CVISN Guide to Integration and Test

POR-99-7194

Draft Version D.1

May 2001

CVISN Guide to Integration and Test

POR-99-7194

Draft Version D.1

May 2001

Note

The Motor Carrier Safety Improvement Act was signed into law on December 9, 1999. This act established a new Federal Motor Carrier Safety Administration (FMCSA) within the U.S. Department of Transportation (DOT), effective January 1, 2000. Prior to that, the motor carrier and highway safety program was administered under the Federal Highway Administration (FHWA).

The mission of the FMCSA is to improve truck and commercial passenger carrier safety on our nation's highways through information technology, targeted enforcement, research and technology, outreach, and partnerships. The FMCSA manages the Intelligent Transportation Systems (ITS) / Commercial Vehicle Operations (CVO) program, a voluntary effort involving public and private partnerships that uses information systems, innovative technologies, and business practice re-engineering to improve safety, simplify government administrative systems, and provide savings to states and motor carriers. The FMCSA works closely with the FHWA ITS Joint Program Office (JPO) to ensure the integration and interoperability of ITS/CVO systems with the national ITS program.

Acknowledgements

The materials included in this guide are gathered from many sources. Many of these are published documents, and these are cited as references. Many other ideas were drawn from informal conversations, working notes, presentations and e-mails that have occurred during the course of the CVISN program.

Draft Issue

This is a draft document. All sections included are complete and have been reviewed by JHU/APL, but not by other DOT contractors or by state/federal government agencies. The purpose of this issue is to obtain comments and feedback on this document from those external organizations before a baseline version is published.

This document is disseminated in the interest of information exchange. JHU/APL assumes no liability for its contents or use thereof. This report does not constitute a standard, specification, or regulation. JHU/APL does not endorse products or manufacturers. Trade and manufacturer's names appear in this report only because they are considered essential to the object of this document.

Note: This document and other CVISN-related documentation are available for review and downloading by the ITS/CVO community from the JHU/APL CVISN site on the World Wide Web. The URL for the CVISN site is **http://www.jhuapl.edu/cvisn/**

Review and comments to this document are welcome. Please send comments to:

Ms. Mary W. Stuart
Johns Hopkins University/ Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099
Phone: 240-228-7001 E-Mail: mary.stuart@jhuapl.edu

CVISN Guide to Integration and Test

Table of Contents

This Page Intentionally Blank

1. INTRODUCTION

The Commercial Vehicle Information Systems and Networks (CVISN) Guide to Integration and Test provides guidance for planning and organizing the test and integration elements of a state's CVISN program. This is one in a series of guides. This document and other CVISN-related documentation are available for review and downloading by the ITS/CVO community from the JHU/APL CVISN site on the World Wide Web. The URL for the CVISN site is: **http://www.jhuapl.edu/cvisn/**.

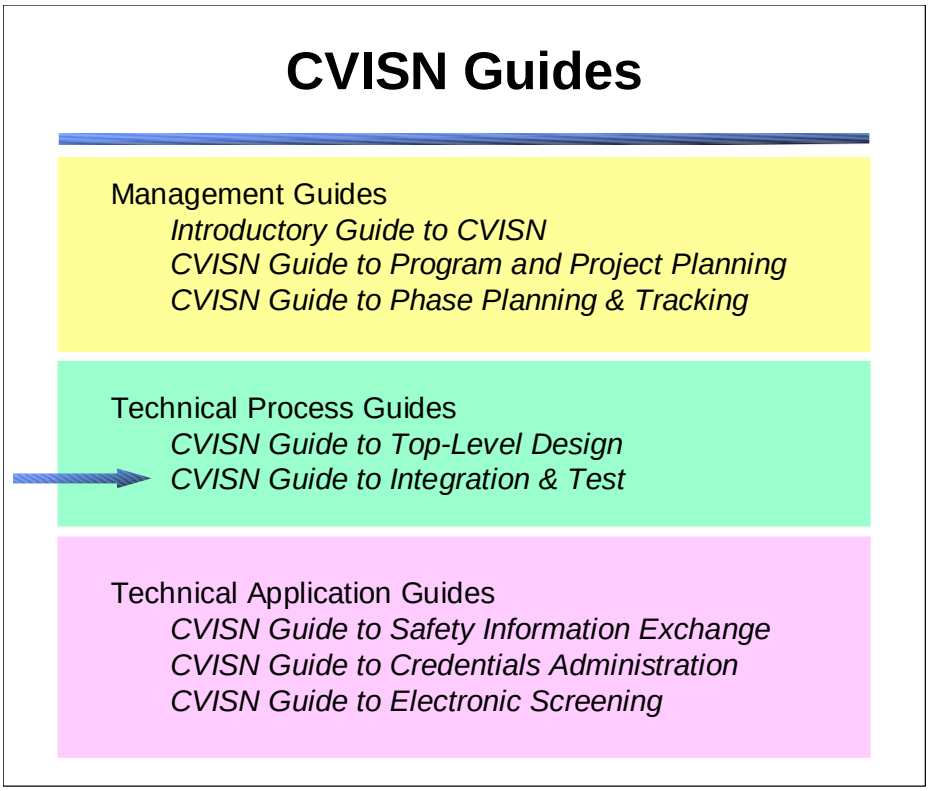

Figure 1–1. CVISN Guide Series

Testing is the process by which you ensure that the CVISN systems you build meet your needs. Testing is important throughout the entire system life cycle of a project, and you should view testing as a continuous activity with each development phase producing some test products. This guide, the ***Commercial Vehicle Information Systems and Networks (CVISN) Guide to Integration and Test,*** focuses on the phase of testing that begins at system integration and

concludes with system-of-systems testing. We call this phase of the development life cycle ***integration and test***. Within the integration and test phase, there are five distinct types of testing:

1. Integration testing
2. System testing
3. Acceptance testing
4. System-of-systems test and evaluation
5. Interoperability testing

This guide will help you plan for and implement the integration and test phase for your state's CVISN projects. You will develop test plans that will define your test objectives, coordinate a testing strategy, and plan all steps and tasks in the testing process to ensure that your testing efforts are efficient and complete.

The remainder of this document is organized to help you to understand integration and test, to produce the test plans and documentation that will guide your efforts, and to execute those plans.

- Chapter 2 defines the scope and concept of integration and test. It discusses what to test and describes types of tests.
- Chapter 3 provides an overall description of the steps in a generic test process. These steps are common to all types of testing.
- Chapter 4 discusses unique characteristics of each type of test.
- Chapter 5 discusses how you can achieve the desired level of confidence in the systems you build while staying within cost and schedule. The duration of integration and test inevitably involves trade-offs between confidence and cost and schedule.
- Chapter 6 discusses the documentation that supports integration and test. Adequately documented test plans will help you get organized and ready to test at each level.
- Finally, Appendix A provides additional references.

2. DEFINITION AND SCOPE OF INTEGRATION AND TEST

2.1 Scope

IEEE Standard 1220-1994 [20] defines the "pieces" of a system as ***components, assemblies, subsystems, and products***. The relationship between the pieces of a system is illustrated in Figure 2–1.

