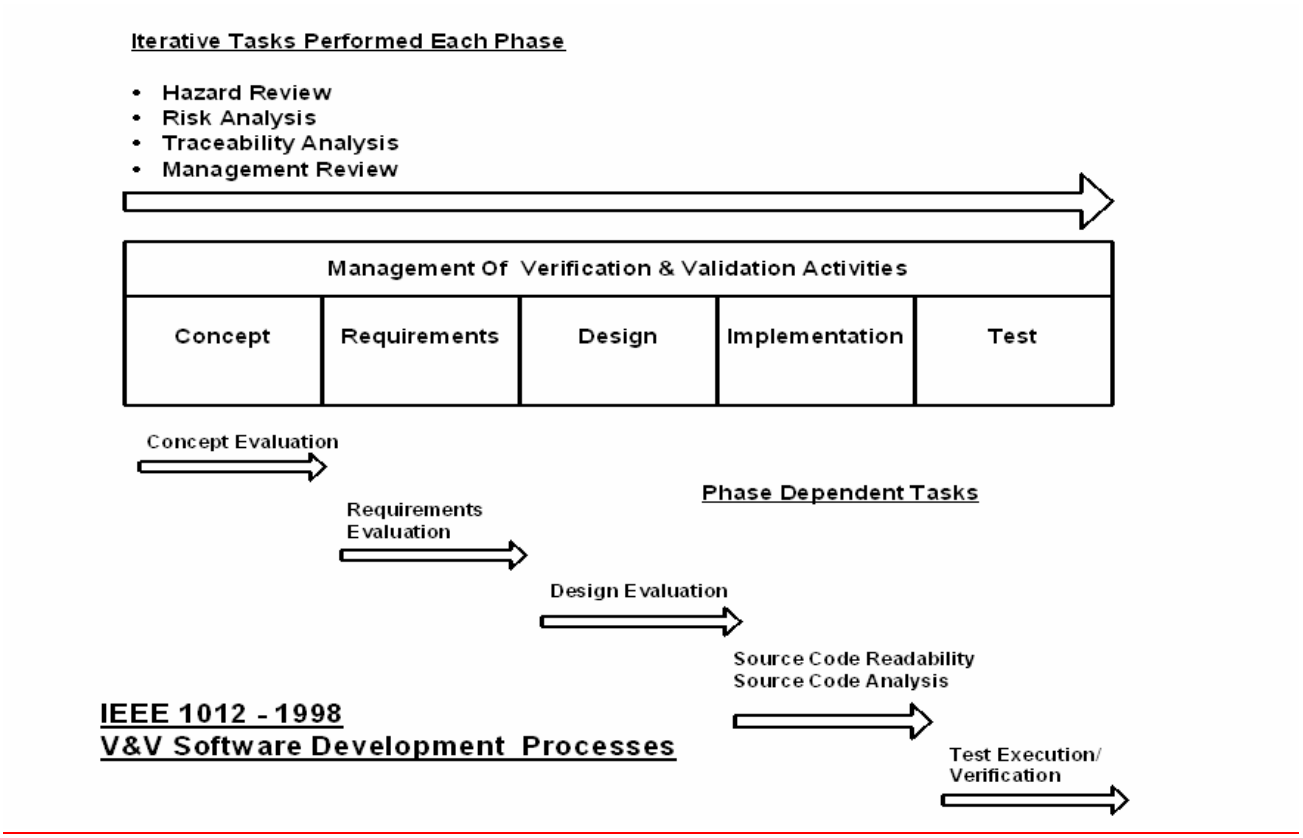


Figure 5-12 (Slide 5.2.8-3)

