IEEE Guide for Software Verification and Validation Plans

Sponsor

Software Engineering Standards Committee
of the
IEEE Computer Society

Approved December 2, 1993

IEEE Standards Board

Abstract: Guidance in preparing Software Verification and Validation Plans (SVVPs) that comply with IEEE Std 1012-1986 are provided. IEEE Std 1012-1986 specifies the required content for an SVVP. This guide recommends approaches to Verification and Validation (V&V) planning. This guide does not present requirements beyond those stated in IEEE Std 1012-1986.
Keywords: baseline change assessment, life cycle phases, master schedule, V&V tasks

---

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Print: ISBN 1-55937-384-9, SH16923
PDF: ISBN 0-7381-0410-8, SS16923

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Introduction

(This introduction is not a part of IEEE Std 1059-1993, IEEE Guide for Software Verification and Validation Plans.)

The purpose of this guide is to provide guidance in preparing Software Verification and Validation Plans (SVVPs) that comply with IEEE Std 1012-1986* (referred to throughout as the Standard). The Standard specifies the required content for an SVVP. This guide recommends approaches to Verification and Validation (V&V) planning. This guide does not present requirements beyond those in the Standard.

When describing specific information to be placed in sections of an SVVP, this guide quotes extensively from the Standard. Not all of the Standard is reproduced. The Standard is a prerequisite for the use of this guide.

This guide does not provide a comprehensive view of the verification and validation process itself. It does provide an overview of the process of creating an SVVP. For each section in a plan, it presents a list of topics that could be addressed when preparing that section. The guide is not specific to any application area.

This guide includes the following:

a) Recommended planning procedures (e.g., criticality analysis) to aid in selecting specific V&V tasks for each phase
b) General guidance on the topics that shall be addressed in the life cycle phase sections of a plan (e.g., task descriptions, methods and criteria, and inputs and outputs)
c) Specific guidance in planning the V&V tasks required by the Standard
d) Samples of V&V planning based on the elevator system, which is described in annex A

As with the Standard, this guide is directed at the development and maintenance of critical software—software in which a failure could affect personal or equipment safety, or could result in a large financial or social loss. The guidance may also be applied to noncritical software. Determination of criticality should be carefully considered.

The Standard and this guide employ a specific life cycle model of the software development process, although plans written in the Standard may make use of a different model.

This guide may be used where software is the system or where software is part of a larger system. When software is part of a larger system, software V&V should be integrated into a system-level V&V effort covering critical hardware, software, and interfaces to operators.

This guide is primarily directed to those preparing SVVPs. This guide may be used by project management, system and software developers, quality assurance organizations, purchasers, end users, maintainers, and those performing V&V.

Neither the Standard nor this guide assumes that the V&V effort is necessarily independent of the development effort. V&V tasks may be performed by personnel doing development tasks, by a separate team within a single organization, by a completely independent organization, or by some combination as assigned within the SVVP.

*Information on references can be found in clause 2.

At the time this guide was completed, the P1059 working group had the following membership:

Jerome W. Mersky, *Chair* **Dolores Wallace,** *Co-chair*
Kirby Fortenberry, *Secretary*

François Coallier
Cora Carmody
Michael S. Deutsch
Michael P. Dewalt
David C. Doty
Caroline L. Evans
Robert V. Ebenau
Eva Freund
Jack Leavenworth
George Lee
Dennis E. Nickle
S. J. Pasquariello
Lee Perkins
Andrew Vaughan
Natalie C. Yopconka

The following persons were on the balloting committee:

H. R. Berlack
William J. Boll, Jr.
Fletcher Buckley
Geoff Cozens
Stewart Crawford
Bostjan K. Derganc
Einar Dragstedt
Robert G. Ebenau
Caroline L. Evans
John W. Fendrich
Kirby Fortenberry
Eva Freund
Roger Fujii
David Gelperin
Yair Gershkovitch
Julio Ganzalez-Sanz
David A. Gustafson
John Harauz
John W. Horch
Peter L. Hung
Myron S. Karasik
Judy S. Kerner
Robert Kosinski
Thomas Kurihara
Renee Lamb
Robert Lane
Ben Livson
Joseph Maayan
Jukka Marijarvi
Roger Martin
Scott D. Matthews
Ivano Mazza
Michael McAndrew
Sue McGrath
Jerome W. Mersky
Dennis E. Nickle
John D. Pope
Patricia Rodriguez
Hans Schaefer
Gregory D. Schumacher
Robert W. Shillato
D. M. Siefert
A. R. Sorkowitz
Vijaya Srivastava
David Terrell
Richard Thayer
George D. Tice
Leonard L. Tripp
Dolores Wallace
William M. Walsh
Natalie C. Yopconka
Janusz Zalewski

When the IEEE Standards Board approved this standard on December 2, 1993, it had the following membership:

Wallace S. Read, *Chair* **Donald C. Loughry,** *Vice Chair*
Andrew G. Salem, *Secretary*

Gilles A. Baril
José A. Berrios de la Paz
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman
Jim Isaak
Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight
Joseph L. Koepfinger*
D. N. "Jim" Logothetis
Don T. Michael*
Marco W. Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley I. Warshaw

Rachel A. Meisel
IEEE Standards Project Editor

Contents

IEEE Guide for Software Verification and Validation Plans

1. Overview

This guide is the explication of IEEE Std 1012-1986[1] (referred to throughout as the Standard). This Standard specified the required content of a Software Verification and Validation Plan (SVVP). This guide provides the guidance and background discussion that a user of the Standard may need in order to write a plan.

This guide is divided into four clauses. Clause 1 provides the overview and scope of this guide. Clause 2 lists references to other standards. Clause 3 defines the terms, conventions, and acronyms used in this guide. Clause 4 provides an overview of software V&V, general planning guidance, and a description of the major V&V activities that continue across more than one life cycle phase (e.g., traceability analysis, evaluation, interface analysis, and testing). Clause 5 provides a sequential review of each of the sections of the SVVP, providing specific guidance for each section. In many cases, the guidance for a section or task is in the form of a list of questions or activities to consider. Note that these lists are not complete because for any given situation there will be other questions or activities that can be addressed. Also, all items in a given list may not be applicable to any given situation.

This guide also contains three annexes. Annex A describes the elevator example that was used to develop sample plan extracts. These sample extracts are also presented in this annex. Annex B describes and overviews software criticality planning analysis. Annex C provides guidance for the seven planning topics that are aimed at V&V tasks in an SVVP.

1.1 Scope

This document provides specific guidance about planning and documenting the tasks required by the Standard so that the user may write an effective plan. Some guidance is in the form of checklists, questions, or activities to consider.

This guide does not present requirements beyond those found in the Standard. Mention of certain methods or activities in this guide does not mean that the user of the Standard must include these in a plan in order for that plan to comply with the Standard. There may be other methods or activities that are relevant to a specific software project or product. The user of this guide is encouraged to survey the standards referenced in clause 2 for additional guidance, particularly guidance for specific tools or techniques.

[1]Information on references can be found in clause 2.

2. References

This guide shall be used in conjunction with the following publications:

IEEE Std 610.12-1990, IEEE Glossary of Software Engineering Terminology (ANSI).[2]

IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).

IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans (ANSI).

IEEE Std 829-1983 (Reaff 1991), IEEE Standard for Software Test Documentation (ANSI).

IEEE Std 830-1993, IEEE Recommended Practice for Software Requirements Specifications.

IEEE Std 1008-1987, IEEE Standard for Software Unit Testing (ANSI).

IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans (ANSI).

IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions (ANSI).

IEEE Std 1042-1987, IEEE Guide to Software Configuration Management (ANSI).

[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

3. Conventions, definitions, and acronyms and abbreviations

3.1 Conventions

Below are a number of conventions adopted within the body of this guide.

— *Standard* refers to IEEE Std 1012-1986.
— *Plan* refers to a Software Verification and Validation Plan (SVVP).
— *Shall* or the imperative form indicates an item or activity required by the Standard.
— *Should* indicates a recommended item or activity.
— *May* indicates an item or activity appropriate under some, but not all, conditions; for which there are a number of acceptable alternatives; or for which there is no professional consensus.
— *Life cycle model* is taken from the Standard. As stated in the Standard, a different model of the life cycle may be used, provided the SVVP written for that model includes cross-references to the original model.
— *System* is used to denote the item that will contain the developed software.
— Material quoted from the Standard is indented and italicized and is followed by a reference to the section (or, in some cases, part of a table) of the Standard.

3.2 Definitions

This guide is consistent with other IEEE standards in its use of terms. For a complete list of definitions, refer to IEEE Std 1012-1986.

Note that definitions for the terms *verification* and *validation* may be used differently by other standards organization.

3.3 Acronyms and abbreviations

The following acronyms and abbreviations are used in this guide:

COTS Commercial-off-the-shelf

CPM Critical path method

IV&V Independent verification and validation

PERT Program evaluation review technique

QA Quality assurance

SDD Software design description

SQA Software quality assurance

SRS Software requirements specification

SV&V Software verification and validation

SVVP Software verification and validation plan

SVVR Software verification and validation report

TBD To be determined

V&V Verification and validation

4. Software verification and validation

Software Verification and Validation (V&V) is a disciplined approach to assessing software products throughout the product life cycle. A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements. V&V provides software management with insights into the state of the software project and products, allowing for timely change in the products or in the development and support processes.

Software verification and validation employs review, analysis, and testing techniques to determine whether a software system and its intermediate products comply with requirements. These requirements include both functional capabilities and quality attributes.

These quality attributes will differ from project to project. Quality attributes that are identified serve the user's need for a software product that is capable of meeting its objectives with adequate performance with no unexpected side effects. The following list enumerates quality attributes that may be specified.

— accuracy
— completeness
— consistency
— correctness
— efficiency
— expandability
— flexibility
— interoperability
— maintainability
— manageability
— portability
— readability
— reusability
— reliability
— safety
— security
— survivability
— testability
— usability

The objectives of the V&V effort are to find defects and to determine if required functions and attributes are built into the software system. The activities of V&V operate on the products of software development and support the following:

a) Verification that the products of each software life cycle phase
    1) Comply with previous life cycle phase requirements and products (e.g., for correctness, completeness, consistency, and accuracy)
    2) Satisfy the standards, practices, and conventions of the phase
    3) Establish the proper basis for initiating the next life cycle phase activities
b) Validation that the completed end product complies with established software and system requirements

A V&V effort is typically applied in parallel with software development and support activities. Some V&V tasks may be interleaved with the development and support process. A V&V effort consists of management tasks (e.g., planning, organizing, and monitoring the V&V effort) and technical tasks (e.g., analyzing, evaluating, reviewing, and testing the software development processes and products) to provide information about the engineering, quality, and status of software products throughout the life cycle.

Planning for a V&V effort begins early in the project so as to aid in assessing the scope of the V&V effort as part of the total effort. This is necessary in order to ensure that V&V resource needs are included in the total project planning. The initial SVVP may provide insights to the financial sponsor of the project concerning development or support plans. It also provides enough information for the sponsor to approve the plan and to monitor its implementation.

It is important on each project to clarify how V&V fits into the overall project life cycle and relates to all project entities (e.g., user, developer, buyer, software configuration management, software quality assurance, etc.). Sometimes the V&V effort will be performed by an entirely different organization, often directly for the buyer of the software. This situation is referred to as Independent Verification and Validation (IV&V). V&V tasks may be performed internally by a completely separate organization within the company or perhaps by the systems engineering group or product assurance group. Selection of some tasks may depend on how a project is organized (e.g., consider not only how the V&V tasks serve immediate V&V objectives, but also how they serve the total project).

The first V&V tasks begin early, with concept documentation evaluation, requirements analysis, and acceptance test planning. Planning the V&V should be closely coupled to the planning of the rest of the project. The planner should anticipate and schedule regular updating of the SVVP to reflect changes in the overall development effort.

Software verification and validation tasks support one another and combine to become a powerful tool that does the following:

— Discovers errors as early as feasible in the software life cycle
— Ensures that required software qualities are planned and built into the system
— Predicts how well the interim and final products will result in final products satisfying user requirements
— Assures conformance to standards
— Confirms safety and security functions
— Helps prevent last-minute problems at delivery
— Provides a higher confidence level in the reliability of the software
— Provides management with better decision criteria
— Reduces frequency of operational changes
— Assesses impact of proposed changes on V&V activities

4.1 Software V&V planning guidance

This subclause describes activities required for effective V&V planning. The purpose of planning and plan documentation is to employ the V&V resources efficiently, to monitor and control the V&V process, and to allow the identification of each participant's role and responsibility. Planning results in a well-structured, thorough, and realistic SVVP that provides the foundation for a successful V&V effort. Documentation of the scope and explicit statement of the objectives provides all parties with an understanding of what is going to be achieved and what is not.

Each project needs its own SVVP, based on the generic plan identified in the Standard. The specific plan will be tailored to the project. The plan may be modified at times in response to major changes. While the basic objectives of the V&V effort are not likely to change over time, there may be major changes that require response. These changes may be associated with development activities (e.g., significant delays in schedules, unanticipated technological changes) or V&V tasks (e.g., loss of resources, inability to develop tools in time). Therefore, procedures for modifying the plan should be included in the plan.

V&V planning, which should be thought of as an integrated part of overall project planning, may be broken down into the following steps:

a) *Identify the V&V scope*. Define a set of software tasks as early as possible. The tasks should be tailored and take into consideration several software factors (e.g., criticality, complexity, available resources including development environment and tools). Determining which V&V tasks to perform requires a decision-making methodology. One such methodology, criticality analysis, is presented in annex B.

Before beginning the next step, the recommended V&V tasks should be reviewed by all participants, especially the user and customer. The next step should begin only after agreement is achieved.

b) *Establish specific objectives from the general project scope*. Detailed, measurable, and achievable objectives establish the conditions of satisfaction with the V&V effort. These conditions of satisfaction provide the basis for assessing and enforcing V&V performance. Measurable criteria, such as defined load limits and response time, are desirable because of the objectivity they afford. Also, the discipline and effort required to formulate measurable objectives reveal ambiguities and differences of expectations early in the V&V effort.

It is important to identify only achievable objectives. The presence of unachievable objectives reduces the credibility of a plan, and of the V&V effort itself, before the execution of the plan begins. A realistic plan enhances the motivation of all parties in a project and increases the value of the V&V effort.

c) *Analyze the project input prior to selecting the V&V tools and techniques and preparing the plan*. Input analysis identifies all the specific products, information, and other documentation that will be available during the project phases. Examples of such products are given in figure 1 of the Standard and are reproduced here in figure 1. Input can be divided into two types—the products and information available at project start and the products and information that will become available in the later phases of the development cycle. The items that are available during the planning stage are either project- or product-related. Project-related information includes the budget, manpower and other resources, development schedules and milestones, and any other project-specific considerations, such as contractual requirements. Product-related information includes concept documentation, requirements specifications, or other formal specifications for the product being developed. Note that terminology for these products varies according to the user or developer environment.

Historical defect data from previous versions of the same or similar software, if available, can be a great aid. Such historical data will be available only if planned for (e.g., by archiving and analyzing anomalies). When planning the V&V effort, prediction and analysis of the nature and number of expected defects aid in the selection of V&V tasks and help to determine the resources required to perform the tasks.

d) *Select techniques and tools*. Identify those tools and techniques (such as those that support traceability analysis) that are available and applicable to the project, and gather sufficient information in order to choose among alternatives. For techniques, this information includes input needed, range of applicability, and resources and skills needed. For tools, this information includes availability of tool and associated documentation, applicability, and resource requirements including cost of operation and training.

Specific information about the software development environment, methodology, and tools should be considered when selecting the tools and techniques to be used in V&V. Techniques and tools need to be selected with the format, structure, and schedule of the development products in mind. Also, while in some cases it is appropriate to have separately developed and qualified tools, it may be more cost effective to use a common set of tools for similar functions.

Obtain a subset of tools and techniques that are tailored to the particular constraints and needs of the project. When doing so consider the skills and training of the personnel and the availability of the other necessary resources.

e) *Develop the plan*. Review the results of the previous steps and prepare a detailed set of tasks (addressing the seven topics discussed in 3.5 of the Standard and annex C of this guide) to meet the V&V goals, objectives, and constraints.

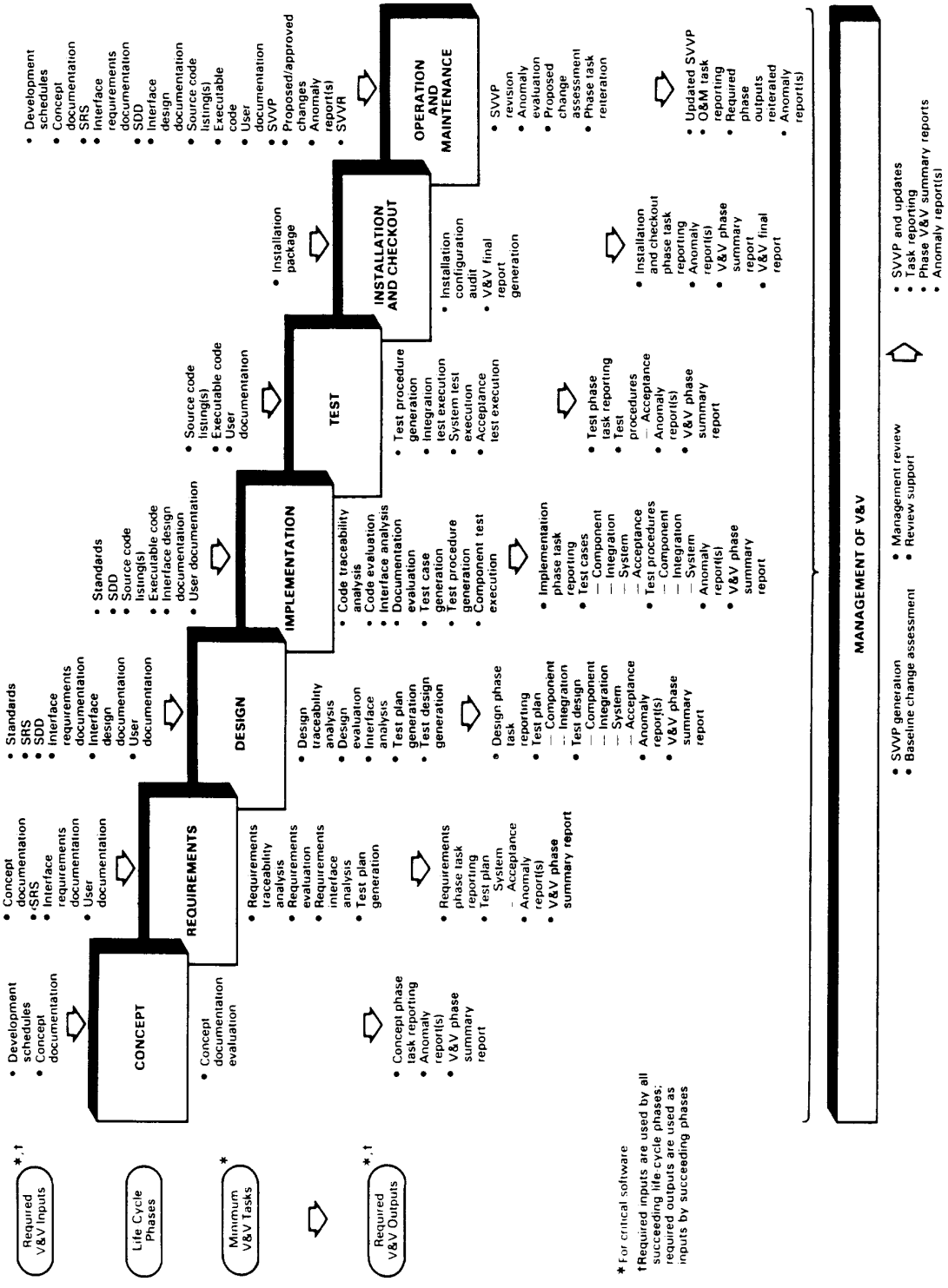

Figure 1—Software verification and validation plan overview

4.2 Integrating and continuing V&V tasks

Several tasks, as described by the Standard, collectively make up continuing activities that go across the different life cycle phases. These general activities are traceability analysis, evaluation, interface analysis, and testing. Two other activities also discussed in the Standard are management and reporting. Management is described in 5.5.1 of this guide. Reporting is described in 5.6.

These activities are horizontal threads that tie together the subsequent phase activities and allow verification to be more effectively conducted. The V&V planner should plan to integrate and share resources throughout the life cycle so that planning and implementation of all tasks of a continuing activity are more effectively performed. See table 1.

In order to avoid redundancy, only the general characteristics of these activities are described in this subclause. Phase-specific applications are described in 5.5. Each activity is described in terms of the functions performed, the kinds of problems it will uncover, the significance of the problems, and the relationship of this activity to other activities.

Table 1—Relationships between tasks

Contribution of Task	**Task**			
	Traceability Analysis	*Evaluation*	*Interface Analysis*	*Testing*
Criticality Analysis	Identifies critical features for tracing.	Identifies critical areas for evaluation.	Identifies critical interfaces.	Establishes priorities for testing.
Traceability Analysis		Maps the location of each feature to be evaluated.	Identifies where interfaces are required between dependent components, and the information to be passed.	Traces test documents. Assists in developing a test strategy by determining how functions are distributed.
Evaluation	Assures the structural validity of the distribution of function.		Establishes that interfaces are appropriate and are correctly implemented.	Identifies attributes that require dynamic validation, and the extent of testing that will be needed.
Interface Analysis	Establishes the need for traceability, and assures that dependent elements can communicate.	Identifies explicit relationships between elements, and their input and output requirements.		Establishes the framework for integration test planning.
Testing	Confirms the traceability and the distribution of functions.	Confirms the evaluation.	Confirms that interfaces are operational.	

4.2.1 Traceability analysis

The Standard requires traceability analysis in the requirements, design, and implementation phases. Traceability is the ability to identify the relationships between originating requirements and their resulting system features. It permits tracking forward or backward through the network of interrelationships that are created as requirements are decomposed and refined throughout a system's life cycle. Traceability provides the

thread that links one element to another. When an element is traced to another, and that element is traced on to another, a chain of cause and effect is formed.

Traceability allows verification of the properties set forth in the concept and that requirement specifications have been

a) Carried forward to the design specifications
b) Implemented in the code
c) Included in the test plan and cases
d) Provided to the customer and user in the resulting system

Traceability, with the aid of forward and backward tracing, facilitates the construction of efficient test plans and permits verification that the resulting test cases have covered the permutations of functional and design requirements/features.

When the trace from all software requirements back to the concept documentation and system requirements is verified, then successive traceability analysis occurs, beginning from the software requirements through all the development representations, user documentation, and test documentation. Each trace is analyzed for consistency, completeness, and correctness to verify that all software requirements are implemented in the software and are associated with the correct design, code, and test information.

In the course of performing traceability analysis, some errors are readily apparent, such as a requirement with no design element, or a design element with no source code, or the converse of these. The analyst examines each trace path to ensure that the connected pieces are the proper ones. To be able to do this the analyst has to understand the intent of both requirements and design.

Traceability analysis can be used to support configuration management, test coverage analysis, analysis of V&V results, regression testing, criticality assessment, and V&V management decisions. Also, traceability analysis is useful in evaluating the software development effort for good software engineering practices.

Traceability analysis is performed to assure that

— Every requirement in the specification is identified correctly
— All traces are continuous from the prior phase through the current phase
— Forward and backward traces between adjoining phases are consistent
— The combined forward traces originating from each specification, fully support that specification
— Each current specification or feature is fully supported by traceable predecessor specifications

4.2.2 Evaluation

Evaluations ascertain the value or worth of an item and help to assure that a system meets its specifications. Evaluations are performed by many persons across all life cycle phases, on both interim and final software products, and may be either a comprehensive or selective assessment of a system.

Evaluations uncover problems in the different products and their relationships. These problems relate to the basic user need for the system to be fit for use in its intended setting. For a product to be fit for use, an evaluation may be used to assure that

— The product conforms to its specifications.
— The product is correct.
— The product is complete, clear, and consistent.
— Appropriate alternatives have been considered.
— The product complies with all appropriate standards.
— The product meets all specified quality attributes.