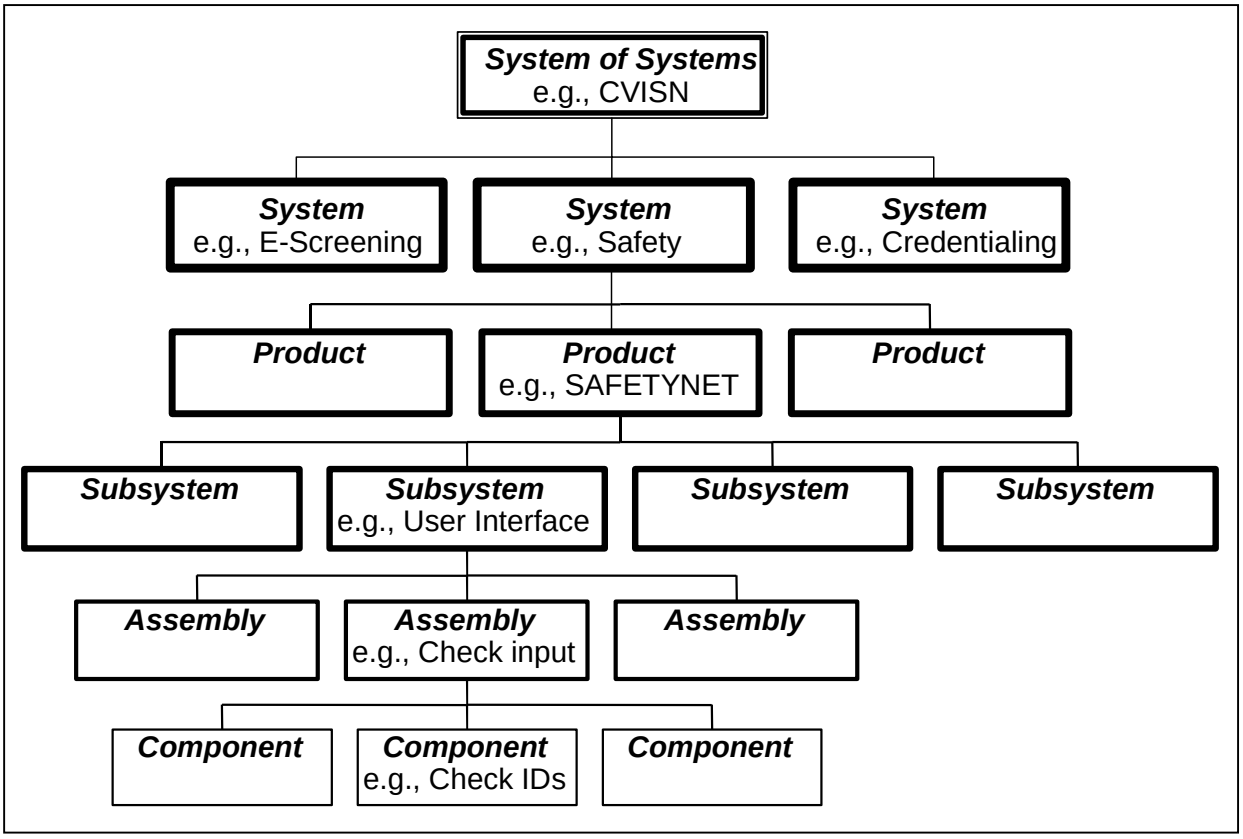

Figure 2–1. A System Consists of a Set of Consumer Products

In the complete scope of the project life cycle, testing occurs at each stage, and integration occurs between sequential stages. Components are the fundamental elements of a system. Developers must test each individual component to ensure that component was built as designed. Components are grouped together to form an assembly. An assembly performs a function, such as the part of the user interface that checks inputs. Developers test assemblies to ensure that they properly execute the functions for which they were designed. A subsystem consists of a group of assemblies. The subsystem is typically the first item that is recognized as a complete unit, for example a complete user interface. To thoroughly test a subsystem, developers must simulate the actions of the other subsystems that interconnect with their subsystem. By interconnecting a subsystem to stubs or simulators, developers can typically conduct very thorough subsystem

tests. Again, testing is for functionality according to design. Even though the subsystem is typically recognizable as a unit, it is not a final product. The subsystem is integrated with other subsystems that together form a real product. Multiple products working together then become part of a state's system. By carefully planning and executing the tests for the smaller pieces, developers can significantly reduce the complexity of the Integration and Test phase. The testers can concentrate on the product and system interfaces and requirements rather than the detailed internal capabilities (and possible anomalies) of the smaller pieces. Many systems can also work together comprising the CVISN system of systems.

The system development process typically begins with requirements definition, proceeds with top-level and detailed design, unit development, unit test, and then Integration and Test. A "unit" refers to the components, assemblies and subsystems of the IEEE hierarchy. In the integration and test phase, we assume that the units have been thoroughly tested and perform as expected. There is a lot of testing of these pieces that occurs before the Integration and Test phase and that is outside the primary scope of this document. In this phase, and in this document, we are only concerned with the following tasks:

- Integrating subsystems into products and testing completed products
- Integrating products into systems and testing completed systems
- Interoperability testing among products and systems
- System-of-systems testing of multiple systems.

The scope of the integration and test phase – and of this document – is limited to that portion of the development cycle beginning with product integration and concluding with system-of-systems test and evaluation. ***The remainder of this document is specifically about the integration and test phase and not about the testing of the pieces of a system.***

2.2 Test Team and Conformance Assurance Team

It is beneficial to have a designated Test Team, separate from and independent of the development team. There are several reasons for this:

- Integration and Test is a major effort. A dedicated Test Team is more likely to conduct this effort with the care and planning it requires.
- The Development Team has a vested interest in having the system work as designed. An independent Test Team is more likely to be objective it its testing.
- A separate Test Team will more thoroughly test the functions of the system. The Development Team will know many shortcuts and work-arounds and will be less thorough in testing.

Sometimes resource and funding constraints do not support a separate Test Team. If, however, you can afford an independent Test Team, the benefits will be significant.

You should also establish a Conformance Assurance Team for each CVO project, task, or activity that uses federal funds. The Conformance Assurance Process will assess consistency with the National Intelligent Transportation Systems (ITS) Architecture, the CVISN Architecture, and the International Border Clearance Architecture. The Conformance Assurance Team should include, at a minimum,

- The State CVISN System Architect,
- Someone who knows the National ITS, CVISN, and IBC architectures and standards; and
- Someone with experience on projects having regional or national implications and working with FMCSA Division Office and Service Center staff, as well.

This group should review the CVISN design and implementation choices, keep up with modifications to the national architectures, and evaluate the state's conformance with the architectures.

The CVISN system architect knows the project's objectives, leads the requirements analysis process, and leads the system design process. The CVISN system architect knows or will learn about all the state systems involved. The architecture and standards expert must be very knowledgeable about:

- National ITS, CVISN, and IBC architectures
- ITS/CVO standards
- CVISN core infrastructure systems
- Experiences of other states that have deployed or are deploying CVISN

The expert should have access to the architecture and standards documents and know them well. The architecture and standards expert is an on-call consultant – especially during the requirements and design phases – and reviews the requirements and design products; helps tailor the interoperability tests; and analyzes the test results.

2.3 Definition

The integration and test phase begins when you begin product integration; i.e., to combine subsystems into a product. In the context of your state's CVISN development, think of integration and test as comprising the following five types of tests:

1. Integration test
2. System test (when done at the product level, we also refer to this as Product Test)
3. Acceptance test
4. System-of-systems test and evaluation
5. Interoperability test

These types of test and their relationship with the development cycle are illustrated in the classic "V" diagram shown in Figure 2–2. The first three terms (integration test, system test and acceptance test) are commonly used in system development literature to describe integration and test activities. The last two terms (system-of-systems test and interoperability test) were defined specifically for the CVISN program, and are not generally used in test literature. However, the concepts they embody are commonly used.

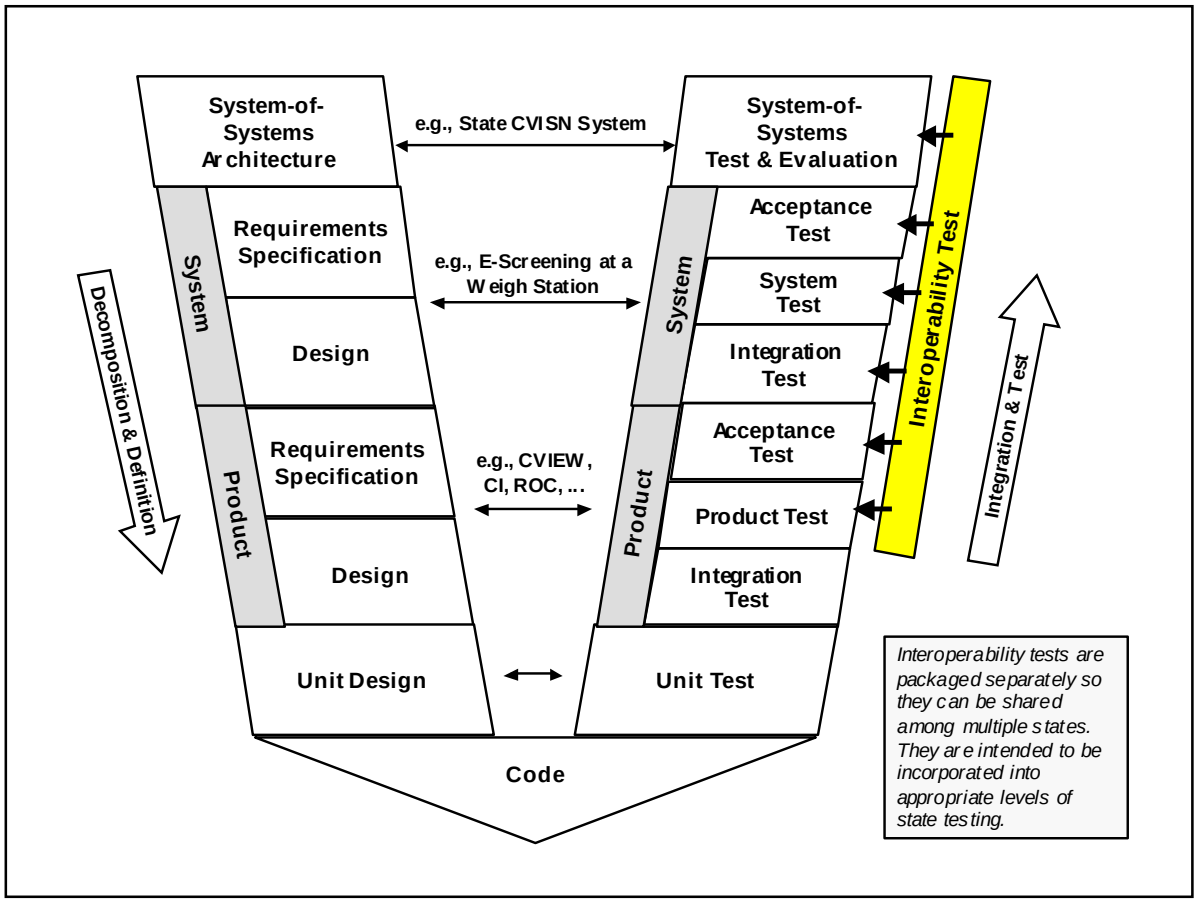

Figure 2–2. The Verification "V" Shape Relates Development Tasks to Corresponding Testing Tasks