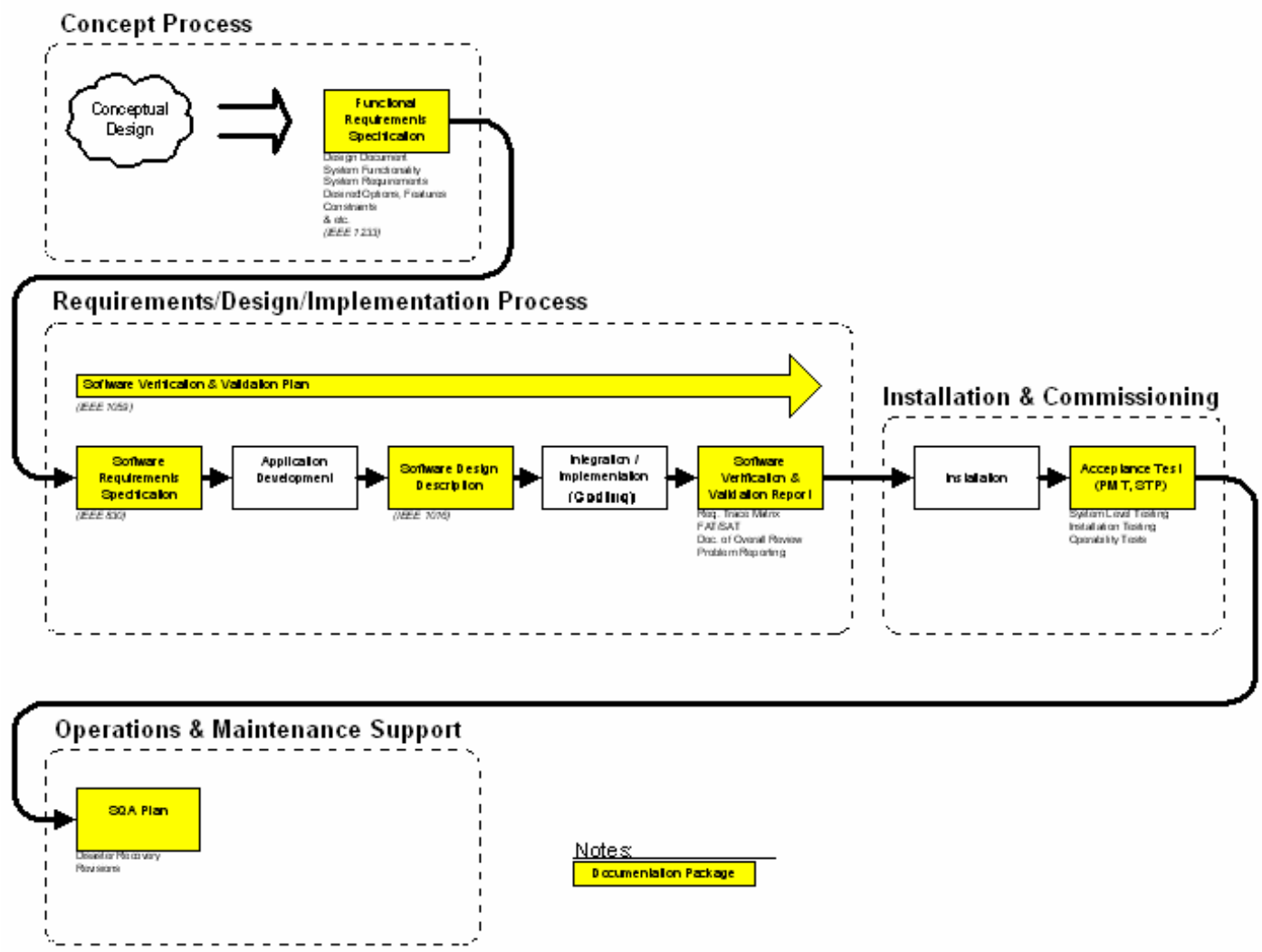


Figure 5-13 Typical Software Development Plan

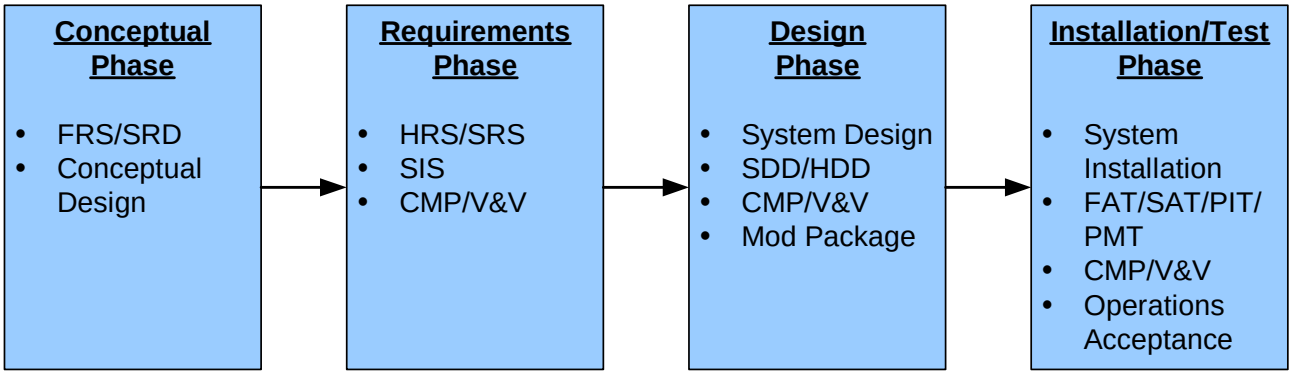


Figure 5-14: Life Cycle Overview of a Digital Upgrade Project

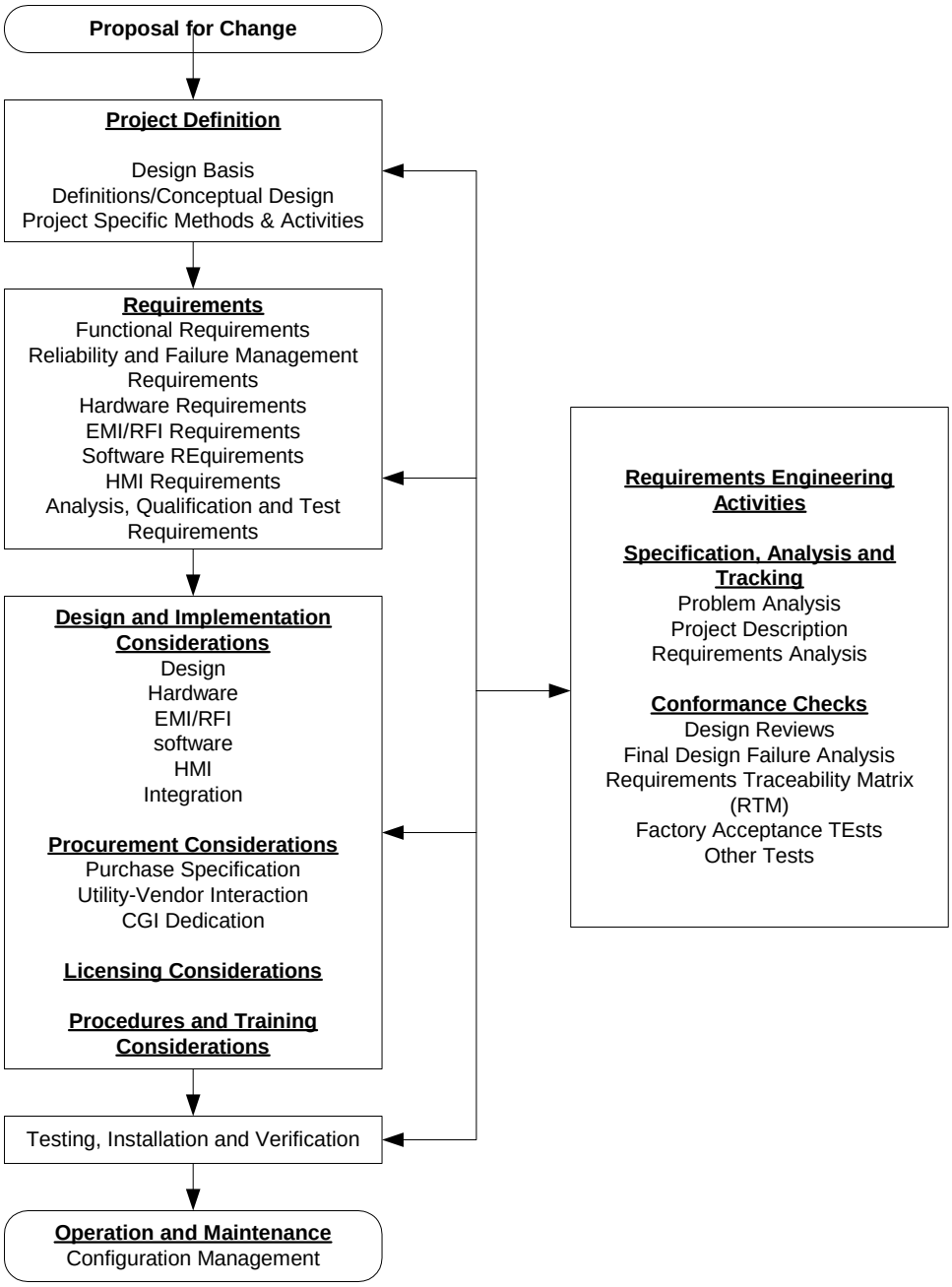


Figure 5-15 Digital Upgrade Life Cycle (Adapted from EPRI TR-102348)