Evaluations not only identify problems, but may help to determine the progress of software development by recommending that the project

— Continue on to the next stage
— Correct specific items first, then continue
— Go back several steps and correct some problems
— Perform additional evaluations, such as a simulation, before that part of the system is further refined
— Monitor the progress of an item whose quality may be doubtful
— Make changes in methodology or software tools
— Make changes in schedules
— Make staffing decisions, such as additional resources, additional training, or changes in assignment

Evaluations are used through all phases and for all types of software products, including user documents, manuals, and other project documents. These may be of many forms, such as text or graphic representations, and in various media, such as paper, magnetic tape, diskette, and computer files. This range of product types and forms requires a large variety of techniques for performing and managing software evaluations.

When evaluating system products, there will be a change in emphasis from one development stage to the next. As the product progresses from a concept to a final product, its functional requirements are first defined. Next, the organization and structure of design elements are established. Then, all executable procedures are prepared. Questions of "what and why," "what-if," "alternatives," and "better-or-best" will predominate during the earlier development stages, while issues of "how," "correctness," and "conformance" will be more frequent in the later development stages.

When selecting evaluation strategies, the V&V planner first uses the features and characteristics of the system as a guide in the selection of the evaluation techniques. The planner shall question the system's functions, how they will be addressed, and what types of problems might be encountered. Second, the V&V planner derives additional evaluation techniques from quality attributes that may be specified for the system. Third, the V&V planner considers other evaluation methods, based on special V&V concerns and experience.

It may be helpful for the V&V planner to consider these three approaches when selecting techniques for evaluating the product:

a) Static analysis of the specification (e.g., reviews, inspections, structure analysis)
b) Dynamic analysis of expected software behavior (e.g., simulation, prototyping, screen emulation, branch execution analysis)
c) Formal analysis (e.g., mathematical proofs)

All these evaluation types can be used in conjunction with one another to provide a powerful synergism. Selection of a particular type of evaluation depends on the results required, tool availability, and cost trade-offs.

Static analysis detects deficiencies and errors through the examination of the software product, manually or automatically, without executing the software code. For example, static analysis can be applied to evaluate the form and content of the specification. It is usually straightforward. It frequently employs various types of reviews. Applicable to all levels of specification, it is able to detect flaws that could preclude or make meaningless the other forms of analysis, or prevent inadequate testing of the product. Static analysis focuses on the form, structure, and content of the product; its interfaces and dependencies; and on the environment in which it is to be used. Static analysis may be applied at all phases of development, and to all documented deliverables.

Dynamic analysis requires the execution of the software itself, or a model or simplified version of the software, to determine the validity of some of its attributes. Dynamic analysis can produce results that are not

available or are more time consuming with the other techniques. Dynamic analysis may also be more tractable than formal analysis, as there are a variety of tools expressly designed to support it.

Formal analysis uses rigorous mathematical proof techniques to analyze the algorithms or properties of the software. It can provide a strong conclusion regarding certain properties of an approach, but it is limited by the difficulty of its application and the general scarcity of automated support. Formal techniques generally require both a formal specification and a formal implementation. For example, the correctness of a natural, English-language requirement could not be proven. Certain assertions could be drawn from the requirement though, and proved to satisfy a set of conditions, thereby increasing the confidence that could be placed in the requirements for those conditions. Formal analysis is frequently used to verify sections of a specification that handle security requirements.

4.2.3 Interface analysis

When information is passed across a boundary (e.g., hardware to software, software to software, software to user) there is always the possibility of losing some information or altering the information content. The task of interface analysis serves to ensure the completeness, accuracy, and consistency of these interfaces. Interface requirements at the design and implementation phases should be identified and analyzed at the functional, physical, and data interface level. The goal of interface analysis is to evaluate the specific software deliverable (e.g., requirements, design, code) for correct, consistent, complete, and accurate interpretation of the interface requirements.

Interface analysis is concerned with data that flows from one part of the system to another. The sources of this information are important in assuring that the intended flow is feasible. Information about the recipient of the interface data is needed to assure that each interface is indeed necessary and sufficient. The correctness of the interface will rely on the use to which it is put, and its form and content will be part of the analysis.

Interface analysis should focus on three interface areas:

a) *User interface*. This should analyze what the human interfaces to the software product are, such as required screen format, protection mechanisms, page layout and content of the reports, relative timing of inputs and outputs, etc.
b) *Hardware interface*. This should analyze what the logical characteristics of each interface are between the software product and the hardware components of the system. Identify electronic devices, firmware, communication devices, and output devices. Then identify applicable standards for these interfaces and verify the current application's interface.
c) *Software interface*. This should analyze what the interfaces to other required software products are (e.g., data management system, operating system, or a library package), and interfaces with other application systems. Identify applicable standards for any other software products and interfaces to other application systems. Verify correct software interfaces to them.

In planning interface analysis consider the following:

— *Are the interface objectives technically adequate and well understood?*
— *Are all data elements well defined?*
— *Are all restrictions and constraints clearly defined?*
— *Were all the different kinds of interfaces taken into account and properly described?*
— *Are all hardware-to-software functional interfaces specified in quantitative terms such as units, bits per second, message formats, priority rules, word length, timelines, and protocols?*
— *Are all software-to-software interfaces functional, and are all data interface levels specified in quantitative terms, such as data timeliness, data definition, data formats, priority rules, message content, and communication protocols?*
— *Are the interface performance requirements well defined, and are the limits specified?*

— *Is the criticality of the interface taken into consideration?*
— *What are impacts if the interface is degraded?*
— *Are the interfaces testable and maintainable?*
— *Have all appropriate standards been identified for the interfaces, including those from the current software application to its environment?*

4.2.4 Testing

> *Testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.*

In the context of software verification and validation, testing can be defined as the testing that is performed in support of the V&V objectives. These objectives may differ from those of the developer. For example, V&V testing may focus on a narrow or critical area or on ensuring compliance to planned testing by the developers.

The scope and organizational responsibilities for the testing defined in an SVVP will depend on project-specific considerations. Once V&V objectives are established, the testing performed is in support of those objectives. Recognize there may be other testing activities beyond those of software verification and validation and there may be other organizations involved. Testing performed as part of V&V may be performed by personnel who generated the software or by personnel independent of the original developers.

The remainder of this subclause provides a description of testing activities performed during software verification and validation. It is not meant to be a guide to testing, but to clarify the concepts and terminology of the Standard, and to be a basis for V&V test planning.

4.2.4.1 Levels of testing

Testing is performed at several points in the life cycle as the product is constructed component by component into a functioning system. These points in the life cycle can be viewed as levels of capabilities that need to be tested. The levels progress from the smallest or single component through combining or integrating the single units into larger components or stages. The Standard describes these levels as component, integration, system, and acceptance. Table 2 further describes these levels of testing.

4.2.4.2 Test planning

V&V test planning is a continuing activity performed throughout the life cycle, starting from the requirements phase up to the test phase. Test plans are developed for each level of testing. These plans correspond to the developed products of the phase. The requirements phase produces the system requirements specification and acceptance and system test plans. The design phase produces the system design document and the component and integration test plans. The implementation phase produces the software product and test cases, test procedures, and component level execution. In the test phase, acceptance test procedures are produced and integration system and acceptance tests executed. An additional benefit of test planning is the additional review of the requirements, design, and interface descriptions.

The Standard provides for the creation of the test plan components in stages at appropriate points and times in the life cycle. Table 3 shows the stages and sequences. Following the table is an explanation of the various test activities.

Table 2—Levels of testing

Testing	Definition in the Standard	Purpose	Traceability
Component Testing	Testing conducted to verify the implementation of the design for one software element (e.g., unit, module) or a collection of software elements.	Ensures program logic is complete and correct. Ensures component works as designed.	From each test to detailed design
Integration Testing	An orderly progression of testing in which software elements, hardware elements, or both are combined and tested until the entire system has been integrated.	Ensures that design objectives are met.	From each test to the high-level design
System Testing	The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.	Ensures that the software as a complete entity complies with its operational requirements.	From each test to the requirements
Acceptance Testing	Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.	Ensures that customers' requirements objectives are met and that all components are correctly included in a customer package.	From each test to customer-specific requirements

Table 3—Test activities in life cycle phases

Activities	Requirements	Design	Implementation	Test
Test Plan Generation	- System - Acceptance	- Component - Integration		
Test Design Generation		- Component - Integration - System - Acceptance		
Test Case Generation			- Component - Integration - System - Acceptance	
Test Procedure Generation			- Component - Integration - System	- Acceptance
Test Execution			- Component	- Integration - System - Acceptance

4.2.4.2.1 Test plans

A test plan spells out the scope, approach, resources, and schedule of the testing activity. It indicates what is to be tested, what is not to be tested, testing tasks to perform, people responsible, and risks. Key points in planning include

— Transitioning from one phase or level to another
— Estimating the number of test cases and their duration
— Defining the test completion criteria
— Identifying areas of risks
— Allocating resources

The testing tasks performed during the requirements phase involve planning for and generating the system and acceptance test plans. Since these tests attempt to demonstrate that both the operational concept and the requirements are satisfied, planning may begin as soon as the requirements are written, although completion of the test plans shall await finishing other requirements phase V&V tasks. More detailed levels of preparation of test procedures and test cases will be performed in later phases.

The system design document is a key input to planning for component and integration test in the design phase. By developing the test plans early, there is adequate time to put test resources in place and to ensure that any untestable design features are identified prior to implementation.

Identifying methodologies, techniques, and tools is part of planning for testing. Methodologies are derived from the overall testing strategy. Techniques and tools are identified according to the specified methodologies. The specific methodologies, techniques, and tools selected will be determined by the type of software (e.g., application, operating system) to be tested, the test budget, the risk assessment, the skill level of available staff, and the time available.

In planning for verification and validation testing, the V&V planner should answer the following questions:

— *Who is responsible for generating the different levels of test designs, cases, and procedures?*
— *Who is responsible for executing the different levels of tests?*
— *Who is responsible for building and maintaining the test bed?*
— *Who is responsible for configuration management?*
— *What are the criteria for stopping test effort?*
— *What are the criteria for restarting test effort?*
— *When will source code be placed under change control?*
— *Which test designs, cases, and procedures will be placed under change control?*
— *For which level of tests will anomaly reports be written?*

It is often difficult to determine when to stop testing or when a reasonable number of defects have been detected. In fact, it may be unreasonable to expect that all defects will be detected. Therefore, completion criteria should be provided as a guideline for test completion. There are at least two common types of completion criteria. One criterion, test case completion, requires that all resultant test cases are executed without observing defects. A second criterion is fault prediction. For example, since the objective of testing is to find defects, the second criterion might be to test until a predefined number of defects have been found (or, conversely, a number of successful tests has been executed).

4.2.4.2.2 Test design

The test plan is followed by the development of test designs in the design phase. Test designs refine the test plan's approach, identify specific features to be tested by the design, and define its associated test cases and procedures. Whenever possible, tests should be designed to be suitable for regression testing; that is, tests

previously executed and verified can then be repeated at a later point in development or during maintenance of the installed system.

Consider the following when developing test designs:

— Required features to be tested
— Specified quality attributes
— Load limits
— Stress tests
— Configurations that the software may need to support
— Compatibility with existent or planned system components
— Security of the system
— Storage limits of the software, and related data
— Performance (such as accuracy) response times, and throughput
— Installability, primarily in the user environment
— Reliability/availability to the specifications
— Recovery from software and data failures
— Serviceability requirements
— Users' guides
— Human factors for usability and acceptability
— Interfaces with other system components
— Hardware interfaces

4.2.4.2.3 Test case

The test cases and test procedures are developed in the implementation phase. A test case specifies actual input values and expected results. The goal in the generation of test cases is to exercise the component's logic and to set up testing scenarios that will expose errors, omissions, and unexpected results. In developing test cases, one desire is to produce the smallest set of cases possible that will still meet the goal. Use of a matrix relating requirements to test cases aids in determining completeness and overlap.

4.2.4.2.4 Test procedure

A test procedure identifies all steps required to operate the system and exercise the specified test cases in order to implement the associated test design. Refer to IEEE Std 829-1983 for a discussion of test procedures.

4.2.4.2.5 Test execution

Test execution is the exercising of the test procedures. Test execution begins at the component level of testing in the implementation phase. The remaining levels of testing (e.g., integration, system, acceptance) are executed in the test phase.

5. SVVP guidance

The following provides detailed information about each section of an SVVP. The information here is organized in the same order as an SVVP and written in accordance with the Standard. In most cases, the following subclauses begin by quoting the requirements of the Standard for that section. Immediately following is the section number or subsection number where this quote can be found in the Standard.

> *The Software Verification and Validation Plan (also referred to as the Plan) shall include the sections shown below to be in compliance with this standard. If there is no information pertinent to a section or a required paragraph within a section, the following shall appear below the section or paragraph heading together with the appropriate reason for the exclusion: This section/paragraph is not applicable to this plan. Additional sections may be added at the end of the plan as required. Some of the material may appear in other documents. If so, reference to those documents shall be made in the body of the Plan. [3]*

When preparing an SVVP, include the following additional information prior to the first section of the plan, when appropriate to the particular project.

a) *Cover or title page*. This will typically include the title of the plan, a document or configuration control identifier, the date, and the identification of the organizations by whom and for whom the SVVP is prepared.
b) *Signature lines or page*. In many cases, the SVVP will require the signature or signatures of the persons responsible for writing or approving the plan.
c) *Revision history.* If the SVVP is updated by issuing change pages, it is helpful to include a list of such pages at the front of the SVVP. A brief history of the document and change bars or some other form of attention to changed material are useful regardless of the distribution method.
d) *Preface or foreword*. This will typically include a brief description of the project and identification of related documents (e.g., project management plan, software quality assurance plan, configuration management plan).
e) *Table of contents*. A table of contents is required if the reader is to make efficient use of the plan.
f) *Lists of figures, tables, or illustrations*. If there are more than a few figures, tables, or illustrations, or if these are referenced at widely-scattered points in the text, reference lists similar to a table of contents should be included. These items are typically listed separately.
g) *Assumptions*. Indicate any assumptions on which the plan is predicated, such as anticipated deliveries, availability of input, or intermediate products.

Include the following material after the last section of the SVVP, when appropriate to the project:

— *Glossary.* Define project-specific terms and acronyms.
— *Procedures*. Provide a clear understanding of the SVVP. In many cases, it is useful to separate the high-level planning information from the detailed procedures. These procedures may also be provided in a separate document referenced in the body of the SVVP.
— *Report formats and standard forms*. Furnish samples of reporting forms that will be useful. If large numbers of standard forms and report formats are to be included, these are best placed in an appendix.

5.1 Purpose

> *(Section 1 of the Plan.) This section shall delineate the specific purpose and scope of the Software Verification and Validation Plan, including waivers from this standard. The software project for which the Plan is being written and the specific software product items covered by the Plan shall be identified. The goals of the verification and validation efforts shall be specified. [3.1]*

See annex A, A.3.1.

The purpose statement of the SVVP provides the highest level description of the software V&V effort. The following topics should be addressed in the purpose statement:

— Identify the project to which the SVVP applies and describe why the plan is written.
— State the goals to be satisfied by the SVVP. For example, a specific V&V effort may intend to validate that all security requirements are satisfied. Another plan may be concerned only with validating performance requirements.
— Clearly summarize the V&V effort and the responsibilities conveyed under this plan.
— Define the extent of the application of the SVVP to the software. Explicitly cite each part of the software to which the SVVP applies and each part not covered by the SVVP. For example, perhaps only two of three subsystems of a computer system will fall under the SVVP (the third component may be unchanged from a previous system), or the SVVP may be invoked only for design verification.
— Identify waivers and changes if the Standard's requirements have been tailored for this SVVP. Examples include items such as addition or removal of sections or use of a different life cycle model. Sufficient detail should be included to demonstrate compliance with the Standard if the SVVP is audited. Descriptions of extensive changes may be placed in an appendix or a separate tailoring document.

5.2 Referenced documents

(Section 2 of the Plan.) This section shall identify the binding compliance documents, documents referenced by this Plan, and any supporting documents required to supplement or implement this Plan. [3.2]

This section of the SVVP should

— Specify documents completely, using any serial or publication numbers available.
— Make certain the correct version of the document is cited.
— Note clearly changes to referenced document list when SVVP is revised.
— List all documents referenced in the plan.
— Be consistent with other referenced document lists in other project documentation or explicitly indicate differences and reasons.

5.3 Definitions

(Section 3 of the Plan.) This section shall define or provide a reference to the definitions of all terms required to properly interpret the Plan. This section shall describe the acronyms and notations used in the Plan. [3.3]

For this section of the SVVP

— Include only terms necessary for understanding the SVVP.
— Keep the list short by referring to lists of definitions in other project documentation.
— If separate lists are maintained in different project documentation, be sure to use the same definitions in each list. It is helpful, to this end, to have a single database of project definitions and acronyms.
— Use definitions from IEEE Std 610.12-1990 and the series of IEEE glossaries where possible.

5.4 Verification and validation overview

(Section 4 of the Plan.) This section shall describe the organization, schedule, resources, responsibilities, tools, techniques, and methodologies necessary to perform the software verification and validation. [3.4]

5.4.1 Organization

(Section 4.1 of the Plan.) This section shall describe the organization of the V&V effort. It shall define the relationship of V&V to other efforts such as development, project management, quality assurance, configuration or data management, or end user. It shall define the lines of communication within the V&V effort, the authority for resolving issues raised by V&V tasks, and the authority for approving V&V products. [3.4.1]

See annex A, figure A.3.

The specific organizational structure of the V&V effort will depend on the nature of the system under development, the developing and acquiring organizations, and the contractual arrangements. Each organization responsible for a V&V effort may want to arrange the V&V effort differently. A V&V effort is simply the set of V&V tasks defined by the SVVP to be performed to achieve a quality product. Organization of the effort is then the assignment of each task to some appropriate person or organizational entity. There is no single, correct organization for all projects.

When planning for the organization of a V&V effort, consider the following:

— Assigning specific responsibility for each task (e.g., accepting the input, performing the task, analyzing the results, reporting the results, making decisions based on the results) where responsibility may be shared
— In the case of overlapping responsibilities, being precise about these assignments
— Using diagrams to show the control and data flow of V&V efforts to clarify responsibilities

5.4.2 Master schedule

(Section 4.2 of the Plan). This section shall describe the project life cycle and milestones, including completion dates. It shall summarize the scheduling of V&V tasks and shall describe how V&V results provide feedback to the development process to support project management functions (for example, comments on design review material).

If the life cycle used in the Plan differs from the life-cycle model in the standard, this section shall show how all requirements of the standard are satisfied (for example, cross reference for life-cycle phases, tasks, inputs, and outputs). When planning V&V tasks, it should be recognized that the V&V process is iterative. The summary of tasks may be in narrative, tabular, or graphic form. [3.4.2]

The master schedule summarizes the various V&V tasks and their relationships within the overall project environment. The objective is to spell out the orderly flow of materials between V&V activities and project tasks. This helps to ensure that V&V tasks are appropriately placed and their deliverables are identified within the larger project environment. Be aware that the development of the master schedule will be an iterative process.

In developing the master schedule, the V&V planner should focus on V&V tasks and their placement within the project schedule and highlight the key V&V tasks, deliverables, and completion dates. If an independent V&V effort is being conducted, the interfaces for delivery of materials, reviews, completion meetings, etc. need to be highlighted. There are many formats for schedule presentation (e.g., Gantt Charts, PERT, CPM) and in some cases analysis of schedule flow. The approach used should be consistent with other project elements.

5.4.3 Resource summary

(Section 4.3 of the Plan.) This section shall summarize the resources needed to perform the V&V tasks, including staffing, facilities, tools, finances, and special procedural requirements such as security, access rights, or documentation control. [3.4.3]

An overview of the resources needed for the V&V effort should be presented. The planner should not repeat the resource requirements of the individual tasks; instead, overall requirements should be summarized and potential conflicts for resources, long-lead items, inefficient uses of resources, and alternative resource options should be identified. For very small plans, all resource information could be placed in this section, rather than in the task discussions. It may be valuable to collect the detailed resource information in an appendix.

Resource types include labor or staffing, facilities, equipment, laboratories, the configuration of the laboratories, tools (both software and hardware), budget and financial requirements, documentation, special procedures and conditions (such as, security, access rights, and/or controls).

— Use graphs and tables as an effective means of presenting resource use. Graphs of resource use versus calendar time or life cycle phase give a quick grasp of the relative amounts of effort involved in the different tasks or phases. Tables of resource use, again by calendar time or life cycle phase, aid in comparing resource use with a budget or with the requirements of other project efforts.
— Include in the equipment and laboratories summary the type of equipment needed, duration needed, particular configurations, and other peripheral facilities that will be needed to perform the total V&V operations.
— In the tools section of the summary list the various tools that are to be used throughout the V&V effort. The tools can be subdivided into software and hardware.
— In the budget and financial requirements, take all the resources into account and allow for additional tools and staff to cope with contingencies.

5.4.4 Responsibilities

(Section 4.4 of the Plan.) This section shall identify the organizational element(s) responsible for performing each V&V task. It shall identify the specific responsibility of each element for tasks assigned to more than one element. This section may be a summary of the roles and responsibilities defined in each of the life-cycle phases. [3.4.4]

See annex A, table A.1.

There are two levels of responsibility for the V&V tasks—general responsibilities assigned to different organizational elements throughout the project and specific responsibilities for the tasks to be performed. A summary of the general responsibilities may be described in this section of the SVVP or in another project-level plan (e.g., a project management plan). If described in another document, this section should contain a summary and a reference to the other document. The specific responsibilities may be described in this section, or this section may summarize the responsibilities defined in the life cycle phase sections of the SVVP.

5.4.5 Tools, techniques, and methodologies

(Section 4.5 of the Plan.) This section shall identify the special software tools, techniques, and methodologies employed by the V&V effort. The purpose and use of each shall be described. Plans for the acquisition, training, support, and qualification for each shall be included. This section may reference a V&V Tool Plan. [3.4.5]

Describe the V&V approach, tools, and techniques and their roles in the V&V effort. This may be in narrative or graphic form. References to technique or tool descriptions should be included. A separate tool plan

may be developed for a software tool acquisition, development, or modification. In this case this section of the SVVP should refer to the tool plan. If a tool is to be acquired or developed, its acquisition or development schedule should be included in the V&V schedule. Determine whether sufficient time and appropriate tasks are allowed for tool acquisition or development.

When planning the use of tools, techniques, and methodologies, consider the following:

a) A description of, or reference to, the methodology selected for the V&V approach
b) Staff experience and training needed
c) Special tools and specific techniques for the methodology
d) How each tool and technique enhances the methodology
e) Risks associated with a tool or technique
f) Status of each tool
    1) *Is it a new acquisition?*
    2) *Are changes needed or is it completely ready for use?*
    3) *Is the required quantization available?*
    4) *Is its documentation acceptable?*
    5) *Is the tool proprietary?*
g) Acquisition or development schedule
h) Necessary support (hardware, other software)
i) Alternate approach for a high-risk tool

5.5 Life cycle verification and validation

This subclause parallels 3.5 of the Standard, using the same life cycle model. The Standard points out that any life cycle model can be employed for development and V&V activities. The Standard can then be adapted to it as long as there is a clear mapping from one life cycle model to the other.

While many tasks can be thought of as going across life cycle phases, most SVVPs are likely to be written and managed with tasks assigned to a particular life cycle phase. Each of the following subclauses amplifies the requirements of the Standard. (See 4.2 of this guide for further guidance on planning the tasks that go across life cycle phases.)

Additionally, the Standard defines management as one of the phases. Of course, management is not a single life cycle phase activity, but rather an activity that goes across all of the individual phases and that ties all of the activities together into a coherent, useful program.

5.5.1 Management of V&V

> *(Section 5.1 of the Plan.) This section of the Plan shall address the seven topics identified in section 3.5 of this standard. The management of V&V spans all life-cycle phases. The software development may be a cyclic or iterative process. The V&V effort shall reperform previous V&V tasks or initiate new V&V tasks to address software changes created by the cyclic or iterative development process. V&V tasks are reperformed if errors are discovered in the V&V inputs or outputs.*
>
> *For all software, the management of V&V shall include the following minimum tasks:*
>
> 1. *Software Verification and Validation Plan (SVVP) Generation*
> 2. *Baseline Change Assessment*
> 3. *Management Review of V&V*
> 4. *Review Support [3.5.1]*

The primary tasks of V&V management are planning, review, and control. The Standard maps these responsibilities into four fundamental tasks for V&V management—SVVP Generation, Baseline Change Assessment, Management Review, and Review Support. Conceptually, management's responsibility is to ensure the positive, successful interaction between the V&V activities and other software development activities to ensure the creation of defect-free software. The planning of the management tasks shall recognize the iterative nature of V&V tasks (e.g., determining when to reanalyze amended software products, when to revise V&V plans to reflect changes in the development process).