Each of the first three types of tests (i.e., integration, system, acceptance) can be applied at either the product or the system level as shown in the "V" diagram. The nomenclature gets somewhat confusing because we speak of "system testing" the product. This is because the word "system" is used in a general sense in the testing world to mean a collection of components that work together to carry out some function. However, the IEEE definition of "system" refers to a specific level in their hierarchy (Figure 2–1). Although this double meaning for system is sometimes clumsy, the context usually makes the intended meaning obvious.

Each of these five test types is critical to the successful deployment of CVISN systems. They are summarized in Table 2–1 below and described further throughout this document.

Table 2–1.
Major Types of Tests in the Integration and Test Phase

Test Type	Test Focus
Integration Test	• Internal interfaces among elements (elements being either units within a product or products within a system). • Bringing elements together and testing using successively more complete "builds • Testing individual functional capabilities
System Test	• External interfaces among products or systems • Multiple functions in an integrated product or system • Product or system performance, capacity, reliability, availability and other "ilities"
Acceptance Test	• Demonstrating product/system capabilities in an operational setting • Satisfying product/system requirements as proven by meeting test criteria • The last step prior to formal acceptance of the contract deliverable • Demonstrating suitability for operational deployment
System-of-Systems Test & Evaluation	• Demonstrate end-to-end capabilities (Does the system do what we wanted?) • Evaluate whether the system achieves the anticipated benefits (Does use of the system achieve the result we anticipated?)
Interoperability Test	• Pair-wise testing of products with CVISN standard external products such as SAFER, or with each other through standardized interfaces • End-to-end functionality of your system and with the other systems that comprise the CVISN operational domain

2.3.1 Integration Testing

Integration testing can occur at two levels: integrating subsystems into products and integrating products into systems. To simplify discussion, we will talk about the level of integrating products into a system in the remainder of this section. The same thoughts apply to integration testing at the product level, with a slight variation in nomenclature.

The group of products that comprise your system must work together reasonably well before it makes sense to test the functionality of the system as a whole. Integration Testing is where you begin to combine products to test their interfaces and to confirm that they link together properly. This process ensures that there is at least a skeleton system that can serve as the platform to begin System Testing. In practice, Integration Testing often encounters two major problems:

1. Integration is premature due to inadequate design and development and/or lack of proper product testing.
2. Integration testing lacks formality in planning and execution.

System Testing cannot begin until you address these problems. If you start System Test too early, you will be frustrated by problems at the product and internal interface level. To avoid the first of these problems, it is important to set firm test criteria for product testing and to test products thoroughly for requirements and interface specifications. These tests will determine a product's readiness for integration. To avoid the second problem, you must ensure that Integration Testing is carefully planned, documented, and thorough. You must test all interfaces systematically against well-defined completion criteria. Thorough, well-planned Integration Test will allow you to begin System Testing without underlying product and interface problems.

Sometimes it is wise to include an Integration Readiness Review in the development process, especially if elements have been developed by physically or organizationally separate teams. The Integration Readiness Review allows representatives from these teams to get together, exchange information, and assess the degree of compatibility of their elements prior to fielding teams for complex and perhaps expensive integration testing.

2.3.2 System Testing

Like Integration Testing, System Testing occurs at the product and system level. To simplify discussion, we will talk about the system level of system testing in the remainder of this section.

System Testing looks at functional performance and system behavior. Sometimes, particularly for smaller projects, Integration Testing and System Testing are combined. Hetzel [1] defines the beginning and end of System Testing as follows:

- System **test design and test development** begin in the requirements phase with the design of requirements-based tests.
- System testing execution begins when a minimal system or skeleton has been integrated.
- System testing ends when developers have measured system capabilities and corrected enough of the problems to have confidence that we are ready to run the acceptance test.

Integration Testing, System Testing, and Acceptance Testing run more or less sequentially. In an ideal development project, developers would integrate all of the products, test the interfaces between products through Integration Testing, and then run the System Tests. In practice, Integration Testing and System Testing are iterative and overlap. This is especially true if product testing was inadequate or if, due to lack of simulators and stimulators, it was impossible to adequately test the products' interfaces prior to integration.

2.3.3 Acceptance Testing

Acceptance testing evaluates the readiness of a product or system for deployment. Normally, a user familiar with the product or system conducts the acceptance test. Because of the well-planned and thorough testing that has already been accomplished, the **acceptance test should not reveal any major surprises**. Acceptance tests should run smoothly and require minimum time. Acceptance testing should demonstrate that the product or system is reliable and usable. In many cases, CVISN Interoperability Tests may be also used as part of acceptance testing. Acceptance testing is sometimes a contractual requirement for a developer or system integrator to demonstrate the system's capability and function prior to receiving compensation. Of course, Acceptance Testing will always reveal some minor problems, but just document these for future fixes and enhancements and concentrate, instead, on overall functionality and performance.

2.3.4 System-of-Systems Test and Evaluation

System-of-systems test is very different from the lower level testing. This type of testing is not generally discussed in the literature on testing. Most testing is focused on how to prove that a specific product or system does what it was supposed to in the way that was required or intended. A system has a well-defined scope and accomplishes a set of functions relatively independently. A system-of-systems brings together multiple systems that each carry out some independent function. A system-of-systems accomplishes some larger function that spans multiple systems. System-of-systems test is focused on demonstrating that these capabilities that span multiple systems work as intended. It is usually done with a relatively few test scenarios, because exhaustive testing of all possible combinations of functions in the various systems under test would be too expensive and time consuming.

System-of-systems evaluation consists of collecting data on actual system operation and measuring whether the system achieves the results anticipated.

CVISN is a system-of-systems. Each of its major systems (safety, screening, credentials) accomplishes a significant function in a stand-alone sense. However, working together these three systems can accomplish some functions with an efficiency and effectiveness that cannot be achieved independently. For example, a screening system can do a better job of sorting vehicles if it has current registration data provided from a credentialing system, or out-of-service data provided by a safety system.

2.3.5 Interoperability Testing

The purpose of interoperability testing is to:

- ensure that designated external product and system interfaces conform to architectural standards
- ensure that multiple systems work together to provide designated end-to-end (i.e., spanning multiple systems from initial input to final result) capabilities.

Interoperability Testing applies at the product, system and system-of-systems level (please refer to the "V" diagram in Figure 2–2). As you develop your CVISN systems, it is critical to include Interoperability Testing in your overall testing strategy to ensure that your state systems are in conformance with the architecture and are therefore interoperable with products and systems used by other states, carriers and Federal agencies. There are two types of interoperability tests: pair-wise interface tests and end-to-end function tests.

Pair-wise interoperability tests verify that selected products or systems within your enterprise can interoperate with CVISN standard external products and systems and that they use applicable external standardized interfaces properly. The success of pair-wise interface testing depends on the conformance of your products to published CVISN interface standards. Interoperability criteria govern interfaces between pairs of products or systems for electronic credentialing, electronic screening, and safety information exchange. Pair-wise interoperability testing examines three kinds of interfaces: DSRC, Custom Interface Agreements for selected interactions (e.g., SAFER-ASPEN), and EDI. The applicable interface standards for CVISN include

- DSRC according to ASTM and IEEE standards for physical and message layers,
- EDI according to ANSI ASC X12 standards and implementation guides, and
- De-facto custom interface agreements for products supplied by FMCSA (such as ASPEN and SAFETYNET).

XML (Extensible Markup Language) promises to be a significant standard for exchange of information through interfaces. Future interface standards may employ XML rather than, or in addition to, EDI.