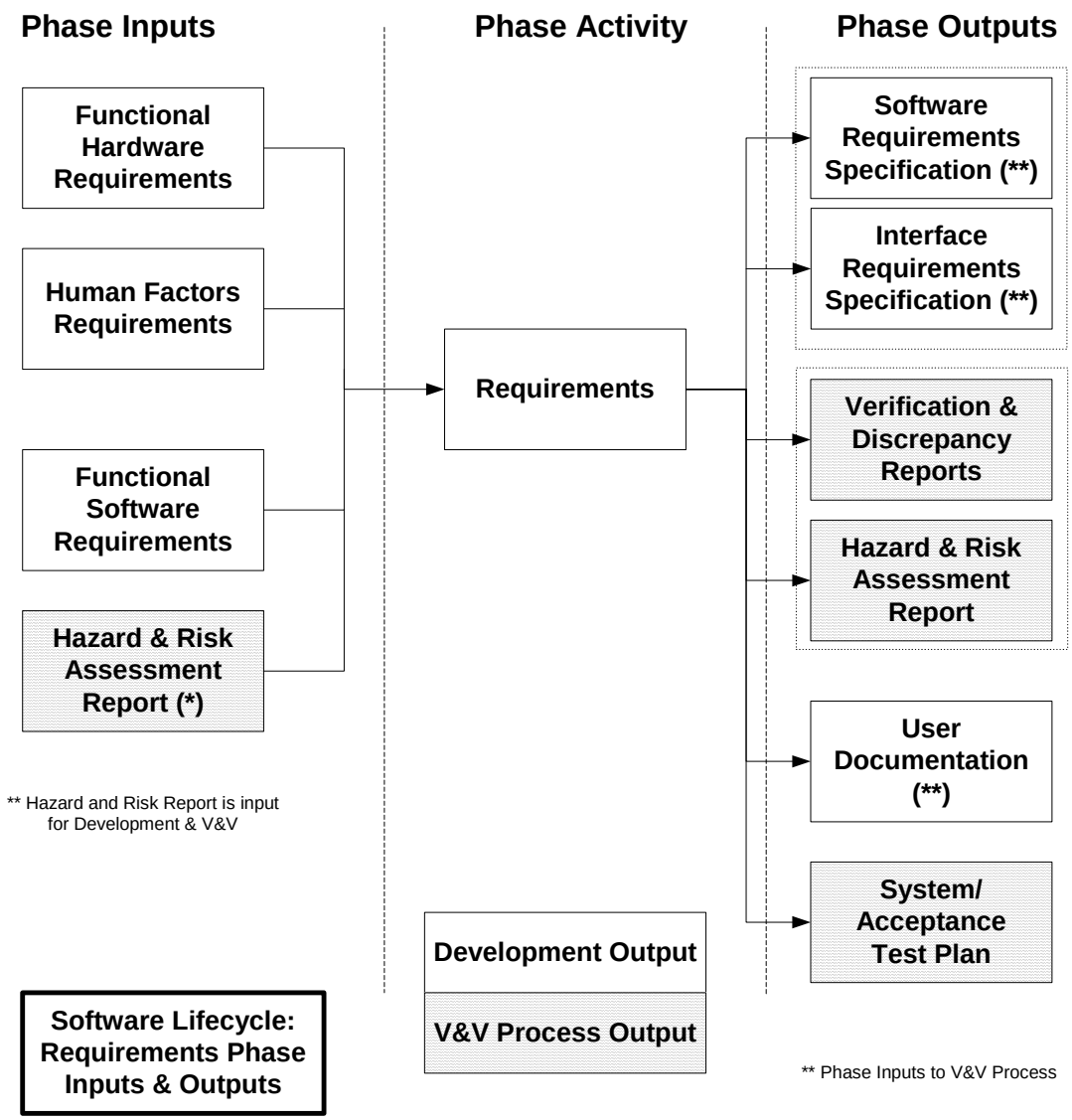


Figure 5-16 IEEE Std 1012 Requirements Phase Activity Inputs and Outputs

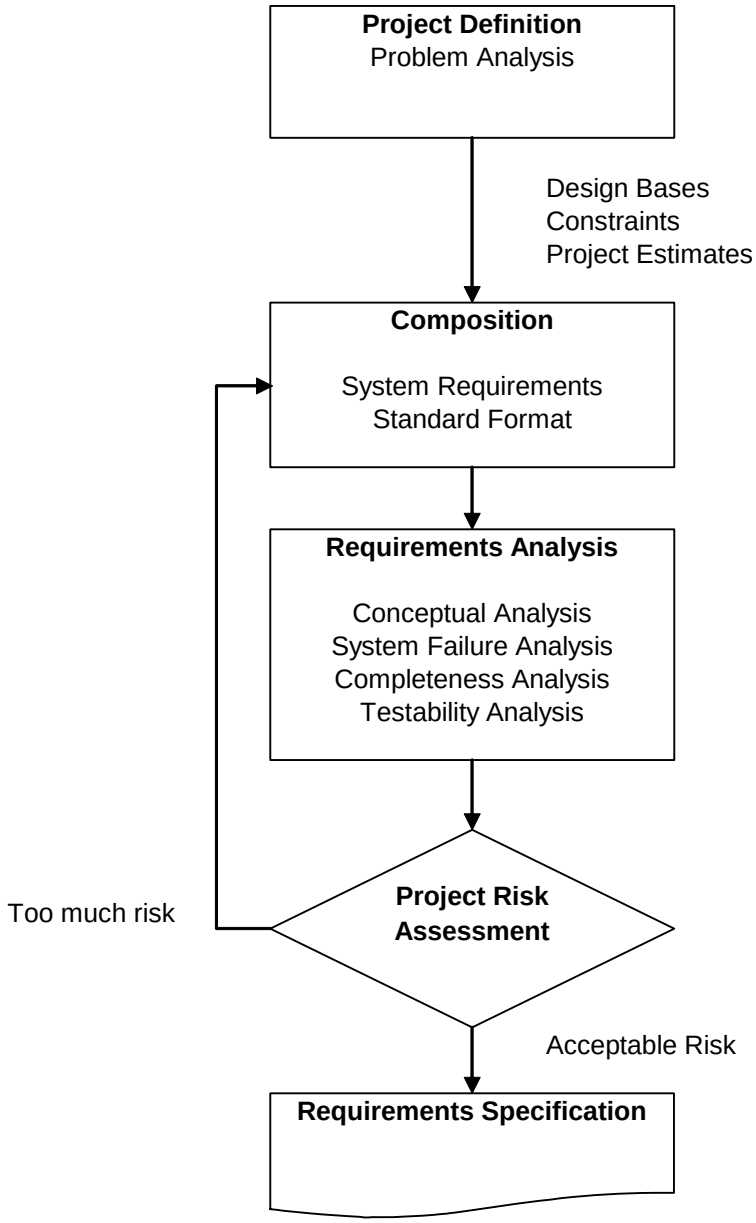


Figure 5-17 Requirements Specification Activities

(Adapted from EPRI TR-108831, Figure 3-1)