5.5.1.1 Software Verification and Validation Plan (SVVP) generation

> *Generate SVVP (during Concept Phase) for all life cycle phases in accordance with the standard based upon available documentation Include estimate of anticipated V&V activities for Operation and Maintenance Phase. Update SVVP for each life cycle phase, particularly prior to Operation and Maintenance. Consider SVVP to be a living document, and make changes as necessary. A baseline SVVP should be established prior to the Requirements Phase. [Table 1, 5.1 (1)]*

See annex A, A.3.4.

The Standard specifies the required content and format for an SVVP. From this starting point, V&V planners should tailor these requirements to the operating environment and the development environment. Tailoring considerations include the following:

- Software development environment and methodology
- Size and complexity of software being verified and validated
- Risk management considerations
- Organization and number of personnel performing V&V tasks
- Relationship of personnel performing V&V tasks to other development personnel, management, and user
- Approvals required for the SVVP
- Method of configuration control, maintenance, and updating of SVVP
- Special focus of V&V required (e.g., safety or security)

V&V planners may be dependent on things outside of their control. As a consequence, the plan may need to be able to respond to occasional, controlled changes to support the overall project milestones and objectives. At the same time it should remain constant in its quality goals and V&V objectives. Some of the uncertainties that should be considered include the following:

- Timely delivery of products
- Whether the quality of the products will be sufficient to allow meaningful verification and validation
- Availability of staff and resources needed to perform the planned tasks

V&V planning is most effectively performed in conjunction with the overall software development planning. Revisions to the plan are prepared regularly throughout the software development life cycle, usually at the completion or initiation of each life cycle phase. The plan, to be effective as a complete specification of the tasks required to achieve the established V&V objectives, shall be thorough, comprehensive, and specific.

The SVVP is often best developed incrementally. An SVVP will start small, based on an early, perhaps incomplete, view of the project. The SVVP will grow and incorporate changes as the software is developed. Eventually, the SVVP will be completed and serve as a record of the complete and detailed plan for the project's V&V activities.

This should not be taken to mean that the plan should be continually revised throughout the life cycle phases. Only those changes needed to provide further or more detailed task management for the future or to make corrections should be made. Revisions to the plan should not be made retrospectively; that is, as a way of removing deviations from the plan. Deviations that arise during the execution of the plan, necessary for project changes or external factors, are addressed in the Software Verification and Validation Reports (SVVRs).

Incremental planning may be prudent for at least two reasons.

a) *It allows for a more realistic view of what is possible in planning in the early stages of a project*. Too often, much time and attention is spent overplanning future life cycle phases when there is the likelihood of significant changes in those phases. For example, certain design and implementation questions may be unanswerable in the concept phase and, as a consequence, precise planning for design and implementation of V&V may not be possible. As decisions are made and results and products obtained throughout the life cycle, more careful planning of subsequent phases is meaningful.
b) *If a plan is to be effective (e.g., if people are to use it to identify responsibilities, to prepare for future tasks, to measure past performance) it should be realistic and achievable*. It should reflect the changes that are inherent in the other development process activities. It cannot be a static view of the desires of management at the outset of the V&V effort without regard to actual developments. The plan should represent reasonable and achievable objectives.

The SVVP is thus referred to as a living document because it may undergo changes and expansion over time. At its first publication (in most cases at the beginning of the life cycle) it will contain detailed plans about the near-term activities and only cursory descriptions of the later phases. When generating input for the SVVP, be prepared to revise and update the SVVP at fixed points. Most effectively, plan for review and revision at the end of each life cycle phase.

Additionally, during life cycle phases it sometimes may be advisable to incorporate revisions to the plan in response to significant changes in other project plans, and to include references to those revised plans. Plan for the maintenance of the SVVP under configuration identification and control. This can be accommodated by maintaining the SVVP on electronic media or in a hard copy format that can be easily updated.

The SVVP should be an item under configuration management. Plan to establish the baseline SVVP prior to the Requirements Phase if V&V begins early enough. In other cases, baseline the SVVP no later than at the end of the current life cycle phase. The use of a configuration identifier and change notations (such as change bars) will allow effective control of the configuration and status of the document.

5.5.1.2 Baseline change assessment

> *Evaluate proposed software changes (for example, anomaly corrections, performance enhancements, requirements changes, clarifications) for effects on previously completed V&V tasks. When changes are made, plan iteration of affected tasks which includes reperforming previous V&V tasks or initiating new V&V tasks to address the software changes created by the cyclic or iterative development process. [Table 1, 5.1 (2)]*

Baseline change assessment may be the most dynamic task performed during a V&V effort. Any change proposal could affect an unknown amount of previously completed development and V&V work. A V&V task should examine the change to determine the nature and extent of the V&V rework required by the change. Because changes will be proposed asynchronously, the SVVP cannot, before the fact, contain detailed planning for any given change assessment; rather, it shall contain general guidance for allocating resources to change proposals as they occur.

The request for baseline change assessment should be documented in a change proposal. The change proposal should indicate the perceived severity of the anomaly (if any), the immediacy of the change, and the systems affected so that the criticality of the software can be determined.

5.5.1.3 Management review of V&V

Conduct periodic reviews of V&V effort, technical accomplishments, resource utilization, future planning, risk management. Support daily management of V&V phase activities, including technical quality of final and interim V&V reports and results. Review the task and V&V phase summary reports of each life-cycle phase. Evaluate V&V results and anomaly resolution to determine when to proceed to next life-cycle phase and to define changes to V&V tasks to improve the V&V effort. [Table 1, 5.1 (3)]

See annex A, A.3.5.

Management review comprises all of the general responsibilities of management for the monitoring, controlling, and reporting of the V&V effort and, in effect, the managing of this plan. In this task, V&V management is responsible for creating the positive interaction between the V&V activities and the other development activities. V&V management is responsible, in particular, for reviewing on-going efforts, accomplishments, and the use of resources. This section of the SVVP addresses only V&V-specific issues in project management review. While not stated earlier, a manager sufficiently technically expert in the system under development and in the process of software V&V is necessary to make informed and useful decisions throughout the project. This need is most keenly felt in the task of management review.

The results and findings of V&V tasks will have a significant impact on the development effort. (See also 5.7 of this guide for control and administrative procedures relating to the distribution of V&V products.) Early identification and correction of defects is one obvious benefit. The monitoring and reporting of levels and types of defects may identify other quality engineering issues in the development process.

Careful management attention to the technical quality and accuracy of the V&V products is particularly important. It is important that inaccurate or incorrect reports are not distributed to development or management personnel lest unnecessary effort be expended responding to illusory or imprecisely stated anomalies.

When reviewing reports, consider the following:

a) *Is the report complete?*
b) *Is the report technically accurate?*
c) *Is the report judicious in its use of language?*
    1) *Does it restrict itself to the technical issues under discussion?*
    2) *Is it restricted to the tasking direction?*
    3) *Is its tone positive?*
    4) *Does it support development success?*
d) *Does the report support effective management decision-making?*
e) *Does it discriminate between levels of severity?*
f) D*oes it clearly identify impact?*
g) *Does it propose alternate approaches?*

Prepare summary documents to make them as useful as possible to development and acquiring organization management. When organizing the results, consider the following:

— Summary assessment
— Itemization of critical outstanding problems and the impact of their not being resolved
— Recommendations about corrective course of action
— Summary of rescheduling impact, V&V, and resource requirements
— Distribution of anomalies according to some predefined categorization scheme and the significance of this distribution

5.5.1.4 Review support

Correlate V&V task results to support management and technical reviews (for example, software requirements review, preliminary design review, critical design review). Identify key review support milestones in SVVP. Schedule V&V tasks to meet milestones. Establish methods to exchange V&V data and results with development effort. [Table 1, 5.1 (4)]

Formal review meetings are a common means of evaluating and approving the products of one life cycle phase before going on to the next phase. V&V can contribute to the effectiveness of management and technical reviews through a variety of functions and tasks. In most cases, these functions and tasks will be similar between one review meeting and the next. As a result, this section of the SVVP need not have a detailed plan for each individual review to be supported; rather, it may contain general guidance when preparing for a review, together with such review-specific information as seems necessary (e.g., listing the contents of the review package for each review).

When planning the review support activities, consider the following:

— Reviewing documentation packages for compliance with specified review requirements; completeness, consistency, and traceability; and identification of open items
— Developing agenda items to cover unresolved anomalies, deficiencies; unresolved open items; and technical alternatives and trade-off studies

Participation in reviews by personnel performing V&V tasks may involve the following:

— Asking questions to clarify points or identify unresolved issues
— Raising issues from prior analysis of review packages
— Assessing impact of V&V on changes under discussion in the review

Note that where an independent V&V agent is used, the developer's contract may need to contain provisions to ensure the availability of necessary materials and access to those performing IV&V.

5.5.2 Concept phase V&V

(Section 5.2 of the Plan.) This section of the Plan shall address the seven topics identified in section 3.5 of this standard. For critical software, Concept Phase V&V shall include the following minimum V&V task: Concept Documentation Evaluation. [3.5.2]

The concept phase establishes the reason for the system. The concept defines the nature of the system that will be developed and enumerates its goals and its risks within technical and business constraints.

Evaluation in the concept phase should establish that the objectives of the system define the user needs to be addressed and the technical and business advantages that are expected. These objectives may be stated in a variety of ways, and may include a statement of need, a business case, feasibility studies, and a system definition.

V&V establishes that risks and constraints enumerate any technical, business, or policy considerations that may impede development of the system. Initial planning should be included and should outline the staffing, time, and cost expected for each alternative. In addition, any regulations and policies that govern the system and its development will be stipulated. Alternative approaches may be included, with their respective advantages and disadvantages.

5.5.2.1 Concept documentation evaluation

Evaluate concept documentation to determine if proposed concept satisfies user needs and project objectives (for example, performance goals). Identify major constraints of interfacing systems and constraints or limitations of proposed approach. Assess allocation of functions to hardware and software items, where appropriate. Assess criticality of each software item. [Table 1, 5.2 (1)]

The actual V&V evaluation performed during the Concept Phase should delineate specific quality goals and areas of risk. These should be reflected in the full-scale development V&V plan in addressing risk mitigation, resource allocation, and selection of methods and criteria. Of particular interest to risk mitigation would be instances where there is a potential imbalance between technical scope and allocated cost and schedule resources for the overall project and the subsequent V&V activities. In addition, the criticality of each software item should be addressed.

5.5.3 Requirements phase V&V

(Section 5.3 of the Plan.) This section of the Plan shall address the seven topics identified in 3.5 of this standard.

For critical software, Requirements Phase V&V shall include the following minimum tasks:

(1) Software Requirements Traceability Analysis
(2) Software Requirements Evaluation
(3) Software Requirements Interface Analysis
(4) Test Plan Generation
  (a) System Test
  (b) Acceptance Test [3.5.3]

The requirements phase is the period of time in the software life cycle during which the requirements, such as functional and nonfunctional capabilities for a software product, are defined and documented. The main product of the requirements phase, the Software Requirements Specification (SRS), should accurately state the software mission, that is, what the software is intended to do. The SRS should be traceable back to the user needs and system concept, as defined in concept documentation. It should be traceable forward through successive development phases and representations, into the design, code, and test documentation. It should also be compatible with the operational environment of hardware and software. The requirements should provide both qualitative and quantitative constraints on subsequent design and implementation options. There are five types of requirements—functional, external interfaces, performance, design constraints, and quality attributes.

An SRS typically describes or specifies several of the following:

— *The system boundary*. What is within the required software and what is outside.
— *The software environment*. Conditions of the surroundings that are imposed on the software (e.g., interfaces, response times, availability, size).
— *The software functions*. What the software is to do and how the software should respond to its environment.
— *The software constraints*. Imposed limits that are placed on the software and its stimulation and responses.
— *The software interfaces*. The nature of the information flow across software boundaries, where this information is found, and under what conditions.
— *The software data*. The contents of the information flows with their formats, and relationships.

— *The software algorithms*. Detailed descriptions of the software algorithms and their conditions in the most applicable terms.
— *The software states*. Stable modes that the software may assume, under which conditions, and with what actions.
— *The software error conditions*. What constitutes a departure from the norm, under which conditions, and what actions to take.
— *The software standards*. Those forms of representation and content of the requirements that are required by the development and user organizations.
— *The hardware interfaces*. What the software must do to transfer data across hardware boundaries and The software quality attributes. The conditions that the software must meet in order to be considered fit for use in its intended application. The attributes will always include conformance to specifications, correctness, and compliance with standards. There may be other quality attributes (e.g., reliability, safety) depending on the type of system and its use.

Requirements verification is concerned with assuring that each item in the SRS conforms to what is wanted, correct, complete, clear, consistent, and, as appropriate, measurable and testable. See also IEEE Std 830-1984.

5.5.3.1 Software requirements traceability analysis

Trace SRS requirements to system requirements in concept documentation. Analyze identified relationships for correctness, consistency, completeness, accuracy. [Table 1, 5.3 (1)]

See annex A, A.3.6. See also 4.2.1.

One goal of the requirements traceability analysis is to establish that the SRS completely satisfies all of the capabilities specified in the concept document(s). A second is to determine which requirements satisfy each need in the concept. Another goal is to establish that the SRS is structured so that the requirements may be traced through subsequent development stages. The SRS traceability analysis makes sure that all of the necessary parts of the software system are specified. Further, it determines where those parts are so that they can be followed in later development steps, and that there are no untraceable requirements.

The plan should identify the following:

— The format of the concept, SRS, interface documentation, and their releasing organizations
— Indexing and cross-reference schemes that are established as part of the requirements specification to facilitate traceability analysis
— Criteria for extracting and identifying discrete requirements from narrative documents
— Acceptance conditions for the SRS, including the criteria for release to configuration control

There may be some requirements that are not directly traceable to other documents. It can be expected that each succeeding development phase will present new information that must be traced to following phases, but are not explicitly present, or only hinted at, by preceding specifications (e.g., concurrence of tasks, standard error recovery procedures). Some of these may be called for by the SRS, but may not be present in the concept documents. All derived requirements should be identified and included in the traceability matrix.

Plan to establish the traceability of the SRS prior to completing the other V&V tasks for this phase. If the traceability analysis is incomplete it is inconclusive to evaluate the SRS, or to perform an interface analysis, or to plan testing. Although these other V&V tasks may be performed concurrently with traceability analysis, they cannot be considered finished until assurance is provided that all of the SRS has been traced.

5.5.3.2 Software requirements evaluation

Evaluate SRS requirements for correctness, consistency, completeness, accuracy, readability, and testability. Assess how well SRS satisfies software system objectives. Assess the criticality of requirements to identify key performance or critical areas of software. [Table 1, 5.3 (2)]

See annex A, A.3.7. See also 4.2.2.

The objectives of the software requirements evaluation are to assess the technical merits of the requirements, to determine that the requirements satisfy the software objectives defined in the concept phase, and to assure that the specifications are correct, complete, clear, and consistent. An additional objective is to determine if any requirements are missing. Where there is an absolute measure available for a requirement (e.g., response time), establish the correctness of the requirement with respect to the measure. The evaluation should establish the technical adequacy of the requirements. The SRS should be evaluated to ensure that all requirements are testable against objective criteria.

Depending on the scope and complexity of the SRS, there may be multiple evaluations. Each evaluation will address a specific purpose and may use specialized techniques and specific participants. A policy may need to be defined that coordinates multiple evaluations of an SRS and correlates their results.

In addition, the scheduling of the evaluation may be determined by the availability of specific portions of the SRS that are required by one or another evaluation. In practice, there are usually parallel efforts underway that will use some portion of the SRS, or an uncertified version, to proceed with either a more detailed level of development, such as high level design, or with an interfacing product. In all cases, though, the limits of this work should be defined.

5.5.3.3 Software requirements interface analysis

Evaluate the SRS with hardware, user, operator, and software interface requirements documentation for correctness, consistency, completeness, accuracy, and readability. [Table 1, 5.3 (3)]

See annex A, A.3.8. See also 4.2.3.

The goal of the requirements interface analysis is to assure that all external interfaces to the software and internal interfaces between software functions are completely and correctly specified. As software reuse and standard software components are more frequently used, the importance of internal interface analysis will increase.

Depending on the form of the SRS and on the tools available, manual or automated analysis of the interfaces may be employed. Mechanisms should be provided to ensure the following:

— The sources and recipients of interface data are accurately described.
— Protocols for transferring and receiving data across interfaces are accurate and complete.
— The SRS interfaces are compatible with the interface documentation.

In addition to the SRS, the documents describing the external interfaces, such as interface definition documents, data dictionaries, and related SRS documents, should be included. Techniques for analyzing internal interfaces between individual requirements should also be included.

The acceptance criteria for the interface documents should be specified, including release to configuration control. Criteria include compliance to format guidelines to ensure traceability, complete resolution plans for all incomplete requirements (TBDs), and notice of completion of interface analysis.

5.5.3.4 System test plan generation and acceptance test plan generation

Plan System testing to determine if the software satisfies system objectives. Criteria for this determination are, at a minimum: (a) compliance with all functional requirements as complete software end item in system environment (b) performance at hardware, software, user, and operator interfaces (c) adequacy of user documentation (d) performance at boundaries (for example, data, interface) and under stress conditions. Plan tracing of system end-item requirements to test design, cases, procedures, and execution results. Plan documentation of test tasks and results. [Table 1, 5.3 (4a)]

Plan acceptance testing to determine if software correctly implements system and software requirements in an operational environment. Criteria for this determination are, at a minimum: (a) compliance with acceptance requirements in operational environment (b) adequacy of user documentation. Plan tracing of acceptance test requirements to test design, cases, procedures, and execution results. Plan documentation of test tasks and results. [Table 1, 5.3 (4b)]

See annex A, A.3.9 and A.3.10. See also 4.2.4.

The V&V testing tasks performed during the requirements phase involve planning for and generating the system and acceptance test plans. Since these tests attempt to demonstrate that both the operational concept and the requirements are satisfied, planning may begin as soon as the requirements are written, although completion of the test plans will generally await finishing the other requirements phase V&V tasks. More detailed levels of preparation of test procedures and test cases will be performed in later phases.

System and acceptance test plans will benefit from any matrices or other organized sets of information that are used for traceability and interface analysis. This information should be designed to include references to test plans and test procedure elements. Test plans should be evaluated and verified to assure that all planned conditions and features of the software under development are tested sufficiently to meet the V&V objectives.

For system testing, the primary goal is to validate that there are no defects among and omissions from the software, the concept document, and the system requirements specification. Specific areas of testing (e.g., performance, security, reliability, availability) may need to be planned.

For acceptance testing, the primary goal is user validation that the software complies with expectations, as reflected by the operational concept functional requirements, and quality attributes. Additional goals of acceptance testing are to establish that the software can be successfully installed and operated by the intended user, is appropriately documented, and that it can be maintained. Wherever possible, it is recommended that the user or user representatives be involved in establishing appropriate acceptance test plans. Both the system and acceptance test plans should identify the following, in addition to general testing requirements:

— All the input documentation required for system testing as well as the protocol for transferring these documents to the V&V staff. These documents should include system requirements specifications, interface requirements documents, operational scenarios, and users manuals.
— The stopping criteria for testing. These could include the fraction of requirements covered, number of errors remaining metrics, statistical reliability goals, etc.
— Provisions for witnessing tests.

5.5.4 Design phase V&V

(Section 5.4 of the Plan.) This section of the Plan shall address the seven topics identified in Section 3.5 of this standard. For critical software, Design Phase V&V shall include the following minimum V&V tasks:

(1) Software Design Traceability Analysis
(2) Software Design Evaluation
(3) Software Design Interface Analysis
(4) Test Plan Generation
  (a) Component Test
  (b) Integration Test
(5) Test Design Generation
  (a) Component Test
  (b) Integration Test
  (c) System Test
  (d) Acceptance Test

Table 1 contains a description of the minimum Design Phase V&V tasks and identifies the required inputs and outputs. The inputs and outputs required to accomplish the minimum V&V tasks shall include, but not be limited to, those listed in Table 1. [3.5.4]

The design phase is the period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements. Removing errors from the designs during this phase will substantially reduce the occurrence of defects in the subsequent code, and will lessen both product and project risk later in the life cycle. Design V&V also indirectly provides an opportunity to locate faults not previously detected in the requirements.

Although simple products may be designed in a single step, design frequently occurs as a multiple-step process. The first level of the design specifies architectural features (e.g., major subsystems and their interfaces). Successive design steps evolve by adding detail until the subsystems have been sufficiently specified for coding to begin. Designs may be represented in many forms, including text, graphical descriptions, pseudo-code representation, or combinations of these and others.

In addition to solving the problem of how to organize and build the required system, designers have a wide range of quality objectives that include:

— Tracing the requirements through all design levels, and ensuring that there are no omissions or additions
— Structuring the design so that it is appropriate to the system objectives and the desired product quality attributes
— Describing all hardware, operator, and software interfaces
— Documenting that the design conforms with all applicable standards, practices, and conventions
— Establishing that the design will satisfy the requirements when fully integrated
— Documenting the design so that it is understandable to those who write the source code and later maintain the product
— Including sufficient information in the design to plan, design, and execute tests
— Controlling the design configuration and ensuring that all documentation is completed and delivered, especially when mixed media are used (e.g., graphic charts, text specifications)

Meeting these objectives provides assurance that all the requirements are represented in the design, that the design will satisfy the requirements, and that the design is testable and will lead to testable code. The responsibility of the V&V planner is to select V&V tasks for the design products, including intermediate specifications, to ensure that these objectives are in fact met.

The V&V planner selects the V&V tasks and their accompanying methods that are appropriate for each level of design for the specified features. While the V&V tasks are repeated through each design level, the V&V methods or techniques used may change. For example, where evaluation of the choice of an algorithm may be appropriate at high-level design, mathematical analysis of the algorithm may be more appropriate at detailed design.

The scope of the V&V effort will be determined by the complexity of the design effort. When planning the V&V tasks for the design phase, consider the following:

— Responsibilities levied by the project plan (e.g., to support the critical design review)
— Design methodology
— Design standards
— Critical or difficult sections of the design
— Design assumptions that require proofs (or references to proofs)
— The presence of complex algorithms that may require extensive analysis
— Resource restrictions (e.g., available computer hardware, timing limitations) requiring sizing and performance analysis
— Database privacy and access requirements needing security analysis
— The level of the design (i.e., different V&V tasks and methods may be appropriate for high-level and detail design)
— The different approaches that will be needed for component and integration testing
— The organization performing the design V&V tasks
— The media and format of the design

V&V schedules should accommodate comments and anomalies emerging from the design V&V tasks that will be addressed. Additionally, potential risks may emerge. An updated list of these identified risks should be maintained, and they should then be reflected by the developing test plans.

5.5.4.1 Software design traceability analysis

Trace SDD to SRS and SRS to SDD. Analyze identified relationships for correctness, consistency, completeness, and accuracy. [Table 1, 5.4 (1)]

See annex A, A.3.11. See also 4.2.1.

Whether manual or automated procedures are used, perform the physical trace in both directions between requirements and design. Analyze the trace to ensure that

— No requirements are omitted or designed more than once.
— No extraneous design pieces exist.
— A requirement addressed by more than one design element is completely and consistently satisfied.

Evaluate each of the relationships for correctness, consistency, completeness, and accuracy. Ensure that

— All conditions of a requirement have been designed.
— The technical relationship between a requirement and its design is correct.
— Any multiple relationships that are detected between requirements and design are necessary.

5.5.4.2 Software design evaluation

Evaluate SDD for correctness, consistency, completeness, accuracy, and testability. Evaluate design for compliance with established standards, practices, and conventions. Assess design quality. [Table 1, 5.4 (2)]

See annex A, A.3.12. See also 4.2.2.