End-to-end function tests (the second type of interoperability test) verify dataflow and data usage between products in your CVISN systems and among your systems and external systems. End-to-end tests verify the functional capability of the CVISN system-of-systems as a whole. Web interfaces for credentialing are also verified through end-to-end tests.

Interoperability Testing is not sequential with the other four types of test identified here. Interoperability testing should begin as soon as the products that use CVISN standards can interface together. You will run and rerun these interoperability tests throughout the Integration and Test phase and whenever product versions are rebuilt. Interoperability tests are special cases of system tests and system-of-systems tests, focused on the interconnections and interoperability of pairs or groups of products or systems.

3. THE COMMON STEPS IN TESTING

The Integration and Test process enforces discipline on the test design and execution and ensures that the required steps are carried out in the appropriate sequence. Figure 3–1 lists the test process steps. These high level steps are the same for integration, system, acceptance, system-of-systems and interoperability testing. In Chapter 4, you will see that each of these specific types of test also has more, specific unique considerations.

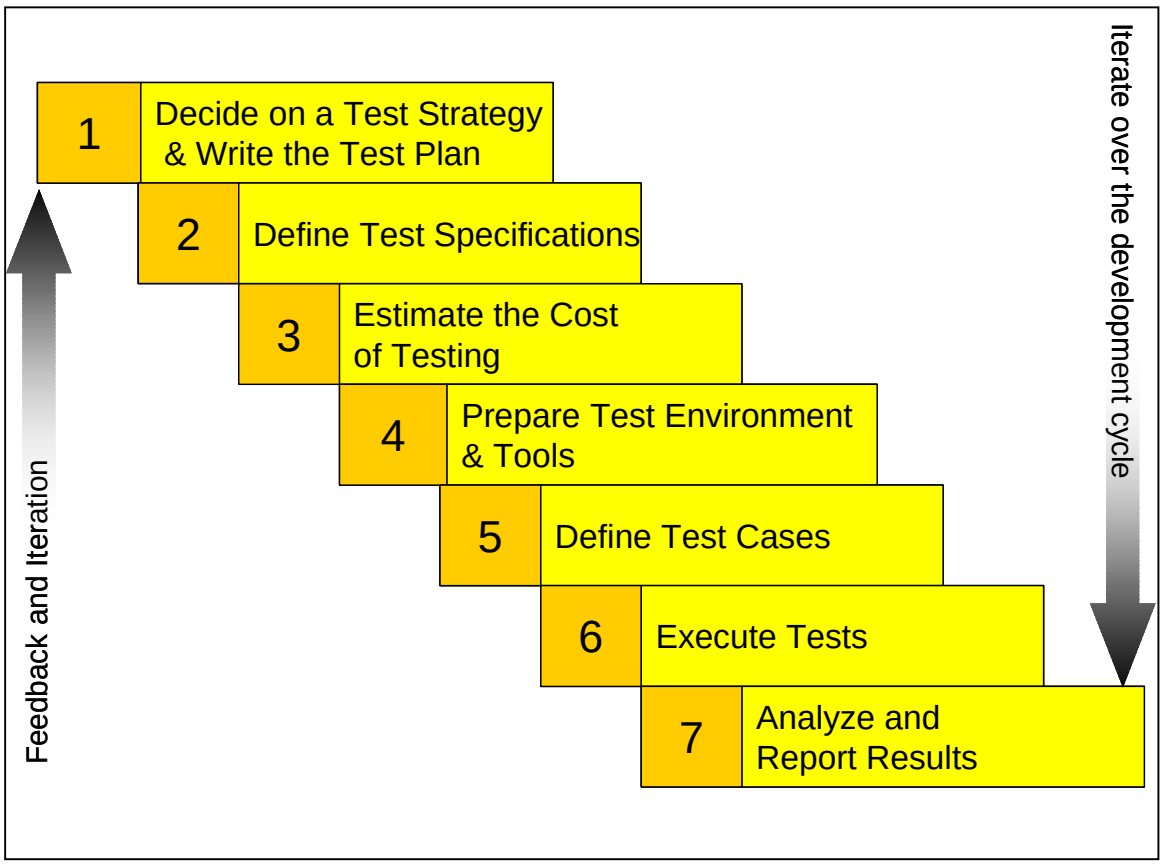

Figure 3–1. Steps in the Testing Process

Note that although the steps are presented as sequential, there is a considerable amount of feedback and iteration. For example, the test plan produced in the first step will have to make rough estimates of cost and scope before the details of steps 2 and 3 are available. As steps 2 and 3 proceed, the original estimates will need to be updated.

3.1 Step 1: Decide On a Test Strategy & Write a Test Plan

It is useful to spend some time thinking about the strategy you will use for testing. You should address the following considerations:

1. Establish high-level objectives and expectations
2. Establish the level of organizational commitment
3. Define the overall approach to testing
4. Assign roles and responsibilities
5. Establish the level of formality and documentation required for test planning
6. Establish the level of tracking and reporting expected for integration and test
7. Establish any tools, methods, or mechanisms to be used to conduct tests, track changes, and report problems
8. Prepare an initial budget estimate
9. Align and reserve resources including people and facilities
10. State organizational, legacy system, schedule, and budget constraints

By establishing this high-level test strategy up front, you pave the way for those that will plan and execute integration, system, acceptance, system-of-systems and interoperability testing.

The strategy should be documented in a test plan. You may choose to have one, overall master test plan. Or it may be convenient to package the test plan as several documents, each corresponding to a different type of test (e.g., integration test plan).

3.2 Step 2: Define Test Specifications

An important step in test design is to establish the requirements to be tested. ***Writing a good set of requirements at the outset of a development project is probably the most important step in the success of the project, in general, and Integration and Test, in particular.*** The requirements have to be complete, so that a system that meets those requirements truly meets your needs. The requirements also have to be testable. In other words, you must be able to execute a test and clearly observe whether the system passes or fails some predefined test criteria. Many test criteria will be based on simple observations – did the system meet the requirement or not. Others may require quantitative measurement; e.g., was the system able to process a transaction quickly enough.

When you are writing down your requirements, you should also be writing down the test that will enable you to test that requirement. The requirements are gathered together in a large table called a requirements traceability matrix that you will use throughout the project to ensure that you are testing and meeting your requirements at each step. Each requirement is a row in the table. Some columns are used to show the correlation between system elements and requirements (which modules satisfies which requirements). Other columns are used to show the correlation between tests and requirements (which tests check which requirements). As you progress through the development life cycle, you will refine test identification to the appropriate level.

A **test case** is a specific test scenario involving a specific test configuration, test inputs, test outputs and a pass/fail criteria. A **test** is a set of one or more test cases. A ***test specification*** is a high-level description of the design of a test.

In the Scope Workshop, state participants develop their CVISN Top Level Design. As part of this exercise, participants develop key scenarios to ensure that all design elements are included, and that essential functions are addressed in the overall design. These same scenarios will most likely correspond to your key tests, as well. As the design progresses, there will be more information about the items to be tested. Keep refining the scenarios and identify the ones that are most critical. Derive your test specifications by analyzing these scenarios. Within each test, you will identify one or more (usually more) specific ***test cases***. Think of a test case as a specific exercise for the system or a specific function within a mode of operation – when the user types "A" and presses enter, the system should do "B".

In developing your test specification, you are going to make a list of tests. For each test you develop, you should include the following descriptive information:

- Test identifier and name
- Summary – give a brief description of the test
- Test item – describe what is being tested
- Function being tested – summarize what functional capability this test is exercising
- Test sequence – summarize the general sequence of events in the test
- Test cases – list the types of cases that should be included (e.g., nominal case, special conditions, error conditions)
- Requirements tested – maps the test case back to the requirements

CVISN has extensive existing materials that you can use for interoperability testing. The "CVISN Operational and Architectural Compatibility Handbook (COACH) Part 5 – Interoperability Test Criteria" [17] contains the criteria that you will use. The "CVISN Interoperability Test Suite Package, Parts 1 and 2" [6, 7] contain extensive description of the tests (referred to therein as test scenarios), test cases, and procedures that apply to CVISN interoperability testing.

3.3 Step 3: Estimate the Cost of Testing

Testing can be expensive, and testing will reveal defects that will be costly to correct. The consequences of not testing, however, can be disastrous. Still, it is important to use test resources wisely. To estimate the cost of testing, you must consider:

- At what phase of the life-cycle will testing occur
- How many test cases will be required
- How much preparation is needed
- How much effort is required in execution
- What special tools and environment are required

3.3.1 Detecting Defects Early

The life cycle testing approach is designed to reduce the cost of testing by detecting system defects at the earliest point possible. As a rule of thumb, it costs ten times as much to fix a defect during the System Testing phase as it does during the Requirements and Design Phases; and one hundred times more to fix the defect after the system is in use. To compute the cost to repair a defect, you have to determine all the changes that will be required to the software and documentation. Late in the life cycle, when interdependent products have been integrated into a complex system, small corrections can have major "ripple effects," and it can be very difficult to compute the actual *total cost* of a defect.