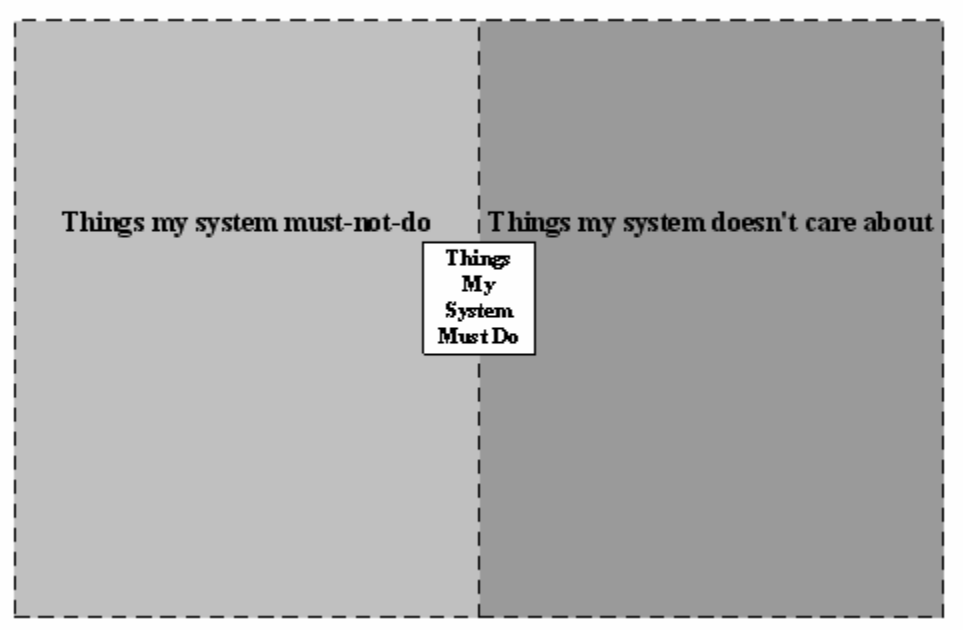


Figure 5-18 Defining Safety Requirements

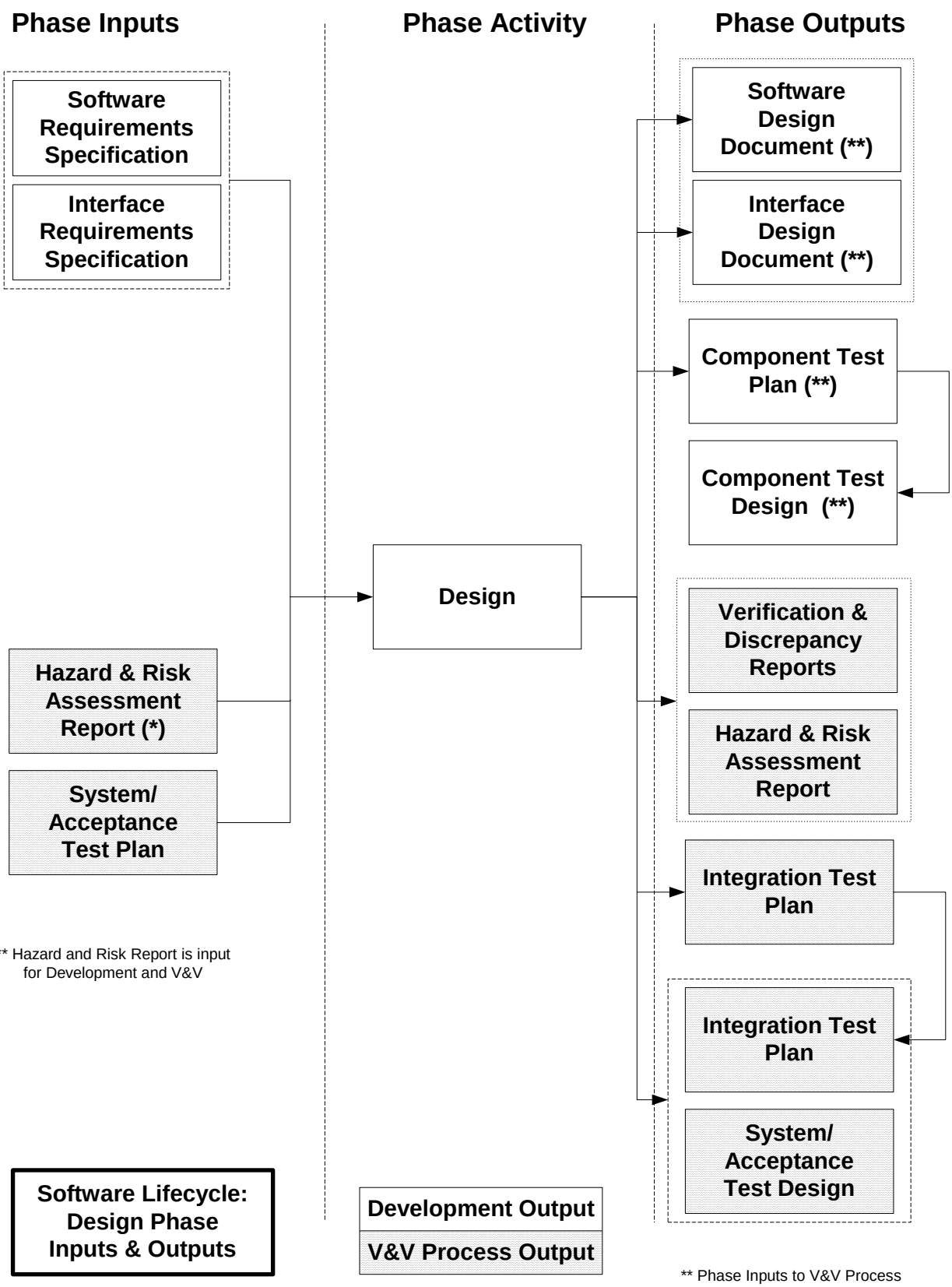


Figure 5-19 IEEE Std 1012 Design Phase Activity Inputs and Outputs

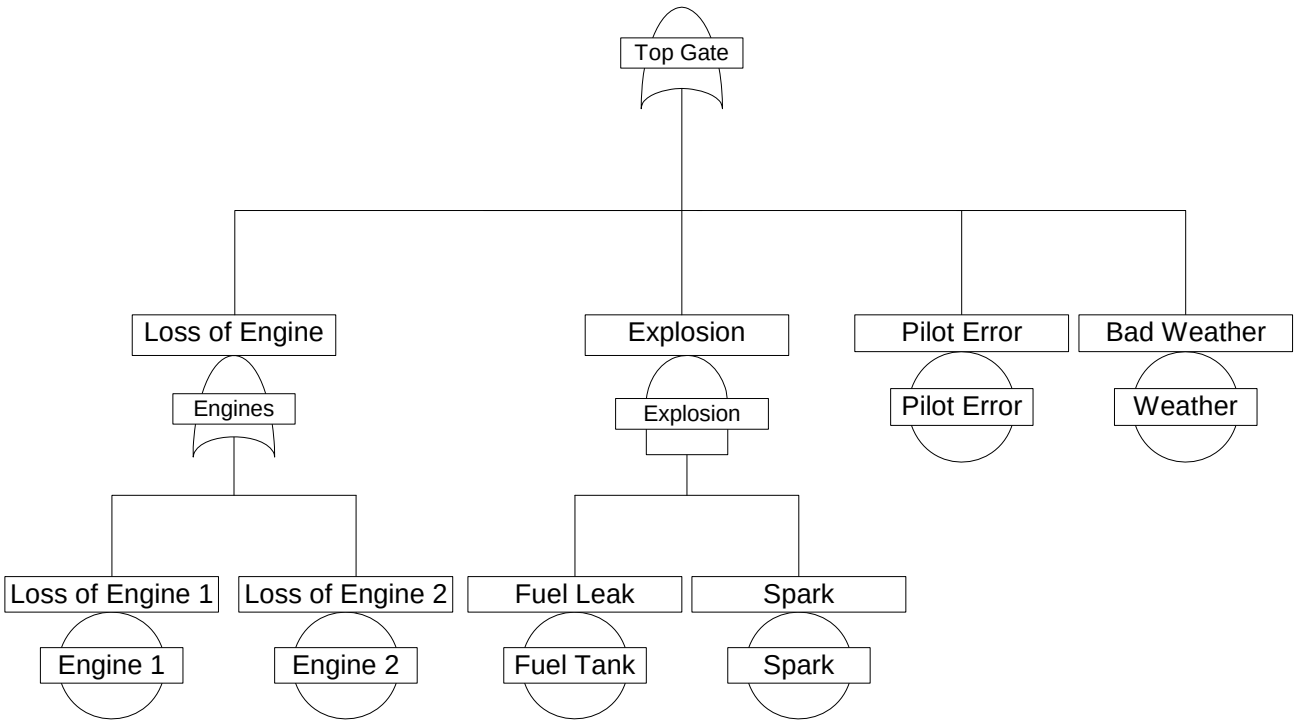


Figure 5-20 Fault Tree Example

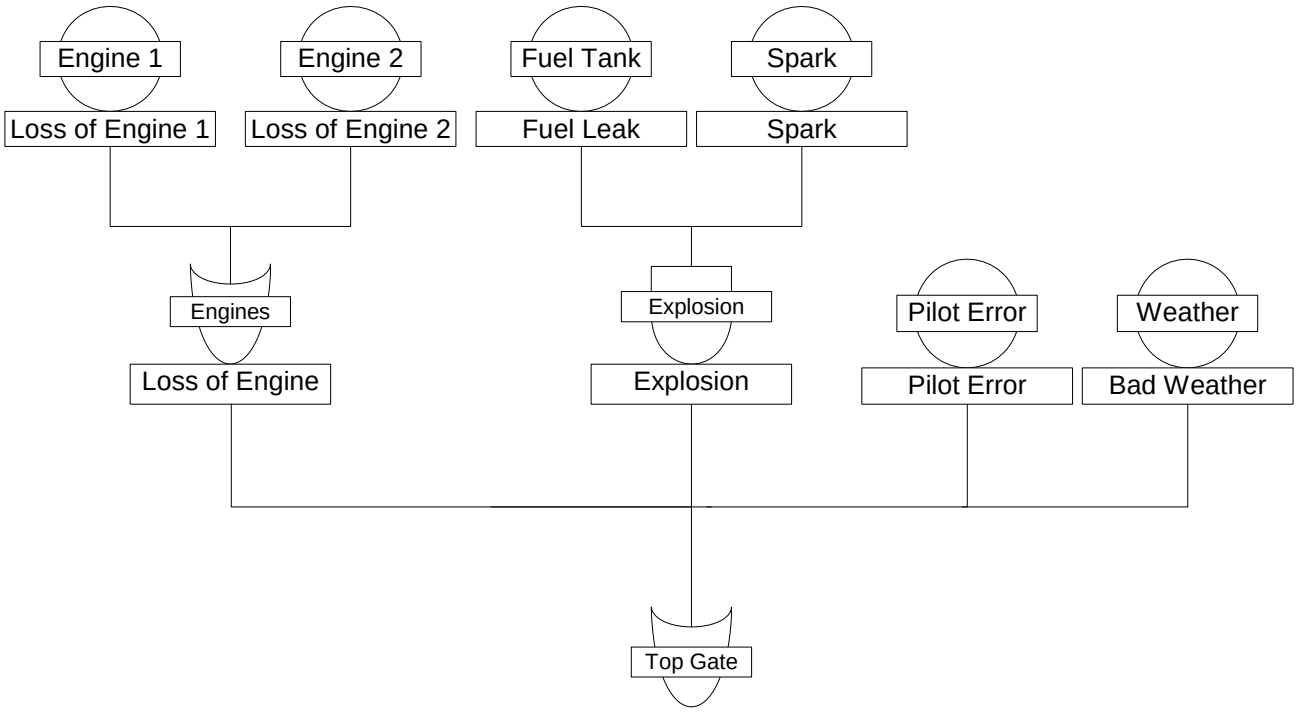


Figure 5-21 Event Tree/FMEA Example

Frequency of Occurrence	Severity			
	(1) Catastrophic	(2) Critical	(3) Marginal	(4) Negligible
(A) Frequent	1A	2A	3A	4A
(B) Probable	1B	2B	3B	4B
(C) Occasional	1C	2C	3C	4C
(D) Remote	1D	2D	3D	4D
(E) Improbable	1E	2E	3E	4E

Risk Categories:

High	Seri-ous	Medium	Low

Figure 5-22 MIL-STD 882B Hazard Matrix

	Catastrophic	Critical	Marginal	Negligible
Reasonable Likelihood	**4**	**4**	**3**	**2**
Probable Likelihood	**4**	**3 to 4**	**2 to 3**	**1 to 2**
Occasional Likelihood	**3 to 4**	**3**	**1 to 2**	**1**
Infrequent Likelihood	**3**	**1 to 2**	**1**	**1**