Evaluate each element of each level of the design. Some of the optional tasks in the Standard may be used for design evaluation. Consider the project needs and their relationships with other project efforts when selecting V&V methods. The methods and techniques that are selected will depend on characteristics of the project. Examine the total project to select the methods for design evaluation that will locate the design errors that present the most risk to the project's success. The results of criticality analysis can help identify those design elements that require more intense evaluation.

5.5.4.3 Software design interface analysis

Evaluate SDD with hardware, operator, and software interface requirements for correctness, consistency, completeness, and accuracy. At a minimum, analyze data items at each interface. [Table 1, 5.4 (3)]

See annex A, A.3.13. See also 4.2.3.

Perform design interface analysis on each level of the software design. When planning design interface analysis, consider the following:

- *Is the use of the data items (i.e., the inputs and outputs) consistent and complete?*
- *Is the interface correct, complete, and necessary?*
- *Are the data items used correctly in the design element where sent?*
- *Are system resources being used properly in the data transfer (e.g., are calls being made by value that should be made by reference)?*
- *For user interfaces, is the interface design understandable? Will the software provide appropriate help? Will the software detect user errors and provide clear error responses?*
- *Has a prototype been built for a critical interface (particularly the user interface)? If so, has it been thoroughly and independently evaluated? Does it demonstrate the critical features properly?*
- *Are the interfaces designed for effective configuration management?*
- *Is an interface needed where there is none?*

5.5.4.4 Component test plan generation and integration test plan generation

Plan component testing to determine if software elements (for example, units, modules) correctly implement component requirements. Criteria for this determination are, at a minimum: (a) compliance with design requirements (b) assessment of timing, sizing and accuracy (c) performance at boundaries and interfaces and under stress and error conditions (d) measures of test coverage and software reliability and maintainability. Plan tracing of design requirements to test design, cases, procedures, and execution results. Plan documentation of test tasks and results. [Table 1, 5.4 (4a)]

See annex A, A.3.14. See also 4.2.4.

Plan integration testing to determine if software (for example, subelements, interfaces) correctly implements the software requirements and design. Criteria for this determination are, at a minimum: (a) compliance with increasingly larger set of functional requirements at each stage of integration (b) assessment of timing, sizing, and accuracy (c) performance at boundaries and under stress conditions (d) measures of functional test coverage and soft-

ware reliability. Plan tracing of requirements to test design, cases, procedures, and execution results. Plan documentation of test tasks and results. [Table 1, 5.4 (4b)]

See annex A, A.3.15.

Use the SDD to plan for component and integration test. A benefit of test planning in the design phase is that adequate time is allowed to put test resources in place and to ensure that any untestable design features are identified prior to implementation.

Component testing demonstrates the adequacy of the functions and quality attributes of each component of the software design. The focus is on software flow patterns and functionality of individual components. The tests could target on accuracy of solutions, data handling, etc. The coverage of the test cases should be analyzed to determine level of coverage of tests to code statements and paths.

The approach taken for integration testing depends on the system design. Integration testing validates the structure of the design, or how well the components perform with other components and fit into the developing system structure. Once the design has been determined, integration test planning should begin to ensure readiness of all test materials and the adequate allocation of testing resources.

5.5.4.5 Test design generation

Design tests for: (a) component testing (b) integration testing (c) system testing (d) acceptance testing. Continue tracing required by the Test Plan. [Table 1, 5.4 (5)]

See annex A, A.3.16–A.3.19.

All of the functional, performance, and user interface tests can now be designed, but not yet executed. Resulting test designs should be traceable from the concept documentation to the system requirement specification, to the system design document, or to other user documentation as appropriate.

5.5.5 Implementation phase V&V

(Section 5.5 of the Plan.) This section of the Plan shall address the seven topics identified in 3.5 of this Standard.

For critical software, Implementation Phase V&V shall include the following minimum V&V tasks:

- *(1) Source Code Traceability Analysis*
- *(2) Source Code Evaluation*
- *(3) Source Code Interface Analysis*
- *(4) Source Code Documentation Evaluation*
- *(5) Test Case Generation*
  - *(a) Component Test*
  - *(b) Integration Test*
  - *(c) System Test*
  - *(d) Acceptance Test*
- *(6) Test Procedure Generation*
  - *(a) Component Test*
  - *(b) Integration Test*
  - *(c) System Test*
- *(7) Component Test Execution [3.5.5]*

The implementation phase is the period of time in the software life cycle during which a software product is created from design documentation and then debugged. In the implementation phase, the V&V tasks are focused on the code and the determination of how well it conforms to the design specifications and coding standards. The objective of V&V in this phase is to determine the quality of the code.

The quality of the code can be determined in several ways. The design specifications are traced to the corresponding code in the program. Code is traced back to the design requirements. These steps are performed to ensure that no requirements have been added, modified, or missed. The program interfaces are analyzed and compared to the interface documentation. A detailed program evaluation is performed to analyze the program for the correct translation of the design specification and conformance to program coding standards. Another activity in the implementation phase is the generation of test cases and test procedures for the various levels of testing (e.g., component, integration, system, and acceptance). The test procedures for the component level of testing will be executed.

5.5.5.1 Source code traceability analysis

> *Trace source code to corresponding design specification(s) and design specification(s) to source code. Analyze identified relationships for correctness, consistency, completeness, and accuracy. [Table 1, 5.5 (1)]*

See annex A, A.3.20. See also 4.2.1.

The goal of traceability analysis from source code to design components and from design components to source code is to identify possible inconsistencies between design and code. This comparison identifies anomalies such as, source code without antecedent design components, design components that do not clearly associate with the source code, or any other issues or abnormalities that emerge from this process. This forward and backward tracing identifies completeness and accuracy of the code. Plan to perform the physical trace in both directions between design components and code.

Analyze the trace to ensure that:

- No design specifications are omitted or coded more than once
- No extraneous code pieces exist
- A design specification addressed by more than one code element is completely and consistently satisfied

5.5.5.2 Source code evaluation

> *Evaluate source code for correctness, consistency, completeness, accuracy, and testability. Evaluate source code for compliance with established standards, practices, and conventions. Assess source code quality. [Table 1, 5.5 (2)]*

See annex A, A.3.21. See also 4.2.2.

Several evaluation techniques can be used depending on the level of risk or criticality desired (e.g. reviews, walkthroughs and inspections could be used). Refer to 4.2.2 for general discussion of evaluation techniques.

When planning source code evaluations, consider the following:

- Using criticality analysis of the code components to determine which code to evaluate
- Ensuring that the coding standards understood
- Ensuring that coding standards are available to staff before coding begins
- Determining how to evaluate code quality attributes
- Identifying code analysis tools that could be used

5.5.5.3 Source code interface analysis

Evaluate source code with hardware, operator, and software design documentation for correctness, consistency, completeness, and accuracy. At a minimum, analyze data items at each interface. [Table 1, 5.5 (3)]

See annex A, A.3.22. See also 4.2.3.

The goal of source code interface analysis is to assess the information and control flow between and within components. The results of the source code traceability analysis may be used in conjunction with this task. The output of this task describes component interface inconsistencies and errors. Interface analysis is effective at detecting errors that can be difficult to isolate in testing. Examples of these errors are

— Software elements that are referenced but not defined
— Software elements that are defined but not referenced
— Incorrect number of arguments
— Data type mismatches
— Data constraint mismatches
— Other data usage anomalies
— Control flow inconsistencies

When planning source code interface analysis, consider the following:

— Physical units precision
— Coordinates references
— Granularity
— Control flow
— Timing requirements
— Data transfer
— Naming conventions

5.5.5.4 Source code documentation evaluation

Evaluate draft code-related documents with source code to ensure completeness, correctness, and consistency. [Table 1, 5.5 (4)]

See annex A, A.3.23.

The goal of evaluating source code documentation with source code is to ensure that the documentation correctly reflects the actual implementation of the source code. Code-related documents could include program support manuals, user manuals, and operations manuals. Due to the stage of the life cycle, these documents are frequently in draft form and will be modified. When planning for source code documentation evaluation consider the following:

— Reviewing for user functionality in addition to technical accuracy, completeness, and consistency
— Making sure the documentation is written at the appropriate level for the audience
— Making sure usability issues (such as index, table of contents, glossary, heading, format, etc.) are correct
— Seeing if a new user can use the documentation to perform the task/job
— Making sure the documentation can handle error correction processes
— Making sure there is follow-up to complete the draft documents

5.5.5.5 Test case generation

Develop test cases for: (a) component testing (b) integration testing (c) system testing (d) acceptance testing. Continue tracing required by the Test Plan. [Table 1, 5.5 (5)]

See annex A, A.3.24–A.3.27.

A test case documents the actual values used for input along with the anticipated outputs. Test cases are developed for each level of testing (component, integration, system, acceptance).

a) *Component test cases* exercise as many paths through the code as are feasible. The V&V planner shall determine and plan for coverage analysis. A tool may be needed to aid in determining coverage.
b) *Integration test cases* focus on testing the interfaces and interdependencies of the components that have been combined.
c) *System test cases* focus on the combination of tests of the computer programs, manual controls, and procedures as they will exist in the new system environment. System test cases will focus on areas such as performance, security, reliability, and human factors.
d) *Acceptance test cases* demonstrate to the user that the system meets the objectives and performance expectations agreed upon by the developers and users of the system.

Refer to 4.2.4 of this guide for general discussion of test cases. Also, refer to IEEE Std 829-1983 for further discussion of test case generation.

5.5.5.6 Test procedure generation

Develop test procedures for: (a) component testing (b) integration testing (c) system testing. Continue tracing required by the Test Plan. [Table 1, 5.5 (6)]

See annex A, A.3.28–A.3.30.

Refer to 4.2.4 of this guide for general discussion of test procedures. Also, refer to IEEE Std 829-1983 for further discussion of test procedure generation.

5.5.5.7 Component test execution

Perform component testing as required by component test procedures. Analyze results to determine that software correctly implements design. Document and trace results as required by the Test Plan. [Table 1, 5.5 (7)]

See annex A, A.3.31.

The component test should demonstrate the internal integrity and correctness of the component. The tests should exercise paths defined between components, and should exercise the data interfaces supported through shared database elements and components.

The planner needs to consider the following:

a) Scheduling test runs
b) Executing predefined test cases
c) Evaluating test results
d) Issuing status reports

5.5.6 Test phase V&V

(Section 5.6 of the Plan.) This section of the Plan shall address the seven topics identified in 3.5 of this standard.... For critical software, Test Phase V&V shall include the following minimum V&V tasks:

(1) Acceptance Test Procedure Generation
(2) Test Execution
(a) Integration Test
(b) System Test
(c) Acceptance Test.[3.5.6]

The test phase is the period of time in the software life cycle when 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. During the Test Phase, V&V activities include

— Generation of acceptance test procedures
— Performance of integration testing according to previously developed test procedures
— Analysis of integration test results
— Performance of system testing according to previously developed test procedures
— Analysis of system test results
— Performance of acceptance testing in accordance with test procedures that have been placed under configuration control
— Generation and resolution of anomaly reports

Planning for the test phase V&V activities should be accomplished within the context of 4.2.4 of this guide. Thus, the planning for test procedures generation, test performance, and results analysis is highly dependent on such factors as organization, schedule, resources, responsibilities, tools, techniques, and methodologies. The nature of the organization and how verification and validation tasking is performed within the organization may determine how successful the test phase will be.

The overall software development schedule will determine when the test phase will begin and approximately how long the test effort will take. The schedule should be evaluated periodically to ensure that enough time is allocated to the test phase. Some practitioners suggest, as a rule of thumb, that enough time includes time to run each test at least three times—the first time the test itself is tested, the second time the source code is tested, and the third time is for regression testing.

Proper coordination and adherence to deadlines are important during this phase. Large projects may involve many people and various organizations. For example, the development organization or third-party vendors may be providing hardware or firmware. Additionally, the client may be providing required information, data, or software. All of them may be working simultaneously, and independently, on complex, interrelated tasks. Because of the interdependencies, it is essential that deadlines be met. It is also important that the entire effort be coordinated through such control mechanisms as change control, configuration management, and reporting.

5.5.6.1 Acceptance test procedure generation

Develop test procedures for acceptance test. Continue tracing required by the Test Plan. [Table 1, 5.6 (1)]

See annex A, A.3.32.

Acceptance test procedures specify the steps for executing a set of test cases or, more generally, the steps used to analyze a software item in order to evaluate a set of features. Refer to 4.2.4 of this guide for general discussion of test procedures. Also, refer to IEEE Std 829-1983 for further explanation of test procedures.

5.5.6.2 Test execution

During the test phase, components are progressively integrated. The integrated components are progressively tested according to the test plan, case, and procedures. At each level of testing, the software is taken from and then returned to a controlled environment under configuration management.

5.5.6.2.1 Integration test execution

Perform integration testing in accordance with test procedures. Analyze results to determine if software implements software requirements and design and that software components function correctly together. Document and trace results as required by the Test Plan. [Table 1, 5.6 (2a)]

See annex A, A.3.33.

Integration testing focuses on testing the interfaces and interdependencies of the components that have been combined. Integration testing validates the structure of the design, that is, how well the components fit into the developing system structure and perform with other components.

5.5.6.2.2 System test execution

Perform system testing in accordance with test procedures. Analyze results to determine if software satisfies system objectives. Document and trace all testing results as required by the Test Plan. [Table 1, 5.6 (2b)]

See annex A, A.3.34.

The system test is a consolidated test of the computer programs, manual controls, and procedures as they will exist in the new system environment. System testing ensures that the procedures work according to the system specifications. The object of the system test is to uncover deviations from the system's business requirements. This means testing the system against its specifications and requirements. System testing focuses on areas such as performance, security, reliability, and human factors.

5.5.6.2.3 Acceptance test execution

Perform acceptance testing in accordance with test procedures under formal configuration control. Analyze results to determine if software satisfies acceptance criteria. Document and trace all testing results as required by the Test Plan. [Table 1, 5.6 (2c)]

See annex A, A.3.35.

Successful completion of acceptance testing demonstrates to the user that the system meets the objectives and performance expectations agreed upon by the developers and users of the system. The acceptance test should not only verify the accuracy and completeness of all software but also ensure that user procedures, user training, and operational considerations have been addressed. A part of the testing will be of the documentation. Publications that accompany the system may also need to be tested.

Before acceptance testing can take place, the following must occur:

a) Support equipment is in place and ready.
b) Required personnel are available according to plan.
c) Test documentation is completed and ready for use.
d) All application components are tested, integrated, and available.
e) Data is prepared.

After the acceptance test has been executed, a test report may be written. The test report may include a summary of the acceptance test, an evaluation of the status of the application (pass/fail), a statement of the results of each test, and conclusions and recommendations.

5.5.7 Installation and checkout phase V&V

(Section 5.7 of the Plan.) This section of the Plan shall address the seven topics identified in Section 3.5 of this standard.

For critical software, Installation and Checkout Phase V&V shall include the following minimum V&V tasks:

(1) Installation Configuration Audit

(2) Final V&V Report Generation. [3.5]

The installation and checkout phase is the period of time in the software life cycle during which a software product is integrated into its operational environment and tested to ensure that it performs as required. Characteristics of installation include the installation staff, the duration of the installation process, the number of installation sites, the number of system versions, and the adequacy of their configurations.

Installation procedures place a completed and tested application into an operations environment in a manner that satisfies user requirements for using the system. Sometimes this process is separate from development and is performed by an organization different from that which developed and tested the application (e.g., by field or customer support engineers). In some cases, installation and its checkout are performed by the software end user. This is typically the case in the personal computer software industry. In many cases, installation occurs in a very short span of time, sometimes in an hour or in a few hours. At that time operation of the software in its production environment is expected to commence. Installation sometimes occurs in installments where one or more subsystems are added to the initial delivery, with a user acceptance testing period occurring after each installation. Each subsystem should be appropriately integrated with previous subsystems before release for acceptance testing.

Installation may be repeated an indefinite number of times, once for each site for each software version. There may be considerable tailoring of a product for each site (e.g., setting site-dependent parameters, building site-dependent command tables); each site may have different installation procedures.

Installation and checkout V&V procedures can ensure that:

— Each site-dependent configuration is correct and complete
— The instructions for installation at each site are an accurate and complete representation of the configuration
— The instructions are appropriately written for the expertise level(s) of the expected installer(s)
— User-oriented checkout tests are adequate to demonstrate satisfactory installation
— The software that has been verified and validated is the same software that has been delivered for installation

Responsibility for installation and checkout V&V may be split between the development and user organizations. The users may take a large role in the installation. Any V&V activities assigned to the user organization should be well defined and documented. Pre-installation V&V establishes and verifies the procedures for user V&V at installation.

Scheduling installation and checkout V&V may be difficult, for all the reasons listed above. If the software is to be installed at a large number of sites, it may be necessary to develop a schedule for a generic installation in advance, to be tailored to each specific installation as required by time or resources. A procedural flowchart or checklist may be valuable.

When planning for installation and checkout V&V, consider the following:

— *Verifying accuracy and completeness*. Ensure the integrity of the data before, during, and after installation with accuracy and completeness controls. For example, if a data file is to be reformatted, plan a test to demonstrate that the integrity of the file is preserved by reformatting. If control totals are maintained, verify the final controls against the initial controls.
— *Maintaining and verifying an installation audit trail*. Verify that all processes and changes that occur during installation are recorded.
— *Assuring the integrity of a previous system*. In many cases, the software being installed is a replacement for an existing system. Verify that the installation process allows the existing system to continue operation until the new system is formally accepted and declared operational. It may be necessary to operate the two systems in parallel for some period of time, or to maintain the old system in case the new system fails.
— *Verifying compliance to installation or checkout standards*. Ensure that the installation and checkout is performed in accordance with appropriate standards, procedures, and guidelines.

Frequently used optional tasks for installation and checkout V&V include, but are not limited to, the following:

a) *Regression analysis and testing*. Changes occurring from installation and test are reviewed. Regression analysis and testing verifies that basic requirement and design assumptions affecting other areas of the program have not been violated.
b) *Simulation*. Operator procedures are tested. Simulation tests also help to isolate any installation problems. This technique may be especially useful before delivery when many site-dependent versions of the software will be delivered.
c) *Test certification*. Test certification is used, especially in critical systems, to demonstrate that the software product is identical to the software product that was subjected to V&V.

It is important that the results of any installation tests be available prior to the completion of installation. The objective of this testing (or checkout) is to determine whether or not the installation is successful. As with other tests, this frequently means that the test results should be predicted before the test starts.

5.5.7.1 Installation configuration audit

> *Audit installation package to determine that all software products required to correctly install and operate the software are present, including operations documentation. Analyze all site-dependent parameters or conditions to determine that supplied values are correct. Conduct analyses or tests to demonstrate that installed software corresponds to software subjected to V&V. [Table 1, 5.7 (1)]*

This task can be divided into three smaller tasks:

a) *Configuration audit*. This determines that all software products required to correctly install and operate the software are present in the installation package. The installation package typically consists of an installation plan with its checkout procedures, cases, and predicted results, source code listings and other development documentation, operator instructions, and end-user documentation.
b) *Site-dependent parameter analysis*. This verifies that the software has been properly tailored for the installation site. This may require analysis and verification of parameter values, command tables, configuration files, or other means of tailoring the software to the site. If site-dependent code is developed, either for installation or as part of the operational system (e.g., device drivers), that software should be verified and validated during its development.
c) *Corroboration analysis*. This verifies that the installed software corresponds to that software subjected to V&V during development. This may require inspection of audit trails, file comparison, or other means of demonstrating that the delivered software matches the verified and validated software.

5.5.7.2 V&V final report generation

Summarize all V&V activities and results, including status and disposition of anomalies in the V&V final report. [Table 1, 5.7 (2)]

Although V&V should be performed for the life of the software, the initial V&V effort may be completed with the installation and checkout of the software. In other cases, different organizations will be performing V&V during maintenance, requiring a hand-off of documentation and responsibility. Whenever V&V is completed, a final report should be prepared to summarize the activities and results of the V&V effort. The format of a final report will be specified in section 6 of the Plan (see 3.6 of the Standard).

This report serves several purposes.

a) It serves as the starting point for V&V of the maintenance of the system.
b) It can be a vehicle for lessons learned, or a way for project personnel to improve the development or V&V process for the next project.
c) It can call attention to outstanding unresolved anomalies from development, installation, or operation.

The final report can typically be written by reviewing and summarizing the task reports, anomaly reports, and phase summary reports, written during the course of the V&V effort.

5.5.8 Operation and maintenance V&V

(Section 5.8 of the Plan.) This section of the Plan shall address the seven topics identified in Section 3.5 of this standard.

Any modifications, enhancements, or additions to software during this phase shall be treated as development activities and shall be verified and validated as described in 3.5.1 through 3.5.7. These modifications may derive from requirements specified to correct software errors (that is, corrective) to adapt to a changed operating environment (that is, adaptive), or to respond to additional user requests (that is, perfective).

If the software was verified under this standard, the standard shall continue to be followed in the Operation and Maintenance Phase. If the software was not verified under this standard, the V&V effort may require documentation that is not available or adequate. If appropriate documentation is not available or adequate, the SVVP shall comply with this standard within cost and schedule constraints. The V&V effort may generate the missing documentation.

For critical software, Operation and Maintenance Phase V&V tasks shall include the following minimum V&V tasks:

(1) Software Verification and Validation Plan Revision
(2) Anomaly Evaluation
(3) Proposed Change Assessment
(4) Phase Task Iteration [3.5.8]

The operation and maintenance phase is the period of time in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified, as necessary, to correct problems or respond to changing requirements.

Operation and maintenance is sometimes not so much a phase as sequence of repetitions of subsets of the life cycle. Multiple versions of a software product may be available and supported simultaneously. Each

version may be found at several sites, each with installation-specific parameters, device drivers, or other code. Different installations may upgrade versions at different times. The operations and maintenance phase may be further complicated by other considerations that depend strongly on the software product and its application and users.

Given this complexity, it is difficult to provide specific guidance for the V&V tasks of this phase, which are often repetitions of previously described tasks. Instead, guidelines are offered here for V&V planning during operation and maintenance.

There are two primary cases to consider:

a) The software was verified and validated (possibly under the Standard) when originally developed.
b) The software was not adequately or formally verified or validated when originally developed.

Planning for V&V during operation and maintenance shall reflect which of these two cases holds true for the software of interest.

The V&V planner has a number of advantages when the software under maintenance was verified and validated during its original development. Complete and current documentation should exist. Verification and validation procedures should be in place and effectively used. Finally, there should be an historical record to use as the basis of future planning.

These advantages can help the V&V planner offset some of the common pitfalls faced during maintenance. Many key development personnel may have left the project, leaving perhaps a reduced maintenance cadre. There may have been a formal transfer from a development organization to a maintenance organization with no connection to the original development. Users will be reporting problems and requesting new functions simultaneously, each with his or her own priority for implementation.

Where the software was not verified or validated before maintenance, the planner faces the above pitfalls as well as some new ones. Documentation may not exist or may not be up to date (this is possibly worse than no documentation at all). Developers and management may not welcome verification and validation. When V&V are being performed for the first time, there may be no basis for planning other than engineering judgment.

Critical software should be verified and validated during maintenance whether or not it was verified or validated during development. If it was verified during development, maintenance is the time where the investment in V&V provides further significant payback. If the software was not previously verified, V&V may be able to bring some order to the possibly perceived chaos, thus easing the lives of the maintenance personnel.

The Standard requires four tasks—software verification and validation plan revision, anomaly evaluation, proposed change assessment, and phase task iteration—specifically for the operation and maintenance phase.

Note that verification and validation without complete, up-to-date documentation can no more be performed in operation and maintenance than in any other phase. Considerable effort in operation and maintenance may be spent in preparing or updating inadequate documentation. When planning V&V for software with missing or inadequate documentation, either include resources for developing necessary documents or ensure that management, developers, and users are aware that the effectiveness of the V&V effort may be seriously compromised. Developing the missing documentation is most worthwhile if extensive new development or thorough understanding of the existing system is required.

5.5.8.1 Software verification and validation plan revision

For software verified and validated under this standard, revise SVVP to comply with new constraints based upon available documentation. For software not verified and validated under this standard, write new SVVP. [Table 1, 5.8 (1)]

Develop a new or revised SVVP regularly during maintenance, since the problems found in the software and the desired new functionality will be constantly changing. Depending on the organization, schedule replanning according to project needs (e.g., revise the SVVP for every major software version) or by time interval (e.g., revise the SVVP every six months).