3.3.2 Number of Test Cases

If you know how many lines of code exist, you can use typical industry practice to estimate the number of test cases required. At the system test level, these results indicate that 1 to 10 test cases are required for every 300 to 500 lines of code. If line of code data is not available, another method is to estimate the number of detailed features (sometimes called function points) that the system has. A feature can generally be expressed in a one-line sentence, for example, "Write output to database." Plan, roughly, for 2 to 3 test cases per feature. The number of test cases required depends, of course, on the importance of the feature and the consequences of failure. Especially important or high-risk features might require 10-20 test cases each [11].

3.3.3 Test Case Preparation

Test case preparation time is the time it takes to develop a test case, document the test procedures, review the test case and procedures, and issue the test plan for execution. Start by estimating the number of test cases needed for a test, and figure out the average time it will take to prepare each test case. Keep in mind the following factors [11] when preparing these estimates:

- Product complexity and size
- Product stability versus volatility
- Readiness to test
- Risk inherent in the product
- Efficiency of the support infrastructure and test execution process
- Tester skills and resource availability

There are many predefined test cases for interoperability testing in "CVISN Interoperability Test Suite Package, Part 2: Test Cases and Procedures" [7].

3.3.4 Test Environment Set-up

Before you can execute specific test cases, you may have to set up a test environment. This can be a trivial or an extremely costly and complex part of the total test effort. It may be necessary to replicate the real environment in a test configuration so as not to disturb operations. There may be additional infrastructure costs just to support the tests; e.g., test lab, networking, communications, databases, system administrators, and computer systems. Time will be needed to design and set up the test environment, wait for vendor products, and, of course, test the test environment.

3.3.5 Test Case Execution

Once you have prepared the test cases and established a test environment, you are ready to conduct the tests. Some tasks to consider in test execution include:

- Preparation (e.g., time to read and understand the test case)
- Set-up of the test environment
- Execution of the test
- Capture of the test responses
- Evaluation of the test responses
- Determination of a test case's pass/fail status
- Logging and reporting of the test results
- Test failures and how to handle (the retest effort can be substantial)

3.3.6 Measuring the Cost of Testing

Estimating the number of test cases, the preparation involved in planning these cases, the effort required to set up a test environment, and the work needed to execute the tests are fundamental steps in estimating the costs of testing. This estimate provides a baseline for improvement in both the estimating and testing processes. By measuring and tracking how much testing actually costs, you will be able to refine and improve these processes. This actual cost data will also be useful in justifying future testing effort. Direct testing costs as a percentage of total development often approach 25 percent. Indirect costs of testing – the result of poor planning and test execution – can sometimes be twice the direct costs. Examples of direct and indirect testing costs are shown in Table 3–1.

Table 3–1.
Direct and Indirect Costs of Testing

Direct Costs of Testing	Indirect Costs of Testing
Writing test plans Designing test scenarios and cases Preparing the Test environment Buying and developing tools Designing and documenting procedures Designing, preparing, and storing test data Renting facilities and information systems Support staff – often involving overtime Regression testing Analyzing results Reporting results	Rewriting programs Corrective action costs Rekeying data Failures Analysis meetings Debugging Unplanned retesting

3.4 Step 4: Prepare Test Environment and Tools

The ***test environment*** is the set of conditions, facilities, and tools that you may need to conduct your tests. You will almost certainly require some equipment, such as computers, printers, and internal and external network connections. You may also require some specific components or simulators and stimulators to mimic the actions of specific products or systems. Later, for Acceptance Testing, you will need access to the real environment. All phases of testing will run smoother if you plan, build, and manage your test environment properly.

As tests are developed, you should also determine where ***test tools*** may be required or may be beneficial. Decide what tools are needed, how they should be acquired (e.g., buy, build), and how long they will be used. Tools can help you:

- Analyze data
- Convert data formats
- Manage test data
- Manage test procedures
- Measure performance
- Automate test execution
- Stimulate an interface
- Simulate an interfacing system
- Capture test results

3.5 Step 5: Define Test Cases

Test cases were identified in the test specification. They are defined and documented in detail in this step. The level of detail is adjusted to fit the needs and resources of the project. The most complete definition would include a written description of the:

- Test case identifier and name
- Purpose
- Test configuration and special requirements
- Initial conditions
- Step-by-step procedures for test execution, data recording and analysis
- Test data
- Pass/fail criteria
- Updated requirements traceability matrix

The test procedures should be written at a level of detail so that the tester knows what to do. Writing down every single step, down to the exhaustive level of what buttons to push, is probably not worth the effort. Let the test team decide what level of detail they need. As a rule of thumb, there should be sufficient detail so that each re-execution follows the same steps.

You may build test data sets to support specific test cases. These data sets will typically include data inputs, configuration information, and expected results. You might also build large test data sets to support System Testing or Acceptance Testing by exercising the system over most of its capabilities.

Whichever type of data sets you create, ***save your test data.*** Constructing new test data every time a change is made takes too long, and these same data sets can be reused to retest the system after many changes are made. Also, save the output data. The outputs from subsequent tests can be compared to the original output data for differences. Test data sets should be managed so that they can be readily recalled and quickly related to the test cases that are to be run. Test data is useful throughout the system life cycle, even long after a system has been accepted and is supporting production operations. ***Regression testing*** refers to the type of test that is run on newly updated versions of software. It repeats earlier tests that have run successfully to make sure that no errors were inadvertently inserted as the change was made. Efficient regression testing depends on having a good set of prior test data and procedures readily available.

3.6 Step 6: Execute Tests

Executing the tests is meant to be a straightforward implementation of the written procedures. The tester should be familiar with the requirements of the system and the operational objectives, but need not be an expert in the products' internals. In principle, the tester's job is to execute the test and attribute pass/fail scores to the tests in accordance with the test criteria. In practice, the testers can be of great benefit to the development team. They will see system behavior that is not anticipated in the test plan and can inform the developers of unexpected problems that come up during testing. ***It is essential that the tester maintains a log and writes down everything that he***

or she does outside the written procedures. The notes that a tester makes during the tests can be invaluable to the developer in locating, recreating, and correcting problems.

Some test procedures can be automated. The choice between manual and automated procedures depends on two factors: (1) does the procedure conform easily to automation and (2) will the procedure be executed frequently enough to make the effort of automation cost effective. If you have many test cases to run and you can – with reasonable effort – automate those cases, then automation can save a lot of tedious work.

3.7 Step 7: Analyze and Summarize Results

Many of the tests that are performed are pass/fail tests; i.e., either the system executes a function properly or it does not. Using the requirements traceability matrix, you will be able to check off many of the required functions of the system as you produce successful test results. Some requirements, however, are quantitative, and in these cases it is also important to measure how well a system performs. Using the test data you gather together with the metrics for system performance you developed during the requirements phase of your project, you can determine if your system is performing within the required tolerance. For example, suppose your system includes a hand held device with a required battery life of 24 hours. Measurements may indicate an actual battery life of only 22 hours in continuous use when the outside temperature is below 40 degrees. Obviously, this is not strictly a pass/fail situation. You have to decide whether or not that performance is acceptable. In addition, quantitative data can help focus your attention on the solution to specific performance problems. For example, perhaps what is needed in the previous example is a battery that is less susceptible to temperature.

In some cases, it is necessary to process data to obtain results that can be compared to requirements metrics. In these cases, an analysis plan should specify the processing (conversions and model predictions) that is necessary. Pay careful attention to a consistent set of units for all measurements.

Finally, you must summarize the test results for presentation to management. Visual aids, such as red-yellow-green charts of the requirements matrix, can help focus necessary attention on the problems. Managers will want to know the answers to three key questions:

- Where are there problems?
- What is the impact of each problem?
- What is required to fix the problem (in time and cost)?

Prepare your summary to answer these questions.

4. UNIQUE CHARACTERISTICS OF EACH TYPE OF TESTING

Up to this point, we have discussed five test types (integration, system test, acceptance test, system-of-systems test, interoperability test) in generic terms. Chapter 3 discussed the steps that were common to all types of tests. In this chapter, we will discuss the unique features of each of the five types individually.

Figure 4–1 shows your state CVISN systems in a general context. In this illustration, the integrated Products A through D comprise one of your CVISN systems (e.g., safety). These products exchange information with each other across interfaces that you have designed and documented. Products C and D also exchange information across standardized interfaces with an external system. These interfaces are controlled by the CVISN Architecture.