Figure 5-23 IEEE Std 1012 Safety Integrity Level

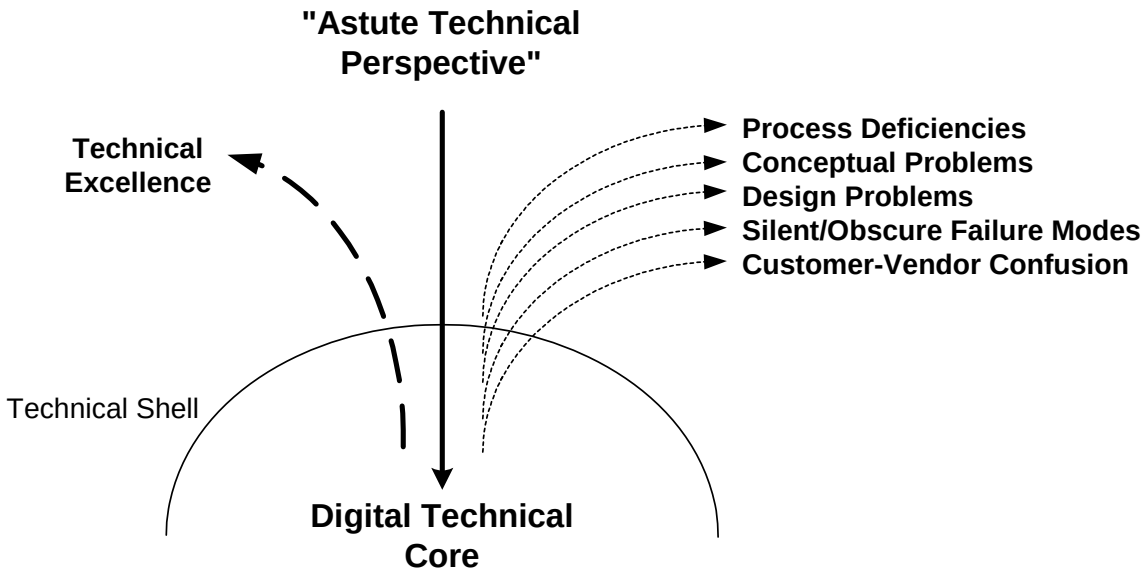


Figure 5-24 CDR Penetrates to Core Architecture

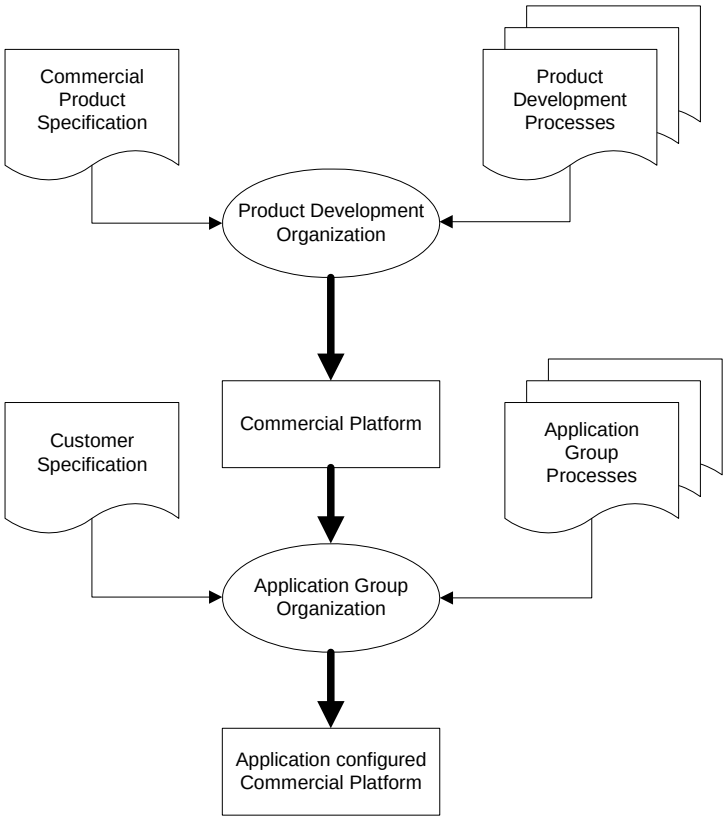


Figure 5-25 Platform vs. Application

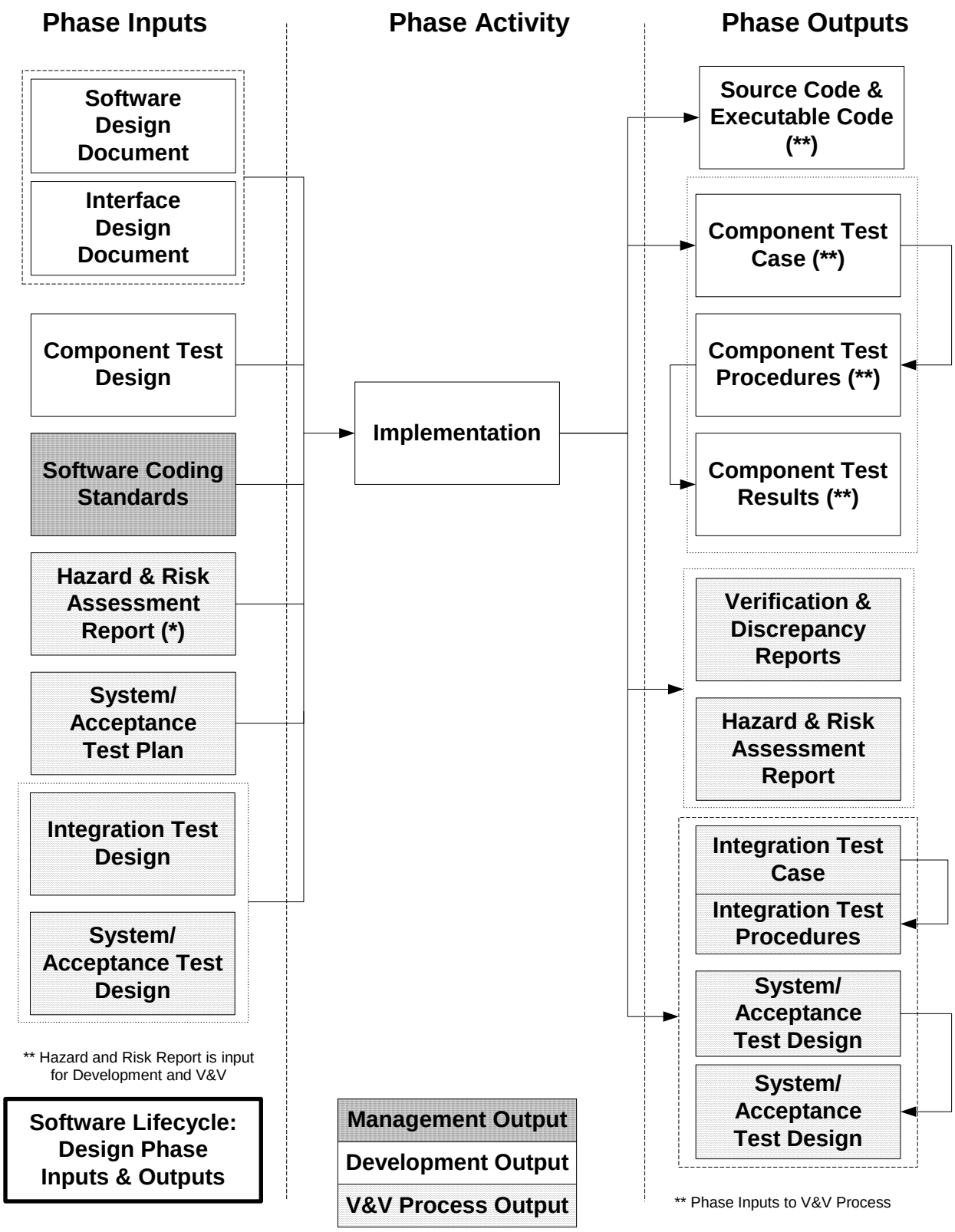


Figure 5-26 IEEE Std 1012 Implementation Phase Activity Inputs and Outputs