5.5.8.2 Anomaly evaluation

Evaluate severity of anomalies in software operation. Analyze effect of anomalies on system. [Table 1, 5.8 (2)]

The only fundamental differences between anomalies found during development and those found during operation are the nature of the environment and the nature of the operators. During development, the software is typically operated by personnel technically knowledgeable of the software design and code. Actual operators, in contrast, may be unfamiliar with the design and code and may be operating an active environment with site-specific equipment, parameters, or code.

These differences affect the way anomalies are handled in maintenance. The descriptions of anomalies may be incomplete. Information on the environment is particularly hard to determine remotely. These two differences make it difficult to replicate the anomaly, which in turn makes resolution difficult.

For some systems, anomaly evaluation may be complicated by site-specific configuration and by the potential multiplicity of software versions. The V&V personnel shall determine the version of the software in which the anomaly occurred and in which other versions the same or similar anomaly could occur. An automated tool for recording and tracking anomalies, particularly if the tool is tied into the development documents and to the proposed change assessment process, can greatly assist the anomaly resolution. The anomaly tracking system could collect performance data as well.

When preparing this section of the SVVP, plan for anomaly resolution in general. Describe how V&V resources should be allocated to an anomaly when it occurs and as it is analyzed and resolved. The detailed procedures for recording, analyzing, tracking, and resolving anomalies should be found in section 7 of the plan.

5.5.8.3 Proposed change assessment

Assess all proposed modifications, enhancements, or additions to determine effect each change would have on system. Determine extent to which V&V tasks would be iterated. [Table 1, 5.8 (3)]

Change proposals typically come from one of two sources—maintainers (to correct anomalies, to improve the software, or to adapt to changed operating environments) or users (to provide more functionality). These changes need to be evaluated in different ways. Changes proposed by maintainers should be relatively easy to evaluate, since they can be referenced to existing code, requirements, and design documentation. Changes proposed by users will often require analysis to determine which requirements, design, and code would be changed and to what degree. Describe, in this section of the SVVP, how V&V resources will be allocated to evaluate the proposed changes.

Changes to software systems, particularly large or complex ones, should be carefully managed. This is typically implemented through a configuration management system with a configuration control board. Where

such a system is in place, plan V&V participation on the board. If such a system is not in place, consider having V&V act as configuration management. V&V is very difficult and very expensive in an environment of uncontrolled change.

Evaluate proposed changes to

a) Clarify an ambiguous or poorly worded proposal.
b) Reveal the true extent of a change.
c) Help management determine which changes to implement and on what schedule.
d) Determine how the change fits into the existing or desired system.

This is one area where V&V may perform a systems engineering role for the software development to prevent the uncontrolled growth of the software. Change evaluation is, in many respects, identical to the V&V evaluation of the concept phase, since many of the same planning and execution considerations apply.

Changes may be proposed as object-level patches to existing software, rather than as redesign and recode at the source level. These changes are particularly difficult to evaluate and debug. Where patching is unavoidable, the patches should be tracked (possibly with an automated database) and the source-level redesign and recode undertaken as soon as practical.

5.5.8.4 Phase task iteration

> *For approved software changes, perform V&V tasks necessary to ensure that: planned changes are implemented correctly; all documentation is complete and up to date; and no unacceptable changes have occurred in software performance. [Table 1, 5.8 (4)]*

The implementation of each change may be thought of as following its own life cycle. This life cycle will include a subset of the full life cycle, depending on the nature of the change. In each phase of this reduced life cycle, plan and execute the V&V tasks required by the Standard for that phase (e.g., if the change modifies design documentation, trace and evaluate the design changes and plan appropriate tests).

5.6 Reporting

> *(Section 6 of the Plan.) This section shall describe how the results of implementing the Plan will be documented. V&V reporting shall occur throughout the software life cycle. This section of the Plan shall specify the content, format, and timing of all V&V reports. These V&V reports shall constitute the Software Verification and Validation Report (SVVR). [3.6]*

V&V reporting communicates the status of the V&V effort and its findings among all interested parties. Those concerned with the results of the V&V activities might be quite diverse as a variety of different persons and organizations may be assigned responsibility for performing V&V. The diversity of the audience requires tailoring the distribution, format, and content of the reports. V&V reports can provide added visibility into the development process and more complete understanding of the product, but only to the extent that the participants in the V&V process share the V&V information.

The timing of the V&V reports is geared to their function and audience. The great cost multiplier for software errors is their latency period, the time between the introduction of a defect into a software product and its identification and subsequent removal. The greatest benefit of V&V in being performed in parallel to the development of the software comes from prompt notification to the developers of any problems identified by V&V.

All V&V activities need to be reported. Planning for V&V requires that sufficient time be allocated for reporting, although specific reporting requirements will vary with the activity. V&V reports may have a

variety of names and formats, such as memos, presentations, marked-up copies of reviewed documents, minutes of meetings, action items, status reports, assessment and analysis reports, review notations, audit reports, inspection reports, test reports, anomaly reports, failure reports, etc. All of these fall into the broad categories of either being required by the Standard or optional.

5.6.1 Required reports

The hierarchy of the four types of required V&V reports ranges from individual findings about specific deficiencies in the software (anomaly reports) to a comprehensive evaluation of the entire development project (final V&V report). The four types of required V&V reports are as follows:

a) *Anomaly reports*. These reports should be promptly forwarded to developers. Quick turnaround requires established procedures to record and characterize identified anomalies, and then to provide the relevant information to the developers. Details of the anomaly reports include descriptions of each anomaly, a severity scale for each, rework approval mechanics, and required administration of the anomaly. (See also 5.7.1 of this guide.)
b) *V&V task reporting*. These reports follow the performance of each well-defined V&V task. Task reports may be viewed by V&V project management as proof of the progress of the V&V effort. Task reports also provide key insights into the need for any mid-course corrections to the development or V&V plans and are put into the V&V management review (see 5.5.1). The relative efficiencies and difficulties of certain V&V tasks may lead to tactical adjustments, such as choice of methodologies and tools. Further, the actual task results may call for strategic changes as well, including reallocation of V&V resources or alteration of schedules. Task reporting, its content and scheduling, should address the needs of one task requiring the outputs of another as input (for example, when test planning requires the results of traceability analysis), and be scheduled to allow an orderly and timely progression of the V&V tasks.
c) *V&V phase summary reports*. These reports serve to summarize and consolidate the results of the V&V activities performed by different organizations for each life cycle phase. The phase summary report may be an extensive formal document or a brief informal notification, depending on the span and depth of the V&V activities performed within a particular phase. The report may also be a briefing delivered as part of a phase review such as a preliminary design review. If the information is too critical to await reporting at the conclusion of the entire phase, interim reports may be generated in a more timely fashion.
d) *V&V final report*. This report summarizes and consolidates the results of the software anomalies, V&V tasks, and all V&V phase reports. The V&V Final Report is prepared at the conclusion of the V&V effort and will provide an assessment of overall software quality and any recommendations for the product and/or the development process.

5.6.2 Optional reports

Frequently, a need arises for more than the four required reports. These optional reports are either the result of some special V&V study or some other, unanticipated activity. Given the unique nature of these reports, there are no specific guidelines for their format or contents. As with all V&V reports, though, the optional reports need to be timely and properly distributed. They should identify the purpose of the V&V activity and describe the approach used, and report their results at an appropriate level.

5.7 Verification and validation administrative procedures

> *(Section 7 of the Plan.) This section of the Plan shall describe, at a minimum, the V&V administrative procedures described in 3.7.1 through 3.7.5. [3.7]*

For a V&V effort to make its maximum contribution to the overall project, it should function in an unambiguous way with a controlled flow of information, orderly processes, and an ability to adapt to unanticipated

situations. Clear, well-defined administrative procedures are an essential ingredient of a successful V&V plan. This section of the SVVP should specify such procedures in sufficient detail to allow for their successful implementation.

At a minimum, the administrative procedures shall address anomaly reporting and resolution, task iteration policy, deviation policy and control procedures, as well as standards, practices, and conventions. The necessary procedures can often be adopted, or tailored for the specific V&V effort, from established procedures, policies, standards, conventions, and practices.

This section of the SVVP should identify any existing administrative procedures that are to be implemented as part of this V&V plan, and any procedures that are to be written and implemented as part of the V&V effort. Existing procedures will also be identified as references in 2 of the SVVP. This section provides the opportunity to interpret the use of the referenced procedures, and to describe any new ones that may be planned or under development.

This section should identify the life cycle phase(s) and V&V task(s) to which each procedure will be applied. The degree of implementation of each procedure should be stated. This section should also indicate which individual or organizational element will be responsible for the enforcement, evaluation, and maintenance of each procedure, and specify how compliance will be monitored and assured.

Diagrams may be provided to show the relationships among the various responsible organizations. To be effective, the administrative procedures should be consistent with the organization and responsibilities defined elsewhere in the SVVP.

A V&V effort consists of both management and technical tasks, and at least three audiences can be identified for the information generated during the project—personnel performing V&V tasks, personnel performing development tasks, and management. In order to assure that the different information needs of each audience are satisfied, the administrative procedures may include one or more distribution lists. Information flow can be facilitated, and many misunderstandings avoided, if the appropriate distribution of development documentation, formal V&V documentation, memos, meeting minutes, status reports, etc., is established in advance.

5.7.1 Anomaly reporting and resolution

> *(Section 7.1 of the Plan.) This section shall describe the method of reporting and resolving anomalies, including the criteria for reporting an anomaly, the anomaly report distribution list, and the authority and time lines for resolving anomalies. This section shall define the anomaly criticality levels. Each critical anomaly shall be resolved satisfactorily before the V&V effort can formally proceed to the next life-cycle phase. [3.7.1]*

The Standard defines an *anomaly* as anything observed in the documentation or operation of software that deviates from expectations based on previous verified software products or reference documents.

The SVVP should describe clear and unambiguous procedures for anomaly reporting and resolution, so that each participants can determine their roles in the process. Reporting and resolving anomalies as early in the development process as possible is one obvious benefit. Specific methods for documenting anomalies and their resolution, including the use of anomaly report forms, should be provided.

In addition to anomaly reporting and resolution tracking, the anomaly reporting process can be a primary means of data collection for the software verification and validation effort. The number and criticality level of the anomalies may determine whether the effort can formally proceed to the next life cycle phase. Data for process monitoring activities such as root-cause analysis also rely on data from the anomaly reporting process.

5.7.1.1 Methods and criteria for reporting

The Standard requires that the SVVP include criteria for reporting an anomaly. The circumstances under which the anomalies will be formally reported should be clearly defined. Disruption of the development process by unnecessary notification should be avoided. Anomaly reporting processes vary from informal to formal and from manual to fully automated reporting and tracking. Project size and criticality issues determine the sophistication of the system.

V&V planning should address the role of informal comments in the anomaly reporting process. Issues may include whether informal comments are to be allowed at all. If informal comments are allowed, the following should be addressed:

— The form in which the informal comments should be (e.g., memo, telephone call)
— When to use informal comments (e.g., for early notification, with anomaly reports to follows)
— The classes of problems (e.g., minor or out of current scope) for which informal comments are sufficient and formal notification is not required

While informal comments are often useful, care should be taken to assure that all significant problems are documented formally and brought to the attention of the appropriate personnel. All anomalies should be documented and reported.

V&V planning should specify

— Who is responsible for recording anomalies and analyzing their impact, criticality, etc.
— Who has the responsibility and authority for approving anomaly reports for issue

In some cases, V&V activities are performed in conjunction with software quality assurance activities, e.g., V&V personnel provide audit support, attend review meetings, monitor tests. In these cases, the SVVP should specify whether V&V anomaly reports are required to duplicate reporting of problems by other groups. For example, if V&V personnel monitor tests and find the testing organization's test incident reports to be adequate, there may be no need for separate V&V anomaly reports for the testing. Any instances of inadequate reporting by the testers would, however, be the subject of V&V anomaly reports in this case.

5.7.1.2 Anomaly report distribution

According to the Standard, the SVVP is to include an anomaly report distribution list. The planned distribution of anomaly reports should be clearly defined, including specification of who gets each report, under what circumstances, and for what reasons.

Reports should be distributed where needed for information and for tracking, as well as for action. If reports are to be distributed on a priority basis, the priority levels should be defined and the distribution criteria established.

The distribution list will depend on the organization and degree of independence of the V&V effort. For example, if the personnel performing V&V tasks are independent of the development group, anomaly reports should be distributed to both the development group and the user (or the development group's higher level management).

5.7.1.3 Methods and criteria for anomaly resolution

The resolution of an anomaly may result in a change to documentation, software, or hardware. The resolution of an anomaly can be a complicated, subjective, and resource-consuming task. The Standard requires that

> *Each critical anomaly shall be resolved satisfactorily before the V&V effort can formally proceed to the next life cycle phase*.

The plan shall specify the procedures for determining the criticality of the anomaly, determining the impact of the anomaly, and for resolving differences between the originator of the anomaly report and personnel responsible for resolving the anomaly.

Responsibility and authority with respect to anomaly reporting should be specified in the SVVP. These should include the following:

— Responsibility for responding to anomaly reports
— Authority for evaluating responses and resolving anomalies
— Responsibility for tracking the status (open vs. resolved) of anomaly reports

To be effective, the specification of responsibility and authority in this area shall be consistent with the organization and responsibilities defined elsewhere in the SVVP. (See also 5.4.1, 5.4.4, and C.7 of this guide.)

5.7.1.4 Timing

V&V planning should address timing considerations for reporting anomalies, as well as for their resolution. As indicated in 5.6 of this guide, anomaly reports should be forwarded promptly. Daily or even more frequent reports may be provided in some instances. Early identification and correction of defects is one obvious benefit of timely notification. Early notification of V&V results should be balanced against possible disruption of the development process.

Effective procedures are needed to assure that anomaly reports are valid and necessary as well as timely. The SVVP may require that anomalies be grouped and held for distribution according to some predefined categorization scheme, so that relationships among anomalies are more easily seen. If so, the maximum allowable hold time should be specified, which may be different for the different categories.

According to the Standard, the SVVP is to include time lines for resolving anomalies. A normal assumption is timely response to informal comments (if applicable) and anomaly reports. The SVVP should indicate the time frame that can be considered to be timely.

5.7.2 Task iteration policy

> *(Section 7.2 of the Plan.) This section shall describe the criteria used to determine the extent to which a V&V task shall be reperformed when its input is changed. These criteria may include assessments of change, criticality, and cost, schedule, or quality effects. [3.7.2]*

The software products that are input to the V&V effort are often changed as, for example, the results of anomaly corrections, performance enhancements, requirements changes, and clarifications. When changes are made, V&V tasks are iterated by repeating previous tasks or initiating new ones. V&V task iteration is needed to ensure that planned changes are implemented correctly, all documentation is complete and up to date, and no unacceptable changes in software performance have occurred.

Administrative procedures should include criteria for appropriate allocation of V&V resources as changes are evaluated. Without such procedures resources might be overextended in one or more areas, to the detriment of the project as a whole.

The criteria used to determine the extent to which V&V tasks are to be repeated when their inputs change may include assessments of change; criticality; and cost, schedule, or quality effects. Historical data may also be helpful in this planning.

Issues to be considered in selecting criteria may include the following:

a) *What portion(s) of development documentation will be re-evaluated in response to a change? For example, would revision of the Software Requirements Specification require review of the entire document, or of only those portions identified by the developers as revised in response to anomaly reports?*
b) *What degree of regression testing will be necessary when the software is changed? (Component? Integration? System? Acceptance?)*
c) *Is it necessary to repeat the V&V tasks until all anomalies are resolved?* (For noncritical anomalies, this may not be necessary or practical.)
d) *Will this V&V effort be continued into operations and maintenance? If not, at what point are changes considered to be maintenance and beyond the current V&V scope?*

5.7.3 Deviation policy

(Section 7.3 of the Plan.) This section shall describe the procedures and forms used to deviate from the Plan. The information required for deviations shall include task identification, deviation rationale, and effect on software quality. This section shall define the authorities responsible for approving deviations. [3.7.3]

Project changes or external factors may make it necessary to deviate from the SVVP. Any such deviations should be documented and approved before they are allowed to occur.

A standard form may be prepared, including task identification, deviation rationale, and effect on software quality. Tracking information may also be included. The personnel preparing and approving the deviation request should be identified. For smaller, less formal projects, it may be sufficient to document the required information in memo form.

Authority for preparing and approving requests for deviation from the SVVP should be comparable to those for writing and approving the SVVP itself. Consider software criticality when defining the necessary levels of authority.

Deviations in the execution of the SVVP are reported in the SVVRs. The approved deviations may be documented in the V&V final report by including copies of the forms or memos, or by listing forms or memos available separately for reference. Deviations that apply to a given task or phase should also be documented in the corresponding V&V task report or phase summary report.

It is not appropriate to remove past deviations by retroactive revision of the SVVP. However, some deviations affect the V&V tasks that are planned for the future. If a deviation is expected to occur again, or if it will have a significant effect on remaining activities, consider revising the SVVP. Subclauses 4.1 and 5.5.1.1 include guidance on revising the SVVP.

5.7.4 Control procedures

(Section 7.4 of the Plan.) This section shall identify control procedures applied to the V&V effort. These procedures shall describe how software products and results of software V&V shall be configured, protected, and stored.

These procedures may describe quality assurance, configuration management, data management, or other activities if they are not addressed by other efforts. At a minimum, this section shall describe how SVVP materials shall comply with existing security provisions and how the validity of V&V results shall be protected from accidental or deliberate compromise. [3.7.4]

Well-defined procedures are necessary for effective control of the software products (inputs for V&V evaluation) and the results of the software V&V efforts (V&V outputs). These procedures, especially those used in software quality assurance and configuration and data management, are similar to those used in software development. In many cases, the same procedures can be employed here. If so, those procedures may be incorporated by reference. The degree to which they are to be implemented should be identified, along with any necessary amendments. New procedures that are planned or under development should also be identified.

The set of procedures should describe how software products and results of software V&V shall be configured, protected, and stored. At a minimum, they should describe how SVVP materials shall comply with existing security provisions, and how the validity of V&V results shall be protected from accidental or deliberate compromise.

The Standard lists configuration management as an optional V&V task. Ideally, effective software configuration management procedures for development will have already been implemented. If not, their implementation as part of the V&V is recommended. Configuration management of the V&V products is also important if the validity of V&V results is to be protected. Additional guidance on effective control procedures can be found in IEEE Std 828-1990 and IEEE Std 1042-1987.

In order to ensure valid results, control procedures should include methods for identifying software items (including documentation), controlling and implementing changes, and recording and reporting change implementation status. These configuration management processes are needed if the V&V results are to be correctly linked with the products that were evaluated and the V&V tasks that were performed. Reviews and audits may be useful in assuring that these processes are effective. Code and media control are also needed, and may be part of configuration management. Finally, records collection, maintenance and retention should be addressed.

The first step in control is the identification of the items to be controlled. The various V&V inputs and outputs should be listed, along with provisions for unique identification of each (e.g., name, version/revision, date). Annex C.3 of this guide discusses V&V inputs and outputs. As is pointed out in annex C, V&V inputs include the V&V plan and procedures as well as the development products that are to be verified and validated.

Items to control include the following:

a) Computer files as well as documents
b) Applications software, operating system software, libraries, test drivers, test data, etc.
c) Any automated tools used as resources in obtaining the V&V results, so that those results will be reproducible if necessary

Identification of the V&V inputs and outputs includes the form (e.g., paper, 5.25 in floppy disk) and format (e.g., pure ASCII text, IEEE Std 1012-1986). The state (e.g., draft or official releases only, TBDs) should be considered. Criteria for release of input documents to V&V, and for release of V&V results, should be defined.

The procedures should specify the means of controlling and implementing changes to the computer files and documentation. Effective change control is needed to avoid attempting V&V on a moving target. Considerations include the following:

— Change review authority and responsibility
— Preparation, routing and approval of change proposals
— Methods of implementing approved change proposals
— Software library control procedures, e.g., access control, read/write protection, change history, archival

See also the discussion of change proposals in 5.5.1.2 of this guide.

Status accounting procedures are needed, for recording and reporting change implementation status. Information on the current status of each item should be known, and periodic reports may be issued. Examples of information that may be collected are

— Latest version/revision of each computer file and document
— Status of change proposals related to each item (e.g., pending vs. issued vs. approved)
— Status of anomaly reports for each item (e.g., open vs. resolved)
— V&V tasks completed for each item
— Review/audit history of each item
— Approved deviations applicable to latest SVVP version/revision

Reviews and audits should be considered, to monitor the effectiveness of the control procedures. Organizational roles in such reviews and audits should be defined. The points in the life cycle at which the reviews and audits will occur, and the items to be covered in each, should be specified. Methods for identifying and resolving problems should be stated.

Code control can be interpreted as the ways and means necessary to protect or ensure the validity of a completed code. Code control may be performed as part of the configuration management process. The procedures described above can be used to cover many of the elements of code control such as specification of code to be controlled, code identification, use of a software library, and code change control. In order to comply with existing security provisions, procedures may also be needed to describe the physical location of the software under control; the requirements for obtaining copies; and the location, maintenance, and use of backup copies.

Media control is concerned with protection of the physical media on which computer files are stored, including storage and retrieval (including off-site storage), access restrictions, and environmental control to avoid physical degradation. Here, as in code control, existing security provisions should be addressed.

Provisions for records collection, maintenance, and retention should include identification of records to be retained, specification of the manner in which they will be maintained (e.g, hard copy, microfiche), and specification of the length of retention for each type of record addressed. Organizational responsibilities in this area include originating, collecting, maintaining, sorting, and protecting records. Authority for accessing, changing, purging, or destroying records should also be considered.

5.7.5 Standards, practices, and conventions

> *(Section 7.5 of the Plan.) This section shall identify the standards, practices, and conventions that govern the performance of V&V tasks, including internal organizational standards, practices, and policies. [3.7.5]*

This section of the plan identifies the standards, practices, and conventions that govern the actual performance of the V&V tasks. New and existing standards, practices, and conventions are to be identified here. These may include documents that direct the V&V tasks and documents against which software products are to be evaluated during those tasks. Both internal and external documents should be considered.

The following are general examples of standards, practices, and conventions that may be applicable.

— Standards for software requirements, design, implementation, test, and documentation, against which the software is to be evaluated
— Detailed procedures for V&V tasks
— Detailed checklists for use in software evaluation
— Standards for reviews and audits
— Quality assurance requirements for the V&V program
— Any standards, practices, and conventions required by the contract

Depending on the project environment, specific standards, practices, and conventions like the following may be required

— Industry standards
— Professional standards
— Government standards
— Regulatory standards

Annex A

(informative)

Extracts from a sample Software Verification and Validation Plan

A.1 Overview

Extracts from a sample software verification and validation plan are provided to represent an SVVP that complies with IEEE Std 1012-1986. The material is intended to show an example of the development of an SVVP. The example is based on a planned software development effort for an elevator system from a fictitious manufacturer, the U&D Elevator Company. The elevator example represents a software application that is meaningful and is easily understood by a wide audience.

The purpose of providing this material is to illustrate

— Realistic planning decisions
— Details and specific requirements in a plan
— Relationship of tasks

The rest of this annex is divided into two sections—elevator reference material and extracts from a sample SVVP.

The extracts from the sample SVVP are presented in a format that explicitly identifies the following seven topics the Standard requires (see also annex C):

a) *Task*. Identify task and how it applies.
b) *Method*. Describe specific method and procedure for each task.
c) *Inputs/Outputs*. Identify inputs and outputs for each task.
d) *Schedule*. Identify schedule for each task.
e) *Resources*. Identify resources for performance of task.
f) *Risks*. Identify risks associated with task.
g) *Roles*. Identify organization or individual responsible for task.

It is important to understand the timing of writing this SVVP in context with the timing of a software development project supporting the elevator example.

The first set of extracts for the sample plan were developed as if the SVVP was being written at the end of the concept phase. These extracts address the V&V effort as a whole as well as specific tasks for the requirements and design phases.

The extracts from 3.5.5 of the SVVP were written as if the plans were being updated and expanded at the end of the design phase. Similarly, the extracts from 3.5.6 of the SVVP represent updates made at the end of the implementation phase.