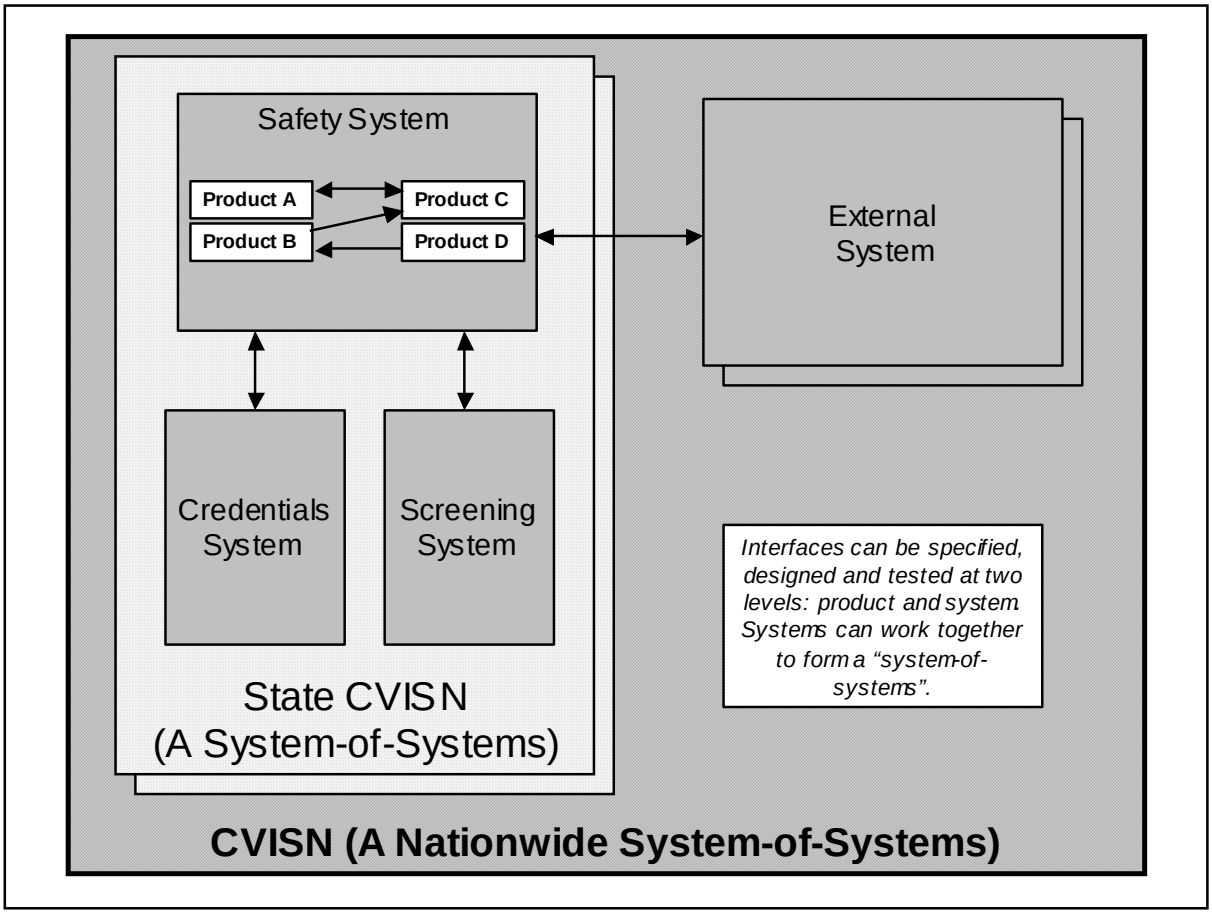

Figure 4–1. State CVISN Systems

To simplify the discussion, we will talk about testing only at the system level and system-of-systems levels (please refer to the "V" diagram in Figure 2–2). The same type of discussion applies at the product level. We discuss this from the point of view of a typical state, but the same principles apply to a motor carrier or developer of a clearinghouse. The diagram in Figure 4–1 illustrates the environments and interfaces you must test.

1. Within your system, you will integrate the products that comprise your system and perform integration testing of the interfaces between products.
2. Within your enterprise, you will perform system testing. System testing consists of two steps: First, you test the interfaces between your system and any other systems within your enterprise with which your system interacts. Second, test that each system performs all its intended functions and meets all performance and "ilities" goals.
3. For contractual reasons, you may be required also to conduct Acceptance Tests at the system level.
4. Considering CVISN as a whole, you will conduct system-of-systems test and evaluation. Everything covered by these tests should have already been covered at a lower level of test. However, this will ensure that everything does in fact work together in an operational environment. In addition, the evaluation effort will measure whether the system achieves the anticipated result. In other words, once testing has demonstrated that the system does what was intended, evaluation examines whether the working system actually provides the benefit anticipated.
5. **Interoperability tests are, in fact, special cases of other test types.** Pair-wise interoperability tests of standardized interfaces between your products and other CVISN products are the external interface part of system testing. The end-to-end interoperability tests are a part of system-of-systems testing, including, when possible, interoperability with other CVISN systems external to your state.

4.1 Integration Testing

4.1.1 Alternative Approaches

System-level Integration Testing is where we begin to combine products to test their interfaces and to confirm that the products link together properly. By its nature, Integration Testing proceeds in steps, but there is no predetermined formula to dictate the order of these steps. Instead, your approach to Integration Testing depends on the project and at what level you are integrating the products.

Hetzel [1] lists six alternative strategies for Integration Testing. Each of these is frequently used in practice. (Hetzel uses the term "modules" in his discussion, but the alternatives are equally appropriate for products.)

1. Top-level modules – Start with the driver and command or top-level modules and then work down by "plugging in" the additional modules.
2. Critical modules – Start with the critical system modules and integrate them, then add the rest of the skeleton around them.
3. Bottom-level modules – Start with the individual programs as they complete unit testing and integrate by working up to build bigger and bigger pieces.
4. Functional modules – Select a specific function and integrate the modules needed for that function, then proceed to the next function, and so on.
5. As-available modules – Take the modules that are ready and fit them together as much as possible.
6. Complete skeleton – Integrate all the modules in the skeleton at once and hold off any Integration Testing until all are interfaced.

There are many factors that must be evaluated in determining the best alternative for your project, and your best approach may be to select combinations of these to support integration. Some factors to consider are:

- How much "extra code" must be written to support the testing process?
- How are products meeting the project schedule (test what you can)?
- Where is system reliability most needed?

Integration Testing often fails because ***it lacks formality in planning and execution.*** You must plan the approach to Integration Testing carefully. Ad-hoc Integration Testing will be chaotic and frustrating and will result in inefficient test effort and inadequate test coverage. By contrast, well-planned Integration Testing will result in a fully tested set of integrated products that are stable and ready for System Testing.

4.1.2 Basic Steps in Interface Testing

The steps involved in testing the interfaces between products are outlined below:

1. Identify each pair of products that interact.
2. Review interface specifications.
3. Design test procedures that test nominal interaction between the products.
4. Verify that each product has been successfully tested in stand-alone mode and can generate or use the data intended for the product interface testing.
5. Connect the products.
6. Execute the test procedures and exchange the pre-defined data between products, for example from Product A to Product C illustrated in Figure 4–1.
7. Capture the output of Product A and verify that the outputs match the interface specification.
8. Capture the input to Product C to verify (a) that the transmission mechanism did not distort the data and (b) that the inputs match the interface specification.
9. Verify that the receiving product, Product C, interprets the inputs correctly. It may be possible to process the data and review normal Product C outputs to determine whether inputs from Product A were interpreted correctly.
10. To establish reliability, repeat these procedures, covering a range of data possibilities, including error conditions, nominal data, and extraordinary data. In particular, look for cases where communication between interfaces "break" to identify areas in which the interfaces need to be more robust.

4.2 System Testing

The objectives of System Testing are to demonstrate that all functions are working as required, that the system is reliable, and that the performance and quality of the system meet expectations, even under heavy load or error conditions. Carefully planned tests in these areas can really reveal how well the system will hold up operationally. Planning for System Testing begins at the requirements phase of the life cycle.

A significant benefit of planning for and conducting System Testing is generation of a good set of test cases and test data that really drive the system through its paces. This is especially useful for retesting the system as changes are made, and these data sets should be managed and retained.

In contrast to Integration Testing, System Testing is viewed externally and from a distance. The system tester looks at an entire assembled system, rather than single products, and cannot be consumed by the details of individual products. These lower level products must work smoothly; otherwise, System Testing becomes a nightmare and a waste of time. This guide assumes that developers and modifiers will test each system's internals.