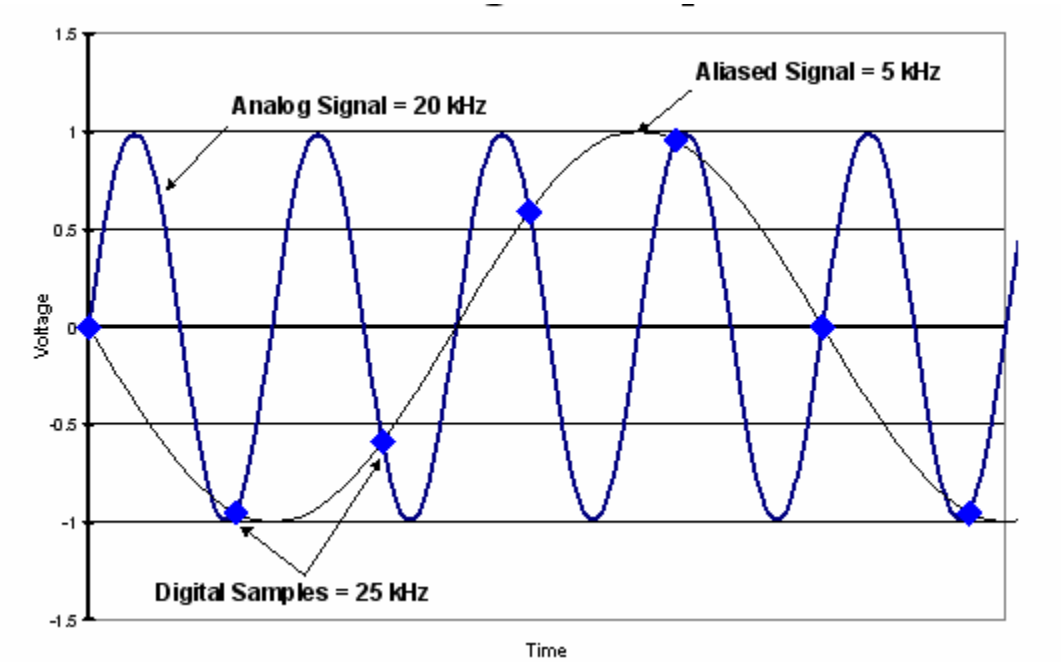


Figure 5-27 Aliasing Example

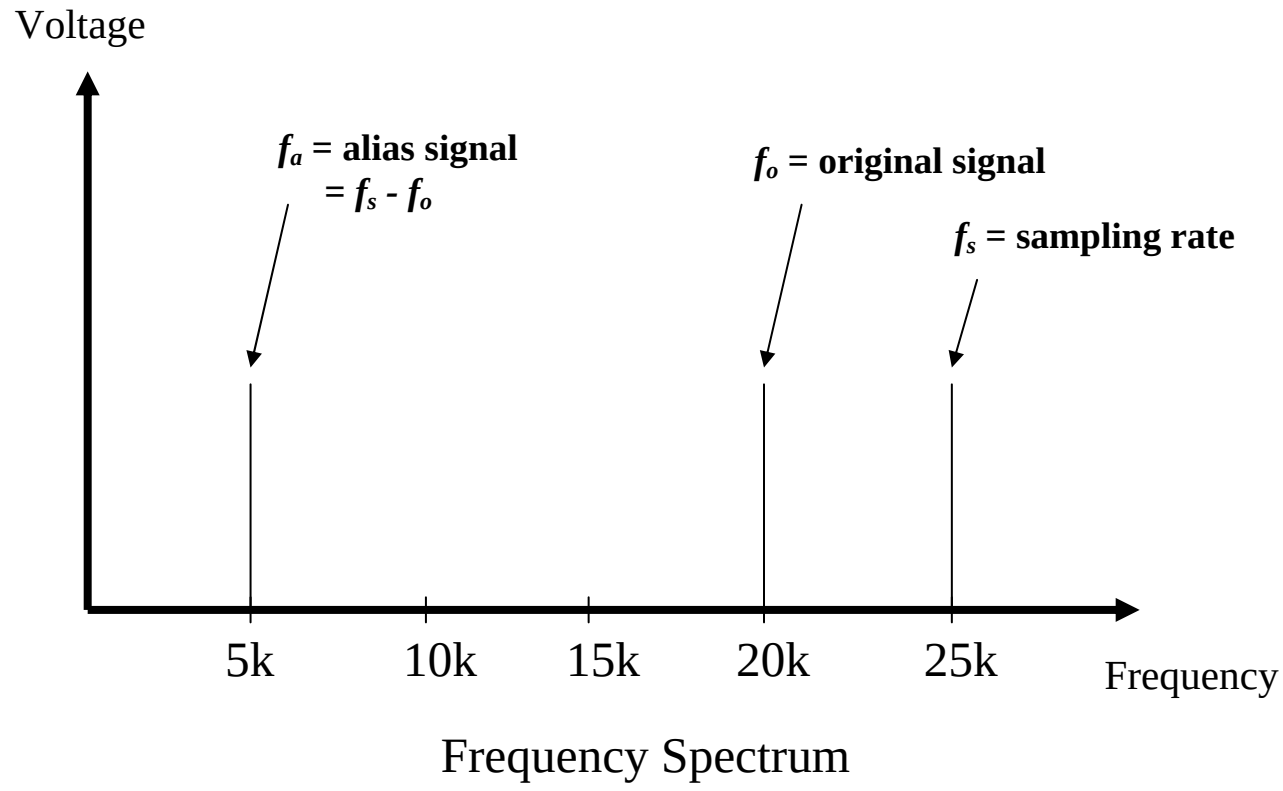


Figure 5-28 Aliasing Example, Continued

Wordlength	Steps	Resolution
4 bits	$2^4 = 16$	6.25% of range
8 bits	$2^8 = 16$	0.39%
12 bits	$2^{12} = 4,096$	0.024%
16 bits	$2^{16} = 65,536$	0.0015%