NOTE—The extracts illustrate a few of the components of an SVVP but are not complete, nor do they represent the only approach to the development of an SVVP.

A.2 Elevator system reference material

OUTLINE OF CONCEPT
for the
U&D Elevator Control System—UDC.100

Project No. 466-A
Prepared for the:
Department of Elevators
By the:
Elevator Design Group

I. SCOPE

A. Identification

The concept document establishes the computer software and hardware configuration items of the U&D Elevator Control System, UDC.100.

B. Purpose

The U&D Elevator Control System, UDC.100, is a distributed system of computer hardware and software, operating as a local area network (LAN) for the control and scheduling of a modular, large-scale elevator installation. It handles all operator control, user requests, elevator scheduling, event logging, monitoring, reporting, maintenance, and security. The Elevator Control System will control up to 16 banks of elevators, with each bank having up to 16 elevators. Up to 256 vertical levels can be served by the UDC.100 elevator system.

II. SOFTWARE DESCRIPTION

Four major subsystems will comprise the Elevator Control System:

a) Master control
b) Bank control
c) Motion control
d) Elevator database

There is a control computer, UDC.HS1100, for each elevator in the system, providing backup and redundancy. The bank control subsystem software, UDC.S1101, and motion control software, UDC.S1102, are both resident in each elevators' control computer. Only one bank controller, though, will be active at any time for each bank. The other idle bank controllers are in standby mode, and can each assume control of their respective elevator banks upon command from master control. The master control computer will be duplexed and the database will be mirrored for backup. These elements and their data relationships are illustrated in figures A.1 and A.2.

A. Elevator master control subsystem—UDC.S110

The master control subsystem will initiate and terminate all operations of the elevator system, and will allow the system operator to direct any elevator to a specific level and to set its mode to either *in-service* or *out-of-service*. The master control subsystem will set the scheduling of each bank controller independently, with either predetermined (e.g., morning arrivals, evening departures, etc.) or custom scheduling scripts (e.g., skip floors 6–10). The master control subsystem will report status information for the operator about all elevators, and log all activity into the elevator database. The master control subsystem will produce analysis reports of elevator activity on request. In addition, the master control subsystem will assist maintenance by controlling diagnostic services for any bank controller and/or motion controller in the system.

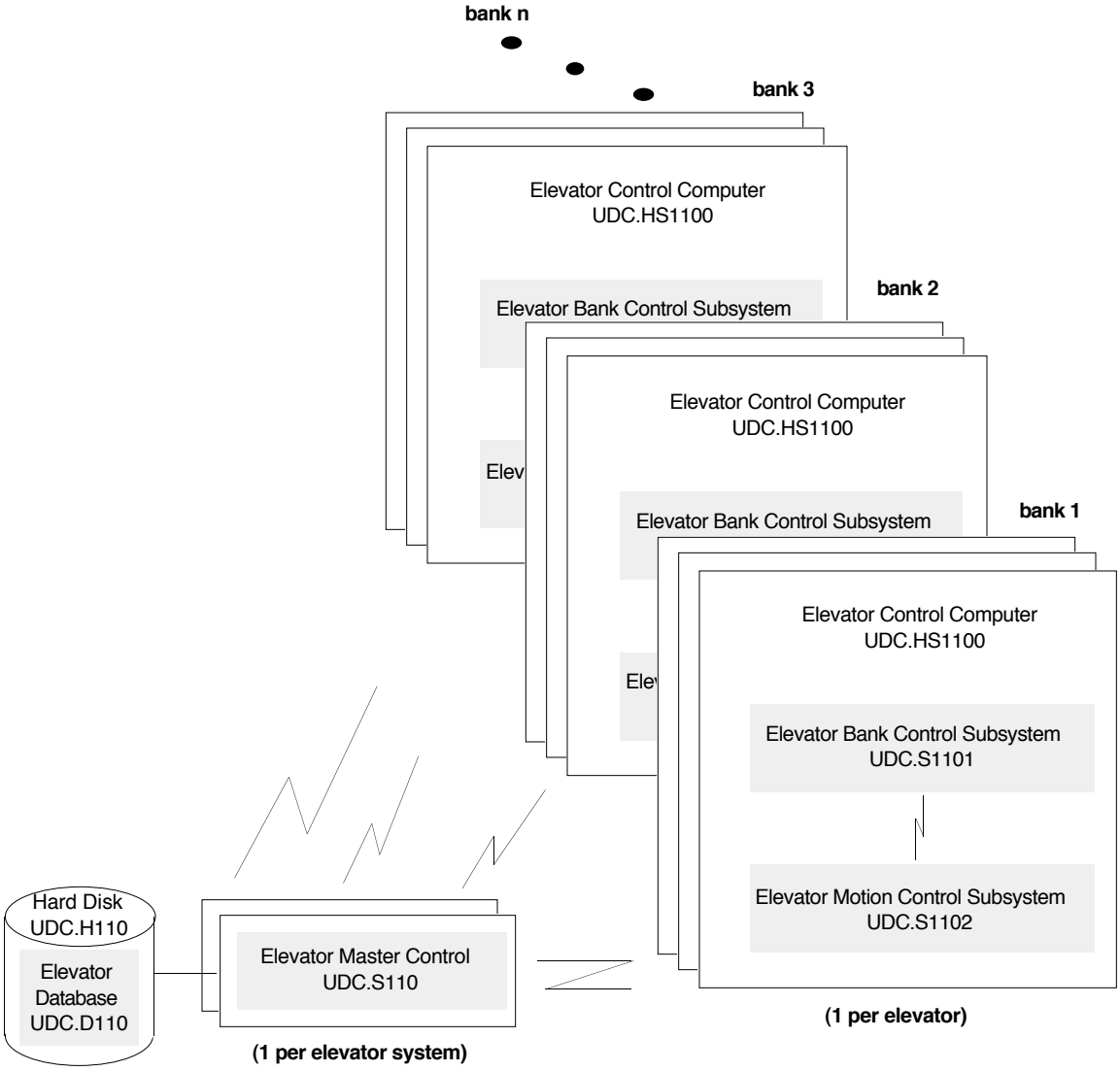

Figure A.1—U&D elevator control system UDC.100

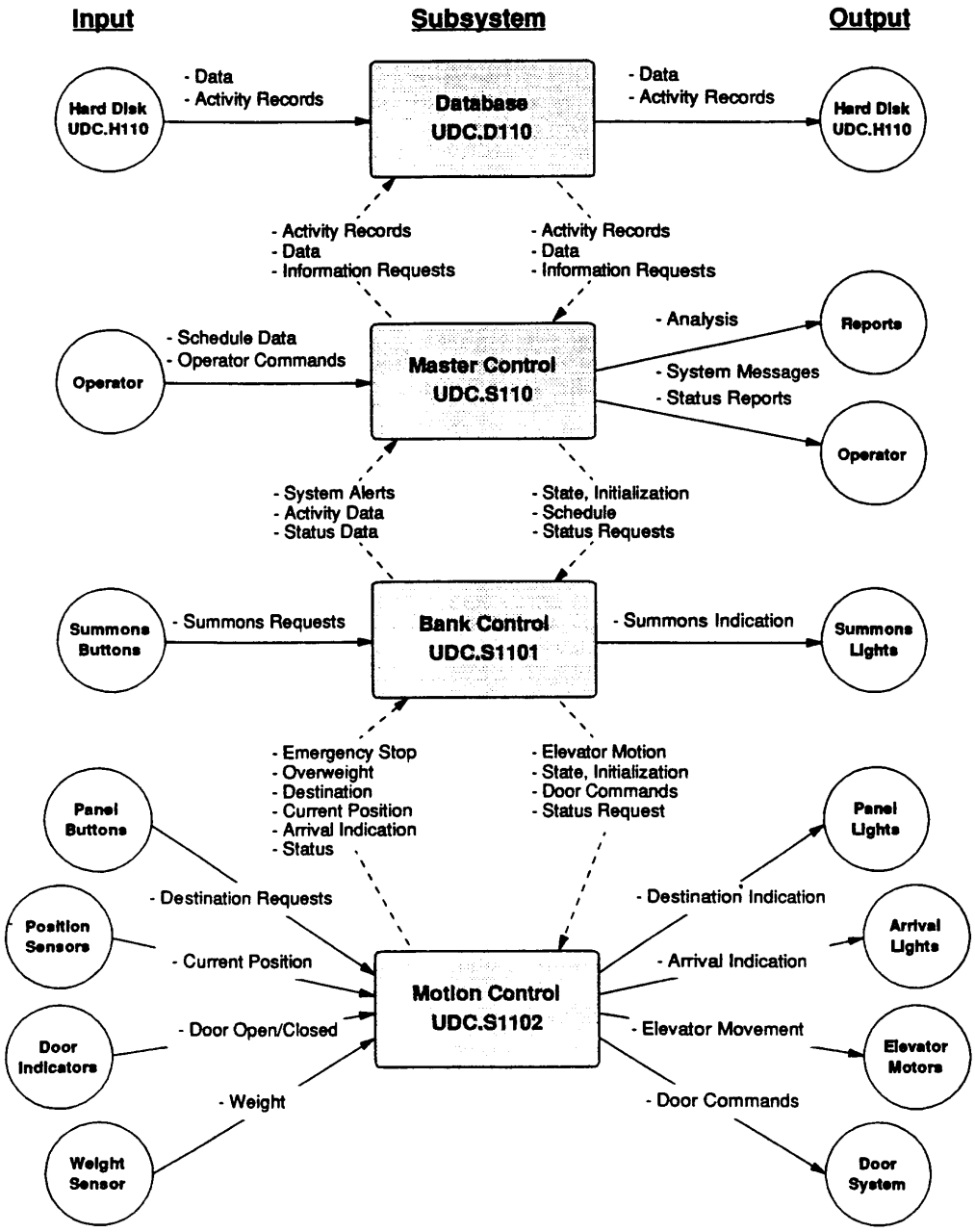

Figure A.2—Elevator control system UDC.100 data flow

The master control subsystem will direct each elevator bank controller, UDC.S1101, and will issue scheduling and operating commands, will request real-time operating status, and will receive and log all elevator command and operating history into the elevator system database, UDC.D110.

The master control subsystem will designate which will be the active bank control subsystem for each bank. This designation can be either directed by the operator, or occur in response to a time-out condition between the master control subsystem and the current bank control subsystem. The operator will be notified if a change of bank controller occurs in response to a time-out condition. In all cases, the change will be logged.

The master control subsystem will include the necessary operator support for entering and updating all elevator system database files. This will include the preparation of scheduling scripts, status reports, and analytical and historical reports.

B. Elevator bank control subsystem—UDC.S1101

Each elevator will have a microcomputer, UDC.HS1100, that will house both the bank control subsystem program, UDC.S1101, and the elevator motion control subsystem program, UDC.S1102. The bank control subsystem of only one of the elevators in a bank will be active at a time and will control the entire bank of elevators, comprising all of their individual motion controllers.

Each in-service elevator control unit will be able to take over as the bank controller, that is, activate its dormant bank control subsystem program, UDC.S1101, if the original bank controller does not respond to a transmission with the master controller in a specified period of time. The master control subsystem will assign the new bank controller from one of the dormant bank controllers within that bank of elevators.

The bank control subsystem will schedule each elevator in the bank according to the scheduling script provided to it by the master control subsystem. It keeps current information on the status of all elevators in the bank and determines which car will respond to each summons.

The bank controller will do the following:

— Schedule passenger requests (from interior panels)
— Schedule responses to summonses from floors
— Command elevator motion
— Set floor summons lights
— Monitor position (parked/in-transit/arrived)
— Monitor emergency stop condition
— Log all elevator activity to the master controller

The bank control subsystem will respond to all master controller commands, such as:

— Activate/deactivate
— Provide status information
— Update scheduling scripts
— Initiate maintenance procedures

In addition, the bank control subsystem will detect, notify the master controller, and remove from service any failing motion controller. In the event of the failure of a motion controller, the associated elevator will be stopped at the next floor and placed out of service.

C. Elevator motion control subsystem—UDC.S1102

There is one motion control subsystem for each elevator car, resident in the elevator control unit, UDC.HS1100, which is the same microcomputer used by the bank control program. The motion controller receives commands for its elevator car's operation from passengers, from its bank controller, and from sensors for the elevator position, door status, and weight. A motion controller will be active concurrently for each in-service elevator, and will do the following:

— Load elevator operations parameter tables

— Control lift motor operation
— Control door operation
— Respond to weight sensor
— Respond to emergency stop switch
— Set interior request (panel) lights
— Set arrival indicators (audible, visual)
— Set progress lights

If a motion controller fails, the elevator car will revert to a fail-safe mechanical operating mode and stop at the next level in its direction. The car can be further directed by the bank controller, if one is serviceable, to perform other operations.

D. Elevator system database—UDC.D110

The database will maintain files for elevator operations, scheduling, status, history, maintenance, and security.

1. Elevator operations files

The operating parameters of each elevator, such as its velocity and acceleration characteristics, door open/close specifications, weight limits, and manual control modes are individually determined. The operating parameters are provided to each elevator's motion controller by the master controller (through the bank controller) during an elevator initialization procedure.

2. Scheduling Files

These files contain the various scheduling scripts that are supplied by the master controller to the bank controller. Scheduling files are generally used by all of the elevators of a bank, although they can be targeted to individual elevators.

3. Status Files

The current elevator system operating environment is recorded, such as assigned schedules, in-service/out-of-service elevators, operators, and timed events (e.g., schedule changes).

4. History Files

They record a history of all elevator movements, schedule changes, and operating modes by date, time, bank, and elevator for a period of up to one year. These files are continuously recorded, and are purged by command. Analysis routines will produce operational history reports for determining scheduling and use patterns.

5. Maintenance Files

A record of the maintenance status and history of every elevator and system component will be recorded.

6. Security Files

A file of passwords and permissions will be maintained for allowed operating personnel.

III. HARDWARE DESCRIPTION

A. Elevator system master control center

A duplex computer, UDC.H4000, with touch-screen color displays, keyboards, audible alarms, two printers, removable disk media, backup tape drives, token-ring LAN telecommunications, and 12–16 MB of memory (see also II, D, Elevator system database—UDC.D110).

B. Elevator control computer

A microcomputer, UDC.HS1100, with 8 MB of memory, and a token-ring LAN port. Removable storage is provided for maintenance. Analog to digital ports will provide control signals for destination requests and indicators, elevator movement, summons requests and indicators, position sensors, door open/close signals, weight sensors, emergency call/stop signals, and manual mode key.

Keyboard, printer, and display may be connected for maintenance, but are not normally present. There is an elevator control computer for each elevator, and both the bank control subsystem and motion control subsystem will be resident.

C. Elevator hard disk

A mirrored set of hard disks, each with 1 GB (1 000 000 000 B) of storage, and 12 ms access time, UDC.H110. This disk subsystem will be accessible from each of the duplexed master control computers, UDC.H4000. Backup will be via streaming tape, and be performed each 24 h. All operating software, utilities, and databases will be stored on-line on this disk.

IV. DEVELOPMENT ENVIRONMENT

A development environment is outlined in which the V&V plan examples are defined.

The software is being produced by a fictitious manufacturer, the U&D Elevator Company. Software development is governed by the manufacturer's standards, which are tailored versions of the IEEE software standards. Tailoring of these standards has incorporated the requirements of the different regulatory agencies controlling the licensing and certification of elevator systems. (Note that fictitious standards and references will be cited by the examples as they are needed.)

The software development methodology can be characterized as being an iterative waterfall process employing real-time structured design methods, using data flow diagrams, data relationship diagrams, module hierarchy diagrams (structure charts), and state transition diagrams. There will be two levels of design, high level and detail. It is assumed that the resulting program code will be written in C language, and will be approximately 250 000 lines of source code. A significant part of this code will be available from reused modules, or commercial sources (e.g., door control, hoist motor control). The schedule calls for software development to be completed in 12 months.

Software testing will be divided into these four categories:

a) *Software component test*. Defined and performed by software development.
b) *Software integration test*. Defined and performed by systems engineering, with quality assurance review.
c) *System test*. Defined by quality assurance, executed and reviewed by systems engineering.
d) *Acceptance test*. Defined and executed by quality assurance, and reviewed by marketing.

Quality assurance has the responsibility defining system testing of hardware and software, and for writing the SVVP. Portions of the plan may be generated by software development. Final approval for the SVVP will reside with project management. There is assumed to be monthly project management meetings, and that marketing will attend for concept and user representation.

The plan will be administered and executed by quality assurance. Specific tasks within the plan may also be executed by other organizations. Hardware development will participate in the execution of the SVVP for interfaces with the hardware components during systems and acceptance testing. Similarly, quality assurance will develop a hardware verification and validation plan (HVVP), and hardware development and systems engineering will assist with its execution.

A.3 Extracts from a sample SVVP

A.3.1 Example for 5.1 Purpose

Purpose

This SVVP specifies the tasks to be performed for the U&D Elevator Control System UDC.100, project 466A-SVVP 1.0. This SVVP is written in accordance with IEEE Std 1012-1986.

Scope

The scope of this SVVP is to verify and validate all UDC.100 developed software and to validate all packaged and embedded software components for conformance to requirements for safety factors, response time, expandability, and external interfaces. Until a customer is under contract, the acceptance testing activities will not be included.

Objectives

The objectives of this SVVP are to verify and validate all developed software and interfaces for conformance with UDC.100 requirements for response time, expandability, and safety factors.

Applicability

This SVVP applies to the products of the requirements, design, implementation, and test phases of the UDC.100 software development cycle.

Waivers

This SVVP deviates from IEEE Std 1012-1986 in that it will not be applied to the following phases:

a) Concept phase
b) Installation and checkout phase
c) Operation and maintenance phase

This SVVP has been developed during the concept phase and applies to subsequent phases. The V&V planning for the installation and checkout phase, and the operation and maintenance phase will be deferred until a customer installation has been identified for UDC.100.

A.3.2 Example for 5.4.1 Organization

See figure A.3.

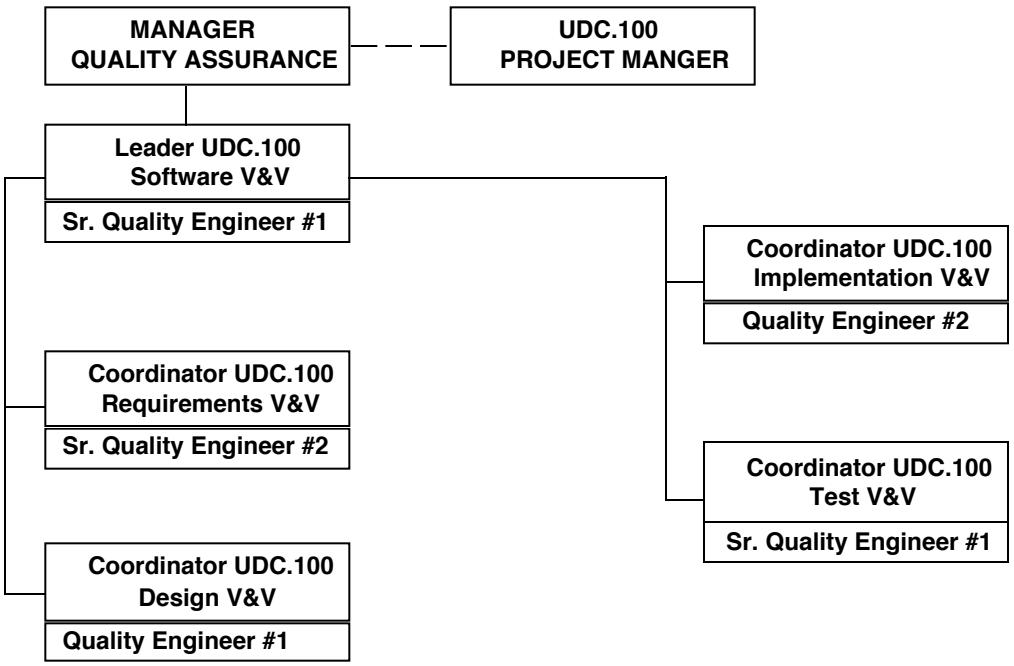

Figure A.3—Software V&V organization—UDC.100 project organization

A.3.3 Example for 5.4.4 Responsibilities

See table A.1.

Table A.1—UDC.100 software V&V project responsibilities

Position	Responsibilities
Sr. Quality Engineer #1	Coordinator—Software V&V Coordinator—Test V&V Management Acceptance Test Plan Generation Acceptance Test Design Generation
Sr. Quality Engineer #2	Coordinator—Requirements V&V Criticality Analysis Requirements Traceability Analysis
Quality Engineer #1	Coordinator—Design V&V Design Traceability Analysis Acceptance Test Procedure Generation
Quality Engineer #2	Coordinator—Implementation V&V Requirements Evaluation System Test Plan Generation Design Evaluation System Test Design Generation Source Code Traceability Analysis Acceptance Test Case Generation Acceptance Test Execution
Quality Engineer #3	Source Code Evaluation System Test Case Generation System Test Procedure Generation
Associate Quality Engineer #1	Requirements Interface Analysis Design Interface Analysis Source Code Interface Analysis Source Code Documentation Evaluation
Software Engineer #1	Source Code Evaluation Component Test Plan Generation Component Test Design Generation Integration Test Execution
Software Engineer #2	Component Test Case Generation Component Test Procedure Generation System Test Execution
Software Engineer #3	Component Test Execution
Sr. Systems Engineer #1	Integration Test Plan Generation Test Design Generation Integration Test Design Generation
Systems Engineer #1	Integration Test Case Generation Integration Test Procedure Generation Integration Test Execution

A.3.4 Example for 5.5.1.1 SVVP generation

a) Task: Update the SVVP at the completion of the Design Phase.

b) Method: Review the output products of Requirements and Design phases. Assess for changes and impacts. Revise plan accordingly.

c) Inputs/Outputs:

1) Inputs

i) Current SVVP

ii) Design phase summary report

iii) Component test plans and test designs

iv) Integration test plans and test designs

v) System test plans and test designs

vi) Current development plans

2) Output: Revised SVVP

d) Schedule: Initiate within two weeks of the publication of the Design Phase V&V Summary Report and complete in five days.

e) Resources: Senior quality engineer for four staff days.

f) Risks: There are no identified risks.

g) Roles: Senior quality engineer #1 will update the SVVP. The manager and quality assurance will approve.

A.3.5 Example for 5.5.1.3 Management review of V&V

NOTE—The example represents one component of management review. Other activities that must be planned include periodic review, daily management, and evaluation of results.

a) Task: Review Design Phase task reports.

b) Method: Review and evaluate intermediate products for overall assessment of quality and adherence to plan. Identify key findings and deviations from plan. Make specific recommendations as appropriate.

c) Inputs/Outputs:

1) The inputs are all SVV outputs

2) The output is the design phase summary report

d) Schedule: Initiate at the completion of the other design phase V&V activities and complete within two weeks.

e) Resources: Senior quality engineer for one staff day.

f) Risks: There are no identified risks.

g) Roles: Senior quality engineer # 1 will prepare the report.

A.3.6 Example for 5.5.3.1 Software requirements traceability analysis

a) Task: Trace the functions for the Elevator Master Control Subsystem that are specified in the concept for U&D UDC.100 to the requirements for component UDC.HS1100, and trace backward to the concept for UDC.100. Identify the response time, safety factors, and expandability requirements.

b) Method: Perform a manual trace, both backward and forward, to determine functional and data relationships, and any functions and data not defined in the UDC.100 concept or requirements for

UDC.HS1100. Indicate where either functions or data cannot be traced from the concept to the requirements, or back from the requirements to the concept for UDC.HS1100.

c) Inputs/Outputs:
   1) Inputs
      i) UDC.100 concept document
      ii) UDC.HS1100 requirements
   2) Outputs
      i) Trace table of the requirements for UDC.HS1100
      ii) Traceability discrepancy report for UDC.HS1100

d) Schedule: Initiate at the completion of the requirements for UDC.HS1100, planned for three months after project initiation and complete within three weeks.

e) Resources: The traceability analysis will require ten staff-days to perform. Reproduction facilities and clerical support are required for three days.

f) Risks: The traceability analysis must be completed prior to the initiation of the design of UDC.HS1100.

g) Roles: The traceability analysis will be performed by senior quality engineer # 2.