In the most general case, the systems you develop have to interact with systems you already own. They may also interact with external CVISN systems, which is the topic of Interoperability

Testing. When systems are integrated with each other, testing is performed in two major steps: (1) system-to-system interface testing and (2) end-to-end function testing.

4.2.1 System Interface Testing

System interface testing addresses the interaction of two systems that exchange data. As stated repeatedly above, each product should be fully tested before system interface testing occurs. System developers should help plan the tests to be sure that various paths in the system design are exercised. It may be necessary for system developers to support the testing process, especially to interpret data inputs and outputs. The testing process and data, however, must be controlled by the tester and/or the test environment. In general, products should be tested with as little modification as possible from the proposed production module. It may be necessary to insert special test code into the systems under test to verify inputs and outputs.

The steps involved in system interface testing parallel those used in interface testing described in the preceding section on Integration Testing.

4.2.2 Function Testing

Function testing addresses a complete user function, normally one that involves several products. Each pair of products should have been successfully integrated and tested before end-to-end function testing occurs. Again, the tester controls the data and process. To the degree possible, a function test should be conducted in the real operating environment, with no special test code inserted into the systems under test. Actual system users should help plan and, if possible, perform the tests.

The steps involved in end-to-end testing are outlined below:

1. Identify the functions that require testing.
2. Identify the products involved.
3. Verify that the products have been tested in pairs (see Section 4.2.1).
4. Map out scenarios that exercise each function in normal operational configurations and predict the results based on the test data to be used.
5. Generate data and run the scenarios using standard operating practices, if possible. Verify that each system receives the data expected from the other systems, that the data are interpreted correctly, and that the correct results are generated.
6. Verify that no unintentional changes to the data occur and that conversions/translations performed in each system yield accurate results

Repeat these tests, covering a range of data possibilities, including error conditions in various systems and user inputs, nominal data, and extraordinary data.

4.2.3 Performance and "ilities" Testing

In addition to testing that each function works, there is usually a need to test that the system meets performance, reliability, useability and other "ility" requirements. For example, the system must carry out its tasks within specified time limits, handle the maximum specified processing load, and operate with required reliability when a communication channel fails. Some of these tests may be combined with functional tests, but others will need to be separate test cases.

4.3 Acceptance Testing

Before you conduct Acceptance Testing, System Testing should have demonstrated that the system is reliable, stable, and functional. You should have measured most of the capabilities and corrected any important defects. User involvement in Acceptance Testing is mandatory. Sometimes, organizations will combine System Testing and Acceptance Testing as one joint test; but there are important differences as listed in Table 4–1 below [11].

Table 4–1.
Comparison of System Testing and Acceptance Testing

System Testing	Acceptance Testing
Performed by systems organization	Performed by users, auditors, an Independent Verification and Validation (IV&V) agent, or a Quality Assurance (QA) organization
Destructive, i.e., designed to be thorough and find all defects.	Demonstrative rather than destructive i.e., emphasis on demonstrating that system works
Oriented to the systems technical issues	Oriented to the business fit of the system to the organization
Scope excludes the end-user	Scope usually includes checking end-user preparation and readiness for system installation, data base conversion, performance, etc.
Almost always performed prior to conclusion of a contract	Performed only about 20% of the time (the rest of the time, the acceptance test is simply rolled in with the system test)
Checks satisfaction of stated requirements	Checks acceptability by the end-user prior to actual business use

4.4 System-of-Systems Test & Evaluation

System-of-systems test is very different from the lower level testing. This type of testing is not generally discussed in the literature on testing. Most testing is focused on how to prove that a specific product or system does what it was supposed to in the way that was required or intended. A system has a well-defined scope and accomplishes a set of functions relatively independently. A system-of-systems brings together multiple systems that each carry out some independent function. A system-of-systems accomplishes some larger function that spans multiple systems.

CVISN is a system-of-systems. Each of its major systems (safety, screening, credentials) accomplishes a significant function in a stand-alone sense. However, working together these three systems can accomplish some functions with an efficiency and effectiveness that cannot be achieved independently. For example, a screening system can do a better job of sorting vehicles if it has near-real-time registration data obtained directly from a credentialing system or out-of-service data obtained from a safety system.

4.4.1 End-to-End Testing

Testing at the system-of-systems level assumes that each system and each interface between systems has been thoroughly tested in previous system level testing. Therefore, the only thing remaining to do is what we have referred to as end-to-end test. These are tests of functions that inherently require multiple systems to carry out parts of the overall function. For example, if we want to ensure that a snapshot arrives at the roadside within one hour after a vehicle has been registered, we need an end-to-end (i.e., system-of-systems) test. Many products and systems have to work properly for this result to be obtained. If all the systems and interfaces have been tested independently, this should work. However, there is no substitute for testing as a final proof that in fact all the parts fit together properly.

System-of-systems testing cannot be comprehensive and test every possible combination of every function in every system. It should focus on a relatively few key scenarios. Choosing which scenarios to use is a matter of judgment. Some factors to consider are:

- Demonstrate that the core functions work well
- Test new and innovative capabilities
- Test risky elements
- Test for end-to-end performance
- Test to ensure data integrity under error conditions or degraded modes of operation

4.4.2 Systems Evaluation

A second aspect of system-of-systems testing is overall evaluation of the system. This activity is done after other tests have proven that the system fundamentally operates as intended. Evaluation then focuses on the question: “Does the system-of-systems produce the result that was desired?” For example, the result desired of an e-screening system is that safe and legal trucks get to bypass weigh stations and trucks likely to have problems are pulled in. Evaluation

would try to determine whether or not this result was achieved. This usually requires a special data collection capability, often built into the operational system. Rather than controlled test scenarios, data and procedures, the evaluation is usually done under normal system operating conditions. Measures of performance and measures of effectiveness must be defined. Baseline data must be collected before the systems are put into place. Data are collected after the system goes operational and are compared against the baseline. Finally, an evaluation report is published which documents the net improvement achieved by the new systems.

We have discussed evaluation at the system-of-systems level. Evaluation could be done at the system level also. However, in the CVISN context, we believe that the ultimate results or benefits are more usefully evaluated by considering the impact of CVISN as a whole versus individual systems comprising CVISN.

4.5 Interoperability Testing

Interoperability refers to the ability of a CVISN system to exchange information with other CVISN systems, and to work with other CVISN systems to accomplish shared functions. Interoperability testing is a term created specifically for the CVISN Program. It is not actually a distinct type of test in a general sense. Rather, it is a selected subset of tests gathered from the product, system and system-of-systems levels testing. We pulled these tests out and gave them a separate name as a matter of management convenience. These tests are special because they can be defined once and executed by multiple states. The purposes of treating them separately are:

- To gain the efficiency of sharing them among states
- To ensure that each state has sufficiently tested for interoperability.

The “V” diagram of Figure 2–2 shows the interoperability tests off to the side of the “V”, with arrows connecting to several types of traditional testing. This is intended to convey that these tests are packaged separately but are in fact run along with or as part of the traditional types and levels of testing.

There are two ways that you will test interoperability. (1) First, you will test the ability of products within your system to properly exchange information with products in other CVISN systems through **standardized interfaces**. We call this pair-wise interoperability. (2) Second, you will test the overall interoperability of your system with other CVISN systems in end-to-end testing. The relationship between these interoperability tests is illustrated in Figure 4–2.