Figure 5-29 Wordlength Example

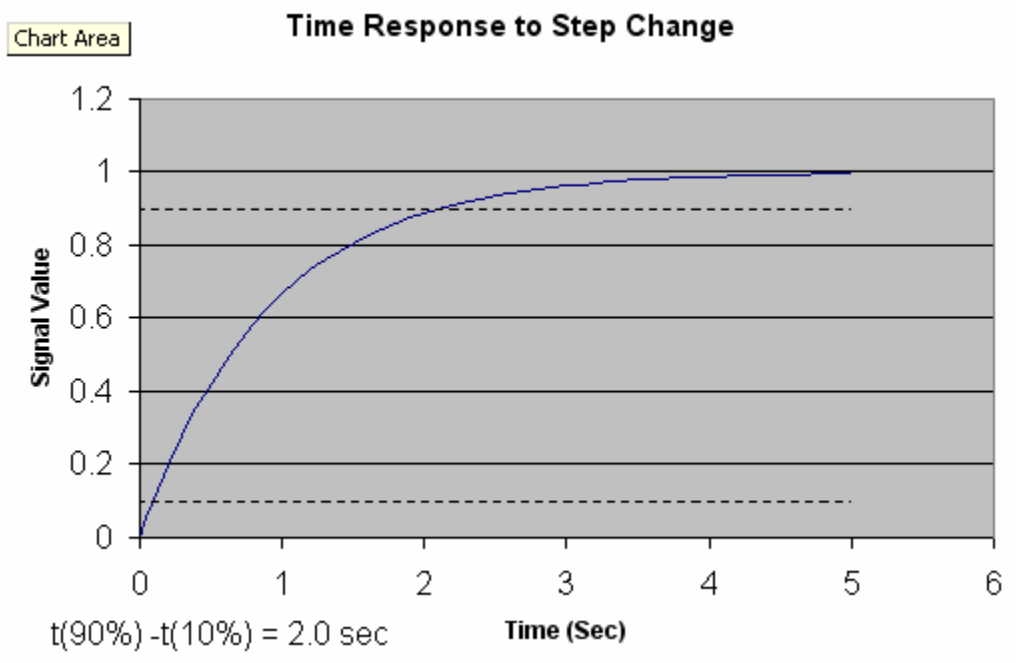


Figure 5-30 Rise Time Sampling Rate Selection Example

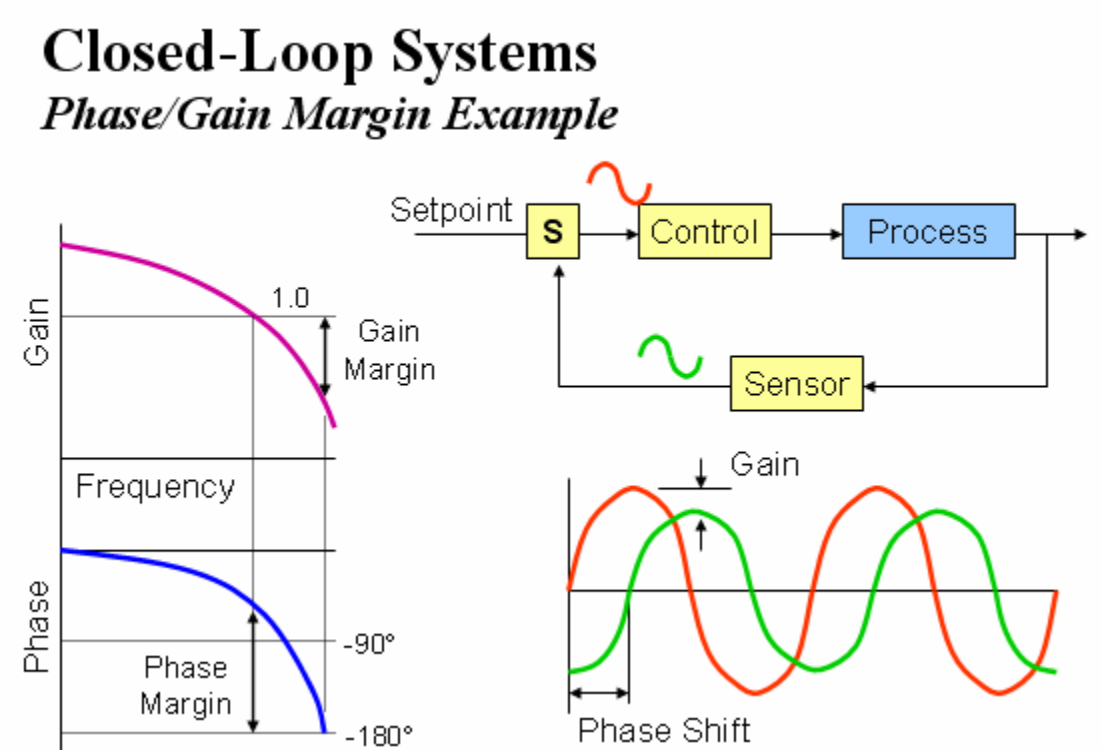


Figure 5-31 Phase/Gain Margin Example

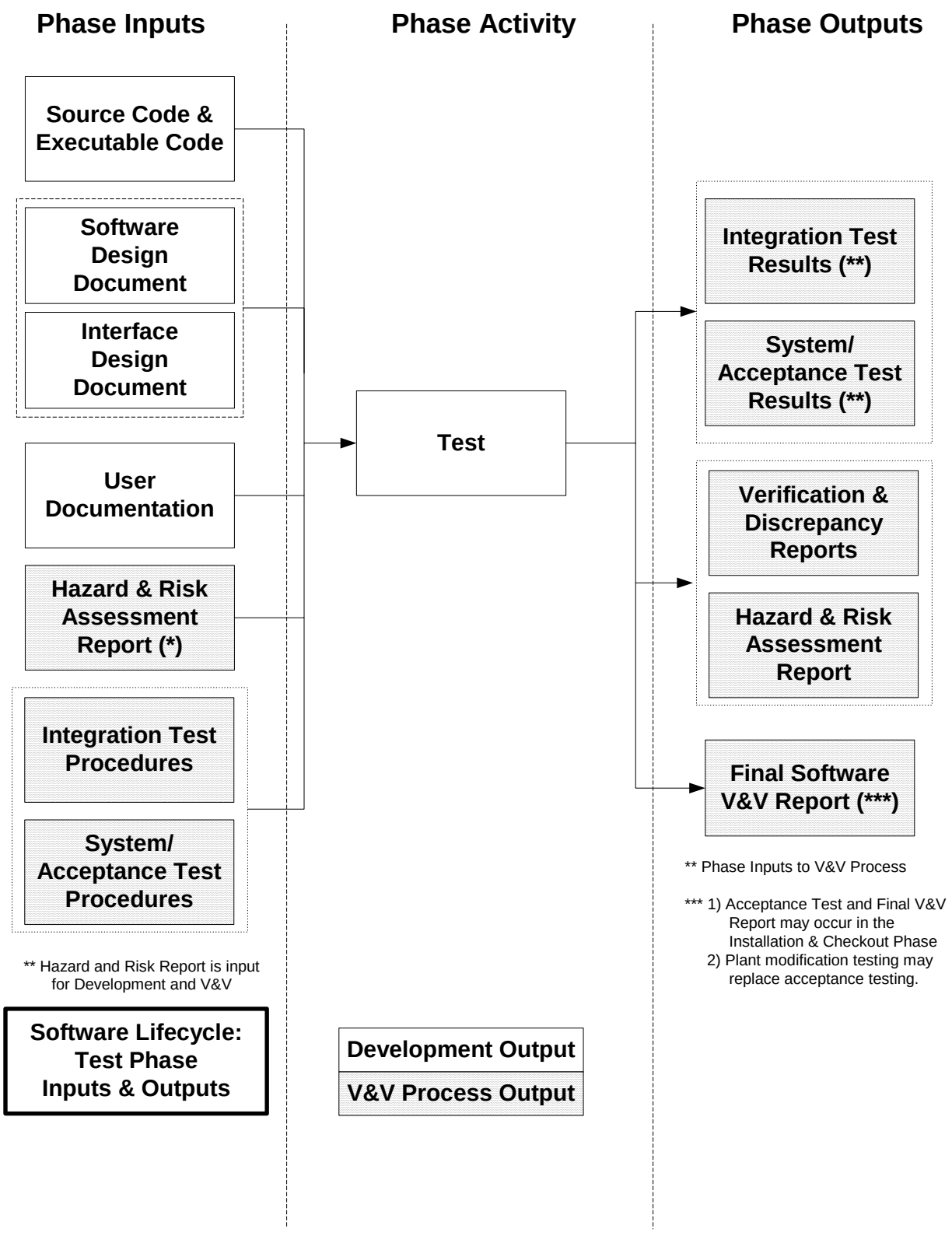


Figure 5-32 IEEE Std 1012 Test Phase Activity Inputs and Outputs