A.3.7 Example for 5.5.3.2 Software requirements evaluation

a) Task: Evaluate that the specified response time requirements for service of an elevator bank do not exceed the capabilities of the specified equipment parameters.

b) Method: Use the Lipoff-Craig queuing model, tailored for the UDC development workstation.

c) Inputs/Outputs:
   1) Inputs
      i) Distribution and maximum service wait times
      ii) Specified by the requirements for UDC.HS1100
      iii) Configuration of the UDC.100 elevator bank (operational parameters are included within the model)
      iv) Distribution of the expected service requests
   2) Output: Expected service distribution report produced by the model

d) Schedule: Initiate at the completion of the requirements for UDC.HS1100, planned for three months after project initiation, and complete within four weeks.

e) Resources: The Lipoff-Craig queuing model software is required, and a quality engineer for one staff week has been allocated for this task. Access to an engineering workstation with the standard network capabilities will be required for 40 hours.

f) Risks: The Lipoff-Craig model has not been validated for the Ockerman hoist motor, UDC.H2620 door retractor, and requests greater than 25 per minute.

g) Roles: Quality engineer # 2, trained in the use of the Lipoff-Craig model.

A.3.8 Example for 5.5.3.3 Software requirements interface analysis

a) Task: Analyze the interfaces between the Bank Control, UDC.S1101, and Motion Control, UDC.S1102, subsystems.

b) Method: The interfaces between UDC.S1101 and UDC.S1102 will be verified by an interface inspection.

c) Inputs/Outputs:

1) Inputs
   i) UDC.100 data dictionary
   ii) Data flow diagrams for the UDC.HS1100 Control Computer
   iii) The interface between UDC.S1101 and UDC.S1102
   iv) Elevator motion timing parameters
2) Outputs
   i) Series of inspection defect lists
   ii) Summary reports
   iii) Inspection report detailing the inspection findings

d) Schedule: Initiate at completion of the requirements for UDC.HS1100, planned for three months after project initiation, and complete within two weeks.

e) Resources: Assigned moderator (16 staff hours), inspectors (one other beside the author for 5 hours each), and inspection meeting room and clerk (4 hours) are also required.

f) Risks: There are no identified risks, except that the interface inspection and the resolution of all identified anomalies must occur prior to design of the two components.

g) Roles: Associate quality engineer # 1, trained in the inspection technique, as moderator, with the participation of the author of the UDC.HS1100 requirements, and an experienced systems engineer as one of the inspectors.

A.3.9 Example for 5.5.3.4 System test plan generation

a) Task: Generate the system test plan for the Elevator Control System UDC.100.

b) Method: Cast the operations for the Elevator Control System UDC.100 as a series of matrices of operations, one matrix for each operational state. These matrices will be used to define test conditions for UDC.100. Test will focus on elevator response and safety features.

c) Inputs/Outputs:
   1) Inputs
      i) Concept document
      ii) SRS
      iii) Trace analysis (from the software requirements traceability analysis task in A.3.6)
      iv) Interface data flow diagrams for UDC.100
   2) Output: Elevator Control System UDC.100 system test plan

d) Schedule: Initiate at the completion of the SRS for UDC.100 and complete before initiating integration test design.

e) Resources: A quality engineer for two months.

f) Risks: There are no identified risks.

g) Roles: Quality engineer # 2, experienced in system test planning for U&D software control systems.

A.3.10 Example for 5.5.3.4 Acceptance test plan generation

a) Task: Generate a draft acceptance test plan for the Elevator Control System UDC.100 to be installed in a large office building for a customer not yet specified.

b) Method: Cast the operations for the Elevator Control System UDC.100 as a series of matrices of boundary values, one matrix for each operational state. These matrices will be used to define acceptance test conditions for UDC.100. Develop an operation suite reflecting expected customer usage and special conditions. Test will focus on elevator response and safety features.

c) Inputs/Outputs:
   1) Inputs
      i) Concept document
      ii) SRS
      iii) Trace for the SRS
      iv) Interface analysis
      v) Expected customer usage patterns and conditions
   2) Outputs
      i) Acceptance test plan matrices
      ii) Plans for providing the input conditions for the UDC.100 customer installation

d) Schedule: Integrate at completion of the SRS for UDC.100 and complete before initiating acceptance test design.

e) Resources: A senior quality engineer for two months.

f) Risks: The Elevator Control System UDC.100 is new, without an identified customer. Specific customer demands may require changes to the draft plan.

g) Roles: Senior quality engineer # 1 trained in acceptance test planning.

A.3.11 Example for 5.5.4.1 Software design traceability analysis

a) Task: Trace requirements in the SRS to each of the high level design components of the SDD for the Elevator Master Control, UDC.S110; Bank Control, UDC.S1101; Motion Control, UDC.S1102; and Elevator Database, UDC.D110. The reverse trace, from each high level design component of the SDD back to the SRS, shall also be performed. Identify the response time, safety factors, and expandability requirements.

b) Method: The U&D Trace Tool 1 analytic tracing program shall associate each high level design component with the appropriate requirements to identify any possible inconsistencies between requirements and design. Anomalies will be identified.

c) Inputs/Outputs:
   1) Inputs
      i) SRS for the Elevator Master Control, UDC.S110; Bank Control, UDC.S1101; Motion Control, UDC.S1102; and Elevator Database, UDC.D110
      ii) Draft software high level design description
      iii) UDC.S110 requirements trace table (from the software requirements traceability analysis task in A.3.6)

      [Both the requirements and high level design shall be in standard ASCII format, written respectively to requirements standard REQ-1.2 and high level design standard HLD-1.4.]
   2) Outputs
      i) Design traceability table
      ii) Anomalies list produced by U&D Trace Tool 1

d) Schedule: Initiate at the release of the draft high level design descriptions and complete within five days.

e) Resources: The U&D Trace Tool 1 support tool is required, and 16 staff hours for a quality engineer have been allocated for this task. Access to an engineering workstation with the standard network capabilities will be required for 8 hours.

f) Risks: U&D Trace Tool 1 is a newly released tool offering improved capabilities, but has not yet been qualified on an operational project. Should unforeseen problems arise, the proven U&D TRACER tool can be utilized, but with more limited capabilities.

g) Roles: This task will be performed by quality engineer # 1.

A.3.12 Example for 5.5.4.2 Software design evaluation

a) Task: Evaluate that the response time for service for each elevator bank will not exceed the specified limit.

b) Method: Use the UDC Elevator System Simulator. Verify that the scheduling algorithms provide designed response times.

c) Inputs/Outputs:

1) Inputs

i) Hardware response times of the UDC.100 components (motor acceleration and deceleration characteristics, car door open and close times, etc.)

ii) Expected request patterns for each scheduling mode (maximum up, maximum down, normal, etc.)

iii) Scheduling algorithms

iv) State transition diagrams

2) Outputs

i) Response time report

ii) Service time distributions

iii) Anomalies report

d) Schedule: Initiate at the receipt of high level SDD and complete within six weeks.

e) Resources: The UDC Elevator System Simulator is required, and one staff month for a quality engineer (with previous UDC Elevator System Simulator experience) has been allocated for this task. Access to an engineering workstation with the standard network capabilities will be required for 160 hours.

f) Risks: The updated parameters for the UDC Elevator System Simulator shall have been calibrated and certified.

g) Roles: Quality engineer # 2 familiar with the use of the UDC Elevator System Simulator will perform the analysis. Results will be provided to the UDC.100 project manager. Any modifications to the scheduling algorithms will be re-evaluated.

A.3.13 Example for 5.5.4.3 Software design interface analysis

a) Task: Analyze the interfaces between the high level design of the Bank Control Subsystem UDC.S1101, and the Motion Control Subsystem UDC.S1102.

b) Method: The specifications for the interface formats, parameters, queuing, tasking, timing, and expected responses will be verified by design interface inspections.

c) Inputs/Outputs:

1) Inputs (for both subsystems UDC.S1101 and UDC.S1102)

i) Design traceability table (from the software design traceability analysis in A.3.11)

ii) Subsystem data flow diagrams

iii) Data entity charts

iv) Subsystem message lists

v) SDD interface standards and conventions

b) Outputs

i) Inspection defect lists

ii) Inspection database reports

iii) Inspection management reports

iv) Anomaly reports

d) Schedule: Initiate at completion of both UDC.S1101 and UDC.S1102 high level design interface specifications and complete within two weeks.

e) Resources: Each inspection will consist of a team of four persons, including the developer of the specification. It is estimated that there will be two high level design interface inspections of two hours each. Thirty staff hours are estimated for the inspections, expected rework, and inspection data analysis and reporting.

f) Risks: There are no identified risks.

g) Roles: Associate quality engineer # 2 as moderator for each interface inspection. The moderator will certify inspection of the interface specifications for UDC.S1101 and UDC.S1102. The UDC.100 project manager and a lead software engineer will receive the inspection management reports and the inspection database summaries.

A.3.14 Example for 5.5.4.4 Component test plan generation

a) Task: Generate the component test plan for the Motion Control Subsystem UDC.S1102 components.

NOTE—There will be a series of four component test plans for each of the subsystems.

b) Method: Develop a plan for testing each component of the Motion Control Subsystem UDC.S1102, using IEEE Std 1008-1987 and document according to IEEE Std 829-1983. Specify use of U&D Code Coverage analyzer to achieve 100% statement coverage for each component. The test will be conducted with hardware and software simulators in the UDC Elevator Testing Lab. The test results will be compared to the expected results for each test case.

c) Inputs/Outputs:

1) Inputs

i) SRS

ii) SDD

iii) Design traceability table (from the software design traceability analysis in A.3.11)

iv) Interface design specifications

v) Hardware and software simulators user's guide

vi) Draft user documentation

vii) UDC code coverage analyzer specifications

2) Output: UDC.S1102 component test plan

d) Schedule: Initiate at completion of interface analysis and complete prior to final design approval.

e) Resources: The component test plan will be prepared by a UDC.100 software engineer and is estimated as requiring two staff weeks of effort.

f) Risks: There are no identified risks.

g) Roles: Software engineer # 1 will prepare the component test plan. A lead software engineer will approve it.

A.3.15 Example for 5.5.4.4 Integration test plan generation

a) Task: Generate the Elevator Control System UDC.100 Integration Test Plan.

b) Method: Develop an integration test plan for UDC.100, which will comprise integrating test plans for the four individual subsystem's integration test plans as well as a integration test plan for the total UDC.100 system.

c) Inputs/Outputs:

1) Inputs

i) SDD
ii) Design traceability table (from the software design traceability analysis in A.3.11)
iii) Interface design specifications
iv) Interface analysis report
v) Criticality analysis report
vi) Draft operating documentation
vii) Draft maintenance documentation

2) Output: UDC.100 integration test plan

d) Schedule: Initiate one month following the start of the design and complete within four weeks.

e) Resources: Integration test planning is estimated as requiring three staff weeks of effort.

f) Risks: There are no identified risks.

g) Roles: Systems engineer # 1 will prepare the integration test plan. The manager of quality assurance will approve the integration test plan.

A.3.16 Example for 5.5.4.5 Test design generation—Component testing

a) Task: Design component tests for the Motion Control Subsystem UDC.S1102.

NOTE—It is expected that the component test plan will specify that the generation of test designs be done for each subsystem.

b) Method: Develop test designs in accordance with the Component Test Plans for the Motion Control System. In addition, design the necessary interfaces and support for the use of the UDC code coverage analyzer.

c) Inputs/Outputs:

1) The inputs are:

i) Component test plans
ii) SDD
iii) Interface design specifications
iv) Draft operating documentation
v) Hardware and software simulators setup procedures
vi) UDC code coverage analyzer operations manual

2) Output: Test designs for the Motion Control Subsystem UDC.S1102

d) Schedule: Initiate two weeks after the component test plan is started, and complete within four weeks.

e) Resources: Three staff weeks are estimated for a software engineer.

f) Risks: There are no identified risks.

g) Roles: Software engineer # 1 will design the Motion Control Subsystem tests.

A.3.17 Example for 5.5.4.5 Test design generation—Integration testing

a) Task: Design integration tests for the four subsystems and integration of the four subsystems into the UDC.100.

b) Method: Develop the test designs for the four subsystems and integration of the four subsystems into the UDC.100. Design for the recording of the test results.

c) Inputs/Outputs:

1) Inputs

i) UDC.100 integration test plan
ii) SDD
iii) Interface specifications

2) Output: Test designs for the four subsystems and integration of the four subsystems into the UDC.100.

d) Schedule: Initiate two weeks after the integration test plan is started, and complete within twelve weeks.

e) Resources: Two staff months are estimated for a software engineer.

f) Risks: Integration test cases for distributed architecture such as UDC.100 have not been developed before. Schedule estimates are based on past experience. Earlier integration test activities and initial test case development times will be monitored to validate the schedule estimates.

g) Roles: Senior systems engineer # 1 will design integration tests. The designs must be approved by the manager of quality assurance.

A.3.18 Example for 5.5.4.5 Test design generation—System testing

a) Task: Design system tests for the Elevator Control System UDC.100.

b) Method: Develop test designs in accordance with UDC.100 system test plan (from the system test plan generation task in A.3.9). Test Elevator Tower will be used. Design for the recording of the test results.

c) Inputs/Outputs:

1) Inputs
i) UDC.100 system test plan
ii) SDD
iii) Interface specifications
iv) Draft operating documentation
v) Maintenance documentation
vi) Test elevator tower configurations

2) Output: UDC.100 system test designs

d) Schedule: Initiate two weeks after the system test plan is started and complete within six weeks.

e) Resources: A staff month is estimated for a software engineer for the design of system test for the UDC Elevator Control System.

f) Risks: System tests haven't been designed before for distributed systems such as UDC.100. Estimates for the schedule are tentative. A margin of 100% schedule and resource overrun needs to be considered for this activity. System test design may be critical to the UDC.100 project.

g) Roles: Quality engineer # 2 will design system tests. The design must be approved by the manager of quality assurance.

A.3.19 Example for 5.5.4.5 Test design generation—Acceptance testing

a) Task: To be developed when a customer is identified and when the acceptance test plan generation task in A.3.10 is completed.

b) Method: Pending

c) Inputs/Outputs:

1) The inputs are pending.

2) The outputs are pending.

d) Schedule: Pending

e) Resources: Pending

f) Risks: Pending

g) Roles: Pending

(Reminder—The extracts from 5.5.5 of the SVVP were written as if the plans were being updated and expanded at the end of the design phase.)

A.3.20 Example for 5.5.5.1 Source code traceability analysis

a) Task: Trace each of the design requirements in the SDD to the Bank Control Subsystem UDC.S1101 source code. The reverse trace shall also be performed. Identify the response time, safety factor, and expandability design requirements.

b) Method: The U&D Trace Tool 1 analytic tracing program shall associate the design requirements to the individual section of source code where they are implemented to identify any possible inconsistencies between design and code. Anomalies will be defined and categorized.

c) Inputs/Outputs:

1) Inputs

i) SDD for Bank Control Subsystem UDC.S1101

ii) The source code from configuration controlled libraries

iii) Design Traceability Table for UDC.S1101

2) Outputs

i) Source code traceability table

ii) List of anomalies

d) Schedule: Initiate at the acceptance of components by configuration management and complete within five calendar days of acceptance of last component.

e) Resources: A quality engineer for 10 staff days.

f) Risks: There are no identified risks.

g) Roles: This task will be performed by quality engineer # 2.

A.3.21 Example for 5.5.5.2 Source code evaluation

a) Task: Evaluate the Bank Control Subsystem scheduling algorithm requirements.

b) Method: Use Monte Carlo simulation to evaluate the scheduling algorithm requirements to ensure conformance with real time response requirements.

c) Inputs/Outputs:

1) Inputs

i) Bank Control program specifications

ii) Source code

iii) Scheduling algorithm design specifications

iv) Monte Carlo simulation software

2) Output: Scheduling algorithm performance characteristic

d) Schedule: Initiate at acceptance of components by configuration management and complete within two weeks.

e) Resources: One software engineer experienced in Monte Carlo simulations for three staff days.

f) Risks: The Monte Carlo simulation has not been used for elevator banks in excess of eight elevators. This task will be performed immediately upon release of code to allow time for necessary recalibration.

g) Roles: Software engineer #1 experienced in Monte Carlo simulation will perform this task. Quality engineer # 3 will review the results.

A.3.22 Example for 5.5.5.3 Source code interface analysis

a) Task: Analyze the source code interfaces for the Bank Control Subsystem UDC.S1101 and Motion Control Subsystem UDC.S1102.

b) Method: The source code for the interface formats, parameters, timing, and expected responses will be verified by interface inspections. The analysis will focus on the complete and correct transfer of data across the interfaces.

c) Inputs/Outputs:

1) Inputs

i) Interface design specification for UDC.S1101 and UDC.S1102

ii) The program specification for UDC.S1101 and UDC.S1102

iii) The UDC.S1101 and UDC.S1102 source code

2) Outputs

i) Inspection defect lists

ii) Inspection database reports

iii) Inspection management reports

iv) Anomaly reports

d) Schedule: Initiate at acceptance of code by configuration management and complete within two weeks.

e) Resources: Each inspection will consist of a team of three persons, including the developer of the code. It is estimated that there will be two code inspections of two hours each. Thirty staff hours are estimated for the inspections, expected rework, and inspection data analysis and reporting.

f) Risks: There are no identified risks.

g) Roles: Associate quality engineer # 1 will be moderator for each interface inspection. The lead software engineer will receive a copy of inspection reports.

A.3.23 Example for 5.5.5.4 Source code documentation evaluation

a) Task: Evaluate operating manual documentation to ensure consistency with the UDC.S110 Master Control Subsystem specifications.

b) Method: Use a document walkthrough to ensure the operating documentation to the source code of the UDC.S110 Master Control Subsystem. Evaluation of the procedures for entering and updating all elevator system database files will be emphasized. Consistency of the operating documentation instructions and error messages will be checked.

c) Inputs/Outputs:

1) Inputs

i) UDC.S110 source code

ii) UDC.S110 operating documentation

iii) UDC.S110 specification

2) Output: List of discrepancies

d) Schedule: Initiate at receipt of the UDC.S110 source code and the draft of the operator manual. Prior review of documentation is expected. Complete within two weeks.

e) Resources: The evaluation process will include author of documentation and representatives from quality engineering and systems engineering. Estimated that three walkthrough sessions of four hours each will be scheduled. Fifty staff hours are estimated for the walkthroughs and expected rework.

f) Risks: There are no identified risks.

g) Roles: Associate quality engineer # 1 will be moderator for each walkthrough session.

A.3.24 Example for 5.5.5.5 Test case generation—Component testing

a) Task: Generate test cases for component testing of the Motion Control Subsystem UDC.S1102.

NOTE—It is expected that the component test plan will specify that the generation of test cases be done for each subsystem.

b) Method: Develop test case specifications to support the test designs previously defined for the Motion Control Subsystem UDC.S1102. Cases will state actual input and expected results of each test to be run.

c) Inputs/Outputs:

1) Inputs

i) Component test plan

ii) Test design document for Motion Control Subsystem UDC.S1102

iii) Hardware interface specifications

iv) Hardware and hardware simulators documentation

v) Motion Control Subsystem UDC.S1102 source code

vi) UDC code coverage analyzer manual

2) Output: Test case specifications documents

d) Schedule: Initiate after the component test designs are started and complete within four weeks.

e) Resources: Three staff weeks are estimated for a software engineer for developing test cases for UDC.S1102.

f) Risks: There are no identified risks.

g) Roles: Software engineer # 2 will develop the component test cases.

A.3.25 Example for 5.5.5.5 Test case generation—Integration testing

a) Task: Generate test cases for integration testing of the UDC.100.

b) Method: Develop test case specifications for the four subsystems and the integration of the four subsystems into the UDC.100. Cases will state actual input and expected results of each test to be run.

c) Inputs/Outputs:

1) Inputs

i) Integration test plan document

ii) Integration test design document

iii) Hardware and hardware simulators documentation

iv) UDC.100 source code

2) Output: Integration test case specifications

d) Schedule: Initiate after the integration test designs are started and complete within twelve weeks.

e) Resources: Two staff months are estimated for a systems engineer.

f) Risks: Integration test cases for distributed architecture such as UDC.100 have not been developed before. Schedule estimates are based on past experience. Earlier integration test activities and initial test case development times will be monitored to validate the schedule estimates.

g) Roles: Systems engineer # 1 will develop the integration test cases.

A.3.26 Example for 5.5.5.5 Test case generation—System testing

a) Task: Generate test cases for system testing of the Elevator Control System UDC.100.

b) Method: Develop system test case specifications in accordance with UDC.100 test designs from the test design generation—system testing task in A.3.18. Cases will state actual input and expected results of each test to be run.

c) Inputs/Outputs:

   1) Inputs
      i) System test plan document
      ii) System test design document
      iii) UDC.100 subsystem's source code
      iv) Operating manuals
   2) Output: System test case specifications

d) Schedule: Initiate after the system test designs are started and complete within fourteen weeks.

e) Resources: Two staff months are estimated for a quality engineer for developing test cases for UDC.100.

f) Risks: System test cases for distributed architecture such as UDC.100 have not been developed before. Schedule estimates are based on past experience. Earlier system test activities and initial test case generation development times will be monitored to validate the schedule estimates.

g) Roles: Quality engineer # 3 will develop the system test cases.

A.3.27 Example for 5.5.5.5 Test case generation—Acceptance testing

NOTE—This is to be determined when a customer is identified and the test design generation—system testing task in A.3.18 is completed.

a) Task: Generate test cases for acceptance testing of the Elevator Control System UDC.100.

b) Method: Pending

c) Inputs/Outputs:

   1) The inputs are pending.
   2 The outputs are pending.

d) Schedule: Pending

e) Resources: Pending

f) Risks: Pending

g) Roles: Pending

A.3.28 Example for 5.5.5.6 Test procedure generation—Component testing

a) Task: Generate test procedures for component test cases developed for Motion Control Subsystem UDC.S1102.

b) Method: Develop steps and actions required to execute each test case defined for component testing of the Motion Control Subsystem UDC.S1102. Procedures are needed for hardware and software simulators and monitors.

c) Inputs/Outputs:

   1) Inputs
      i) Component test plan designs
      ii) Component test cases for Motion Control Subsystem UDC.S1102

iii) Hardware interface specifications
iv) Hardware set-up procedures
v) Hardware simulators operating procedures
vi) Motion Control Subsystem UDC.S1102 source code
vii) UDC code coverage analyzer manual

2) Output: Component test procedures document

d) Schedule: Initiate one week after component test cases are started. Complete within two weeks after receipt of final component test cases.

e) Resources: One staff week is estimated for a software engineer for developing test procedures for UDC.S1102.

f) Risks: There are no identified risks.

g) Roles: Software engineer # 2 will develop the component test procedures.

A.3.29 Example for 5.5.5.6 Test procedure generation—Integration testing

a) Task: Generate integration test procedures for Elevator Control System UDC.100.

b) Method: Develop steps and actions required to execute each test case defined for the four subsystems and the integration of the four subsystems into the UDC.100.

c) Inputs/Outputs:

1) Inputs
i) Integration test plan designs
ii) Integration test cases specifications
iii) Hardware interface specifications
iv) Hardware set-up procedures
v) Hardware simulators operating procedures
vi) UDC.100 source code

2) Output: Integration test procedures

d) Schedule: Initiate one week after integration test case generation starts. Complete within two weeks after receipt of final integration test cases.

e) Resources: One staff month is estimated for a systems engineer.

f) Risks: Integration test procedures for distributed architecture such as UDC.100 have not been developed before. Schedule estimates are based on past experience. Earlier integration test activities and initial test procedure development times will be monitored to validate the schedule estimates.

g) Roles: Systems engineer # 1 will develop the integration test procedures.

A.3.30 Example for 5.5.5.6 Test procedure generation—System testing

a) Task: Generate test procedures for system testing of the Elevator Control System UDC.100.

b) Method: Develop test procedures required to execute each test case defined for system test of the Elevator Control System UDC.100.

c) Inputs/Outputs:

1) Inputs
i) System test designs
ii) System test case specifications for Elevator Control System UDC.100
iii) Test elevator tower set-up procedures

iv) Hardware timing monitors set-up procedures

v) UDC.100 source code

vi) Operating manuals

2) Output: Test procedures documentation

d) Schedule: Initiate after test case generation starts. Complete within two weeks after receipt of final system test cases.

e) Resources: One staff month is estimated for a quality engineer for developing test procedures for UDC.100.

f) Risks: System test procedures for distributed systems such as UDC.100 have not been developed before. Schedule estimates are based on past experience. Earlier system test activities and initial system test procedure development times will be monitored to validate the schedule estimates.

g) Roles: Quality engineer # 3 will develop the system test procedures. Software quality engineer will approve.

A.3.31 Example for 5.5.5.7 Component test execution

a) Task: Execute test procedures and cases for Motion Control components.

b) Method: Follow the test procedures for Motion Control components test. Procedures define the equipment set-up, initialization procedures, and execution steps. Analyze the results of testing to determine success or failure for each test.

c) Inputs/Outputs:

1) Inputs

i) Component test designs

ii) Component test cases

iii) Component test procedures

iv) User documentation and the source code

2) The outputs are:

i) Component test reports

ii) Anomaly reports

d) Schedule: Initiate after completion of component test procedures and acceptance of code by configuration management. Complete within two weeks.

e) Resources: One staff week is estimated for a software engineer.

f) Risks: There are no identified risks.

g) Roles: Software engineer # 3 will execute the component tests.

(Reminder—The extracts from 5.5.6 of the SVVP represent updates made at the end of the implementation phase.)

A.3.32 Example for 5.5.6.1 Acceptance test procedure generation

a) Task: Generate test procedures for acceptance testing of the Elevator Control System UDC.100.

NOTE—This is to be developed when a customer is identified and the test case generation—acceptance testing task in A.3.27 is completed.

b) Method: Pending

c) Inputs/Outputs:

1) The inputs are pending.

2) The outputs are pending.

d) Schedule: Pending
e) Resources: Pending
f) Risks: Pending
g) Roles: Pending

A.3.33 Example for 5.5.6.2.1 Integration test execution

a) Task: Execute integration test procedures and cases for Elevator Control System UDC.100.
b) Method: Follow the integration test procedures for UDC.100. Procedures define the equipment set-up, initialization procedures, and execution steps. The test execution will be for the four subsystems and integration of the four subsystems into UDC.100. Analyze the results of testing to determine success or failure for each test.
c) Inputs/Outputs:
    1) Inputs
        i) Integration test designs
        ii) Integration test cases
        iii) Integration test procedures
        iv) User documentation
        v) Source and executable code
    2) Outputs
        i) Integration test reports
        ii) Anomaly reports
d) Schedule: Initiate at completion of integration test procedures, component test execution and acceptance of the code by configuration management. Complete within ten weeks.
e) Resources: Two staff months are estimated for a software engineer.
f) Risks: Integration test execution for distributed systems such as UDC.100 has not been performed before. Schedule estimates are based on past experience. Earlier integration test activities and initial integration test execution times will be monitored to validate the schedule estimates.
g) Roles: Software engineer # 1 will execute the integration tests.

A.3.34 Example for 5.5.6.2.2 System test execution

a) Task: Execute system test procedures and cases for Elevation Control System UDC.100.
b) Method: Follow the system test procedures for UDC.100. Procedures define the equipment setup, initialization procedures, and execution steps. Analyze the results of testing to determine success or failure for each test.
c) Inputs/Outputs:
    1) Inputs
        i) System test designs
        ii) System test cases
        iii) System test procedures
        iv) User documentation
        v) Source and executable code
    2) Outputs
        i) System test reports
        ii) Anomaly reports

d) Schedule: Initiate after completion of system test procedures, completion of integration test execution and acceptance of the code by configuration management. Complete within six weeks.

e) Resources: One staff month is estimated for a software engineer.

f) Risks: System test execution for distributed systems such as UDC.100 has not been performed before. Schedule estimates are based on past experience. Earlier system test activities and initial system test execution times will be monitored to validate the schedule estimates.

g) Roles: Software engineer # 2 will execute the system tests.

A.3.35 Example for 5.5.6.2.3 Acceptance test execution

a) Task: Execute acceptance test cases for Elevator Control System UDC.100.

NOTE—This is to be developed when a customer is identified and the acceptance test procedure generation task in A.3.32 is completed.

b) Method: Pending

c) Inputs/Outputs:

1) The inputs are pending.

2) The outputs are pending.

d) Schedule: Pending

e) Resources: Pending

f) Risks: Pending

g) Roles: Pending

Annex B

(informative)

Critical software and criticality analysis

Criticality analysis is a methodology for allocating V&V resources to individual elements of the software. The goal of criticality analysis is to ensure the cost-effective use of these resources, especially when the resources available are insufficient to analyze all areas of the system at a uniformly thorough level. Simply stated, criticality analysis is a systematic methodology for applying V&V resources to the most important (critical) areas of the software.

Criticality analysis is related to the concept of critical software. Critical software, according to the Standard, is software *in which a failure could have an impact on safety or could cause large financial or social losses*. If a system satisfies that definition of criticality, the Standard requires that a specified minimum set of tasks be performed. The level to which these tasks should be performed is not, however, described in the Standard. In criticality analysis, a more systematic approach is taken in making decisions about critical software.

Criticality analysis is performed in four steps:

a) *Establish criticality levels to be applied to the system under development*. Criticality levels may be assigned values of High and Low; High, Medium, and Low; or one of a range of numerical values from 1 to *N*. In most cases the use of two or three levels is sufficient for planning purposes.

   There are frequently several different types of criticality relevant to a given system. Impact on safety, impact on the system mission, financial risk, and technical complexity are examples of types of criticality.

   Develop definitions for each level and type of criticality. For example, in the case of safety, the level High may be assigned if failure could result in personal injury, loss of life, or loss of property; Medium if failure would result merely in damage to equipment; and Low if failure would impact neither persons nor property. Similar levels are developed for each criticality type. These levels should be defined clearly and unambiguously, so that different analysts would reach the same criticality assignment independently. This goal is sometimes difficult to achieve.

b) *Identify a set of requirements sufficient to characterize the system*. This step should be part of the development process. Sometimes, however, a complete or useful set does not exist and must be developed. Agreement on what comprises a complete and faithful high-level representation should be reached with the developers, users, V&V agents, and overall program management. The actual level of detail will depend on the complexity of the system. Each identified requirement is then assigned a number and is cross-referenced to the paragraph number within a requirements specification document in which it is found.

c) *For each criticality type, assign a criticality level to each requirement, resolving multiple ratings into a single criticality level*. If the requirements are ambiguous or not quantitative, there may be different assessments of criticality level by separate analysts. This may indicate that the requirements need refinement or revision. In the case of simple differences, a conservative approach (especially for designated critical software) is to assign a higher (more critical) value.

   When there are multiple types of criticality, different ratings may be assigned for the different types. For example, suppose there are several criticality types including technical complexity, safety, and financial risk. In this example, suppose the criticality level for technical complexity is Low, safety is Medium, and financial risk is High. These different ratings may be resolved into a common rating in one of two ways.

   The first method is the more conservative and is recommended for systems designated “critical.” In this case, the criticality level of the requirement is the highest of the criticality levels for the different

types. The second approach is to average the ratings for the different types. Weighted averages can also be used. Rounding up is the conservative approach to resolve fractional averages. In the example above, straight averaging would result in a rating of Medium. If technical complexity were twice as important as each of the other two, Medium would again result because of rounding.

d) *Use the criticality level of each requirement to determine the V&V method and level of intensity appropriate for that requirement*. Depending on the criticality of a requirement, more thorough analysis may be required. For a high-criticality requirement, for example, an exhaustive technique might be selected and a specialized software tool developed for this technique. A low-criticality requirement may receive only low-level analysis, or none at all if resources are severely limited.

V&V planning is a subjective process. Criticality analysis is a methodology that guides the planner through this process in a way that allocates the most thorough analysis to the most critical requirements. It reduces the chance that significant requirements will be overlooked or insufficiently analyzed. Because of the stepwise nature of the process the decisions and trade-offs made in the planning process are more visible. This visibility allows review and critique of the plan to be made in a constructive way since individual planning assumptions and decisions may be identified and analyzed.

Annex C

(informative)

The seven planning topics

The Standard calls for seven topics to be addressed for each task or activity. There is no requirement to address each of the seven topics individually. The intent of the Standard is to indicate topics that should be covered. The structure and means of addressing these topics is left to the planner. This annex is intended to provide insight into what is meant by each of the topics and guidance for addressing them.

C.1 V&V tasks

> *Identify the V&V tasks for the phase. Describe how each task contributes to the accomplishment of the project V&V goals. For all critical software, the SVVP shall include all minimum V&V tasks for the management of V&V and for each life-cycle phase. Any or all of these minimum tasks may be used for noncritical software.*
>
> *Optional V&V tasks may also be selected to tailor the V&V effort to project needs for critical or noncritical software…. The optional V&V tasks … may be applicable to some, but not all, critical software. These tasks may require the use of specific tools or techniques…. The standard allows for the optional V&V tasks and any others identified by the planner to be used as appropriate. [3.5, (1)]*

When describing V&V tasks, include the name of the task, the task objectives with respect to the project V&V goals, and any task features that may affect a project goal or the V&V effort. The task descriptions establish specific objectives for the life cycle phase V&V.

When planning the V&V tasks, consider the following:

a) *The scope of each task*. Determine whether the entire software system is addressed or some subset of that system (i.e., identify to what products it applies and the selection criteria, if application is less than 100%).

b) *The V&V objective(s) served by each task*. Determine whether verification, validation, or some combination of these is to be addressed. Each task selected should demonstrably satisfy (in whole or in part) a project-level V&V or quality assurance goal. V&V tasks customarily have the following objectives:

   1) To verify that the software specifications and the outputs of each life cycle activity are complete, correct, clear, consistent, and accurately represent the prior specification
   2) To evaluate the technical adequacy of the products of each life cycle activity (e.g., assessing alternate solutions), and to cause iteration of the activity until a suitable solution is achieved
   3) To trace the specifications developed during this life cycle phase to prior specifications
   4) To prepare and execute tests to validate that the software products fulfill all applicable requirements

c) *The dependency relationships between the V&V tasks and other tasks* (e.g., from what task does this task take input or to what task does it provide output).

d) *The criticality of the software*. Determine the level of criticality of the software. The Standard defines a set of minimum tasks for critical software (see annex B for more detailed guidance on this consideration).

e) *The project environment*. Review and evaluate project plans, including schedule, change control, configuration management, and resources. These resources may be a limiting factor in choosing tasks that require the development of new tools or methods.

f) *The complexity of the product and development activity.* Estimate or evaluate product complexity, product risks, and development complexity (e.g., project size and organization).

g) *The number and types of expected defects*. If possible, estimate the number and types of expected defects. This information can often be determined theoretically or empirically. This information can be used to evaluate the feasibility and benefit of a V&V task.

h) *The experience and expertise of the staff*. An important factor to consider is the time and training required to competently execute V&V tasks.

C.2 Methods and criteria

Identify the methods and criteria used in performing the V&V tasks. Describe the specific methods and procedures for each task. Define the detailed criteria for evaluating the task results. [3.5 (2)]

Identify the methods used in performing the V&V tasks. A V&V task may use more than one method (e.g., code inspections and algorithm analysis for source code evaluation). A particular method may be used for more than one task (e.g., timing analysis for both design and source code evaluation). A chart, table, or other graphic representation may be used to show the relationships of V&V methods to tasks.

When selecting the methods, consider the following:

a) Critical features of the system
b) Integration of methods to achieve task goals
c) The organization and personnel performing the method
d) The project budget, schedule, and resources
e) The inputs to the task, such as
    1) Standards under which the inputs were developed
    2) Level of abstraction of the input
    3) Form of the input
    4) Expected content of the input
f) The outputs from the task; the purpose and destination of each; and the form, format, and expected content
g) Defects the method can be expected to uncover (e.g., misuse of a standard, an algorithm, a new technology, or a difficult feature)

Describe each method and its procedures. State how the method satisfies all or part of the goals of a V&V task required by the SVVP. State any unique information contributing to the selection of the method (e.g., the method is known to uncover a specific type of error common in such engineering problems.) If the method is applied only to some of the input required for the V&V task, identify the appropriate sections. When several methods will be applied to a task, describe the relationships between the methods and the procedures to be followed to ensure that the entire task is performed.

Identify the criteria for evaluating task results. These could include criteria for considering the execution of a task complete (e.g., unit structural testing will continue until 75% of the paths and 100% of the decisions have been tested).

C.3 Inputs and outputs

> *Identify the inputs required for each V&V task. Specify the source and format of each input... Identify the outputs from each V&V task. Specify the purpose and format for each output. [3.5 (3)]*

Identify the inputs to each task. There are two types of inputs for V&V tasks the development products to be verified and validated and V&V plans, procedures, and previous results. The development products are the objects of verification or validation. The V&V plans, procedures, and results guide the V&V activities. The V&V planner shall consider both types of input. It may be that specific forms or formats of the inputs are required when V&V is performed. In many cases, there may be contractual requirements for the format (e.g., IEEE standards, military standards), form (e.g., storage medium, specific word processor), and delivery of the inputs.

When specifying the inputs to the tasks, consider the following (where additional description is needed):

a) The source of each input. That is, the specific organizational element or person.
b) The delivery schedule for each input. If V&V planning is not integrated with other planning, inputs may not be received in a timely fashion. The schedule shall include sufficient time for reproduction and distribution as needed.
c) The temporal or logical relationships between the inputs when several inputs are required for a given task or method. For example, a task may begin without product A, but absolutely require product B and product C.
d) The state of the inputs (e.g., draft or official releases only). The evaluation of draft material is sometimes recommended to detect and correct errors early. This may not always be possible (e.g., for contractual or political reasons) and may often be undesirable (the material may change too rapidly).
e) The form (e.g., paper, 5.25 in floppy disk) and format (e.g., pure ASCII text) of the input. As with any other software specification, this must be unambiguous (e.g., floppy disks may be 3.5 in, 5.25 in, or 8 in, and may be in a large number of recording and word-processing formats).

Identify the outputs from each task. The outputs of a V&V task will be used by (at least) three audiences personnel performing other V&V tasks (including tasks in other phases of the life cycle), personnel performing other development tasks, and management. These audiences require technical feedback on the status of the products to determine how development should proceed. These audiences may have different specific information requirements to be satisfied.

a) Personnel performing other V&V tasks need detailed technical information to perform those tasks.
b) Development personnel require feedback about the development products verified or validated.
c) Management requires summary information about schedule, resource usage, product status, and identified risks.

V&V outputs include task reports, anomaly reports, test documentation, plans, and phase summary reports. V&V outputs are the tangible products of the V&V effort, and should be planned with care.

When specifying the outputs from the V&V tasks, consider the following:

— The form and format of each output (The same criteria may apply to output as to input, e.g., standards, media.)
— The destination of each output (specific personnel or organizational elements)
— The delivery schedule for each output (Output shall be provided in a timely fashion.)
— The state of each output (i.e., will drafts of an output or only official versions be released)
— The lifetime of each output (i.e., will a given report be used by later phases as input, or will it be an archived document for reference)
— The requirements for the configuration management of the output

C.4 Schedule

Identify the schedule for the V&V tasks. Establish specific milestones for initiating and completing each task, for the receipt of each input, and for the delivery of each output. [3.5, (4)]

This section shall describe the project life cycle and milestones, including completion dates. …describe how V&V results provide feedback to the development process to support project management functions…. When planning V&V tasks, it should be recognized that the V&V process is iterative. [3.4.2]

The purpose of scheduling is to establish and organize resources to meet defined objectives under the specified constraints of time and events. To be successful, the plan writer shall recognize that the schedule for V&V tasks is part of a larger schedule for an entire project. The available resources and constraints of time and events from the other interfacing parts of the project shall be determined prior to starting the scheduling. After the external environment has been established, the specific requirements for the V&V tasks shall be determined. Any resulting conflicts with the overall schedule shall then be resolved. The iterative nature of the schedule is a factor in planning during initial schedule creation as well as during the lifetime of the project. Most projects involving software encounter substantial change activity prior to release of the initial products. Although the extent of this change is rarely known at project initiation, the schedule should have mechanisms to respond to changes.

The V&V tasks are involved with all life cycle phases. The V&V planner should establish what organizational, schedule, and documentation interfaces are required with the organization responsible for each life cycle phase. Keep in mind at all times that the V&V tasks will be synchronized with the respective project activity (e.g., requirements analysis cannot begin until the requirements documents have been produced).

When scheduling a V&V task with an organization responsible for a specific life cycle phase, consider the following:

a) Due dates for required inputs, such as specification, detailed descriptions, source code, or other data
   1) Establish completion criteria for each input (e.g., requirements released to configuration control) as well as the media and format for transmittal
   2) Schedule for any conversion required (e.g., entering requirements into electronic format, transferring between different formats)
b) Scheduling of participation in internal reviews of particular life cycle phases to ensure that V&V concerns, such as testability, are addressed during product creation
c) Due dates of the results of each task to project management to be responsive to program needs
d) Access to key personnel from the interfacing organization for consultation

Once the requirements from other organizations have been established, the specific V&V tasks for each of the life cycle phases should be determined. For each V&V task selected, the completion criteria should be specified before the resources required for that task can be determined. Estimating required resources can be accomplished by using results of previous projects, various automated tools based on product and project parameters, or outside consultants. These estimates can be effected by the use of tools. Although these estimates will have a degree of uncertainty at the beginning of the project, they can and should be updated throughout the project as better information is available.

When scheduling V&V tasks within a specific life cycle phase, consider the following:

a) Scheduling of V&V outputs and reports to provide timely and effective feedback to development personnel
b) Allowing for the evaluation and internal review of results by V&V management
c) Defining explicit V&V task stopping criteria
d) Defining criteria for establishing the amount of regression testing required due to changes

As with general project scheduling there are many formats for schedule presentation (e.g., Gantt Charts, PERT, CPM) and in some cases analysis of schedule flow. The approach used should be consistent with other project elements. Use of automated tools is encouraged. The criteria and approval process for implementing schedule changes should be specified.

C.5 Resources

> *Identify the resources for the performance of the V&V tasks. Specify resources by category (for example, staffing, equipment, facilities, schedule, travel, training). If tools are used in the V&V tasks, specify the source of the tools, their availability, and other usage requirements (for example, training). [3.5 (5)]*

Resources may be defined as sources of supply or support, in (perhaps artificial) contrast to inputs, which are the objects of verification or validation. Resources typically required for V&V tasks are personnel, manual or automated tools, and finances. Other resources may be required for specific projects (e.g., security controls, travel resources).

Specify staffing resources by identifying the people who are responsible for the given task, either by name or in more general terms (e.g., by job title or classification). In the latter case, provide sufficient detail to insure the selection of personnel capable of adequately performing the task.

When planning the staff for the V&V tasks, consider the following:

- The number of persons or full-time equivalents (e.g., two people half-time). Where staff support is readily available, this may be a statement of expected personnel use. Where staff is in short supply, this may be a key constraint on the tasks and methods selected, or on the schedule or coverage of the selected tasks.
- The required skill level and specific experience. Clearly identify unique requirements (e.g., senior analyst with experience using method X to perform requirements tracing for secure operating systems). Consider staff experience and expertise when predicting the effectiveness of V&V tasks, and when evaluating the relevance of historical data. An inexperienced staff can be expected to introduce (or fail to detect) more defects than a staff of experts. Exercise care not to overspecify skill levels; descriptions should specify the minimum levels necessary to perform the tasks. This allows maximum flexibility in staffing each task. If individuals with requisite experience are likely not to be available, plan for training (or acquiring) skilled staff. Indicate the lead time required for staff development. Identify other credentials (e.g., security clearances, bonding) required, and plan appropriately.

Identify any manual or automated tools that are to be used in performing a task. Checklists and documented, standardized V&V procedures are examples of manual or nonautomated tools. When planning for the use of a manual tool, consider the following:

- The tool should be sufficiently applicable to the task that it produces meaningful, specific results. Otherwise the adaptation of the tool to the project should be planned. For example, generic checklists should be tailored to the needs of the project.
- The tool should either have been qualified or evaluated prior to its use in the given task.

Automated tools include software tools, simulations (hardware, software, or hybrid), test environments, test beds, and the use of electronic communications. As with manual tools, explicit description of their use is essential to good planning and to obtain their benefit to the project.

When planning the use of an automated tool, consider the following:

— Identifying the specific tool (including version identifier, if available) and any actions necessary to prepare for its use (e.g., development, tailoring, adaptation, qualification)
— Identifying special resources needed for its use, including computers and other hardware, databases, test beds, facilities and access to them, training, and maintenance
— Planning for licensing or other contractual arrangements necessary for use
— Planning for achieving task objectives if usable tools are not available

Financial resources may be addressed, although this is usually not done on a task-by-task basis within the SVVP. Special financial considerations, such as trade-offs that might be made during the execution of a given task, could be included.

Include travel requirements for a task (e.g., to attend design review meetings), and cross-reference travel plans to the schedules. Other resources that could be addressed are special procedures, such as security controls. In general, these and other topics should be included at the task level only when they are specific to the task and differ from the procedures in place for the project as a whole.

C.6 Risks and assumptions

> *Identify the risks and assumptions associated with the V&V tasks, including schedule, resources, or approach. Specify a contingency plan for each risk. [3.5 (6)]*

Any plan is based on a number of assumptions. These assumptions create elements of risk in the plan. If the assumptions are violated, the project will not proceed as planned. In many cases, the risks may be mitigated if the assumptions are documented explicitly and contingency plans prepared.

Contingency plans should be realistic. They need not be spelled out in detail unless warranted by the level of risk and the criticality of the V&V task. If the only alternative is a general slip of the project schedule, this should be stated. When documenting risks and assumptions, consider the following:

a) *The software project schedule*. A normal assumption is that all software or system components and deliverables will be delivered on schedule and that the deliverables will be complete, controlled, and up-to-date. By documenting which tasks depend on other tasks for input or for resources, the potential disruption of the project because of late deliveries may be minimized.
b) *The software project complexity.* When the software architecture, components, or deliverables are technically complex, there is a risk that the software will be insufficiently evaluated in early phases, leading to increased errors in later phases. When the software project organization is complex, there is a risk of miscommunication between the elements, leading to interface and documentation errors. When the V&V tasks are complex, particularly in relation to available personnel and other resources, there is a risk that the schedule will not be met.
c) *The availability of adequate resources*. When there are insufficient human resources (e.g., skills, experience) or inadequate material resources (e.g., computing resources, test beds, load generators, code exercisers, code static analyzers), there is a risk that the V&V tasks will be performed inadequately.
d) *The presence of adequate product development processes (e.g., configuration management, reviews, standards)*. This affects the number of expected defects, which will affect any V&V effort prediction. A normal assumption is that there will be timely response to informal comments (if applicable) and anomaly reports.

Additional risks possibly present in software verification and validation efforts include

— Development and first use of software tools
— Use of COTS
— First applications of V&V techniques to larger or more complex applications
— Changes in life cycle models

C.7 Roles and responsibilities

> *Identify the organizational elements or individuals responsible for performing the V&V tasks. Assign specific responsibilities for each task to one or more organizational elements. [3.5 (7)]*

Define specific roles and responsibilities for each task, including individually specifying planning, performing, reporting, evaluating, coordinating, and administering if meaningful. Assign each of the roles to specific individuals or organizational elements. In many cases, some roles will be defined by the required inputs and outputs for the task. For example, for a source code walkthrough, some person will supply the code, someone else may distribute the code to the review team, and others may make up the review team. More than one role may be assigned to an individual or organizational element.

Identifying, by name, the personnel who are responsible for a task makes the construction and coordination of a master schedule simpler. For example, periods of overcommitment or underuse of a given person can be more readily identified. Individual participants in a project may easily identify and prepare for their roles by reading the SVVP.

On the other hand, identification by name alone has shortcomings. If personnel change, the rationale for selecting a specific individual for a task may be lost. For large or long projects, there may be too many individuals and too many uncertainties to identify in a reasonably sized plan. For these reasons, it is often a good idea to identify personnel by organization or job classification.

A matrix, table, or chart may be used to assign V&V tasks to organizational elements and to job classifications or individuals within the effort. The plan should be as specific as possible (e.g., specify the number of people for a task). When appropriate, in a proposal for example, the biographies of personnel may be included (perhaps in an appendix). For large projects, the assignment of responsibilities to individuals may be placed in an appendix or in subsidiary documents.