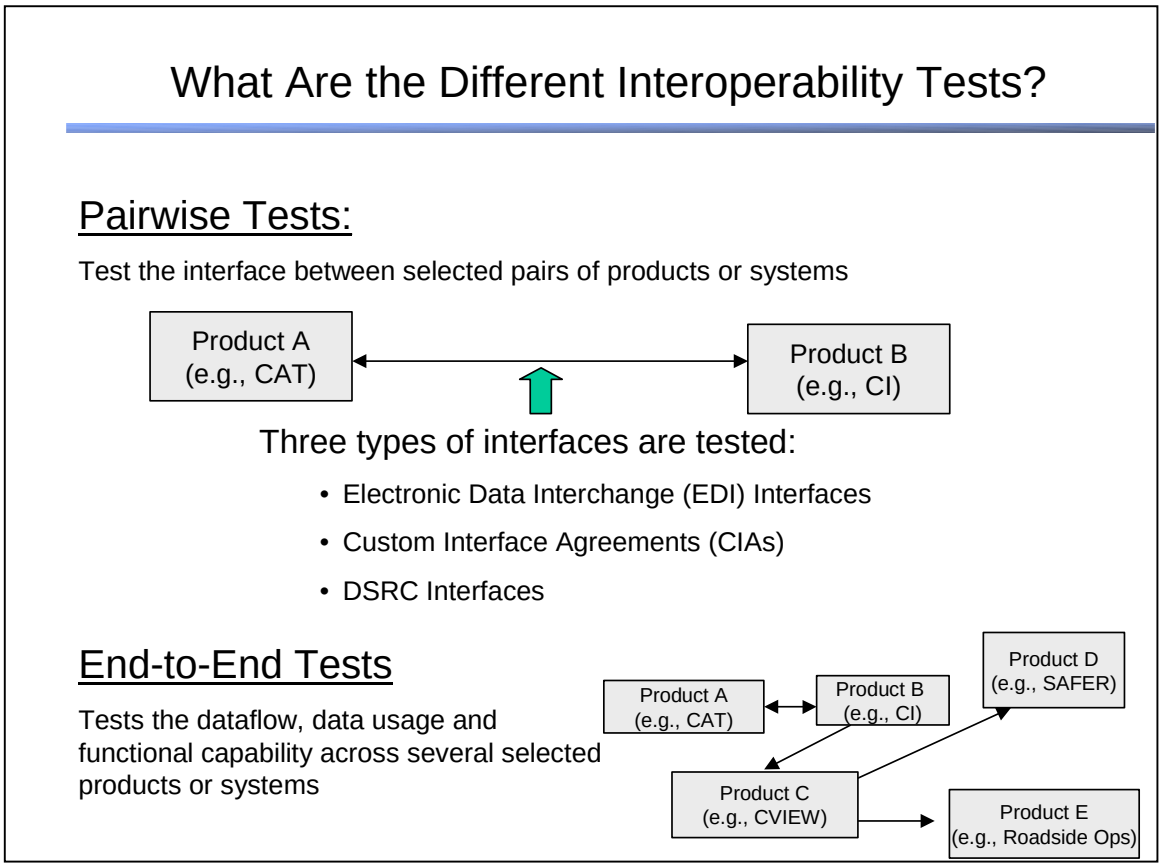

Figure 4–2. Interoperability Test Types

The purpose of pair-wise Interoperability Testing is to ensure that your system will interoperate properly within the larger CVISN domain by testing the controlled interfaces between products. The "CVISN Operational and Architectural Compatibility Handbook" [12-16] provides a comprehensive checklist of what is required to be compatible with the CVISN operational concepts, architecture, and standardized interfaces. In particular, these references provide the Interoperability Test Criteria that your CVISN system must satisfy to achieve interoperability with credentials administration, electronic screening, and safety information exchange. These criteria apply for both pair-wise interface operability and end-to-end operability. Interoperability tests should be conducted throughout the Integration and Test phase. They should be done early and repeated as necessary.

The CVISN Test Facility at APL (Figure 4–3) is equipped and staffed to support initial and on-going interoperability testing activities. Using this facility, testers verify that products claiming to be compatible with standard interfaces are actually compliant.

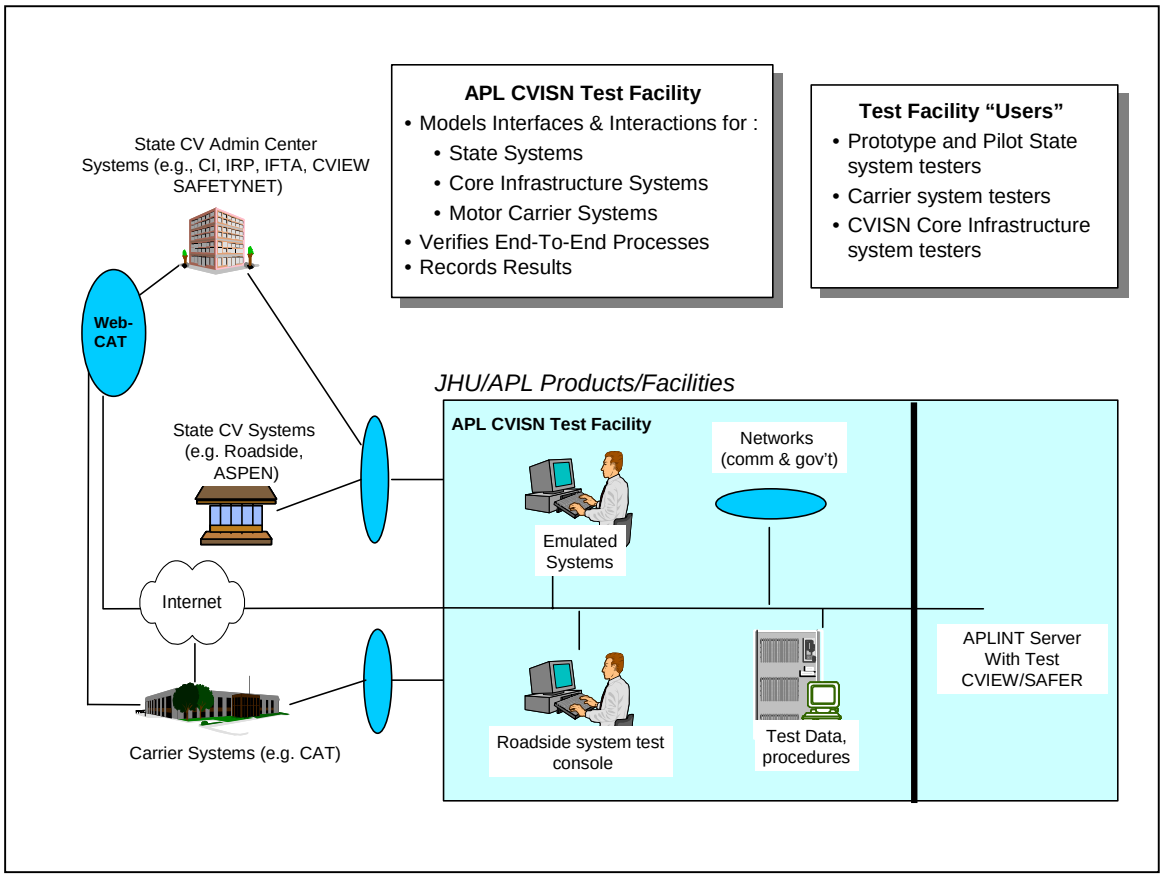

Figure 4–3. APL CVISN Test Facility

The facility allows system developers and testers to model interfaces and interactions for state systems, core infrastructure systems, and motor carrier systems. Potential users of the CVISN Test Facility include CVISN deployment states; carriers; CVISN core infrastructure system developers; EDI developers, and DSRC developers.

The facility is made up of computers, networks, translators, DBMS's, select CVO prototype systems and other products. Specific capabilities include the following products:

- APL Roadside Operations Computer (ROC) – used to receive snapshots from SAFER and CVIEW (and query for snapshots to verify results)
- ASPEN Inspection reporting – Used to send inspection reports to SAFER and SAFETYNET (and to CVIEW with ASPEN32)
- APL CVIEW system – used to provide snapshots to Roadside Systems
- Oracle DBMS – for database support
- PC Anywhere for remote connectivity
- WWW access/browsers

Interoperability testing should start with data prepared by APL for use in the prototype states. This data consists of inputs and expected outputs; database entries for MCMIS, SAFER, etc.; data entry field information; transponder messages; and EDI messages. This data is documented in Part 4 of the Interoperability Test Suite [9]. The data is stored and is available on request.

The test facility environment is controlled and provides access to multiple networks and the Internet. To use the facility, a user needs to establish a user account, schedule testing time, and ensure that the necessary applications are in place to conduct testing.

For a full discussion of CVISN Interoperability Testing, please refer to The "CVISN Operational and Architectural Compatibility Handbook, Part 5, Interoperability Test Criteria" [16], "ITS/CVO Interoperability Test Suite Package Parts 1 to 4" [6 to 9], and "ITS/CVO Architecture Conformance: Interoperability Testing Strategy" [10].

This Page Intentionally Blank

5. HOW DO YOU KNOW WHEN TO STOP TESTING?

When you begin any test, it is important to know up front what will constitute successful completion of the test. These success criteria are established during test planning, and if the success criteria are met for any given test, then that test case is logged as passed. Collard [11] suggests some important indicators to help determine when you should stop testing:

- Passes have been logged for all test cases in the detailed test plan, or, if some test cases have not yet passed, their consequences are considered minor.
- The important defects discovered during testing – defined as those that you decided had to be fixed before system release – have been fixed, retested, and logged as passed.
- All defects that have been found but not been fixed have been reviewed to confirm they are minor.
- Test productivity falls off; i.e., no or very few new errors are being found, despite considerable on-going test effort. Some organizations use (as a pre-defined, numeric cut-off threshold), the number of weighted defects found per additional 100 hours of testing.
- The number of defects found and fixed meets a pre-defined goal, which was set based on the predicted total number of defects.

The duration and scope of testing is a tradeoff between acceptable risk and cost, as characterized in Figure 5–1 [2] below.

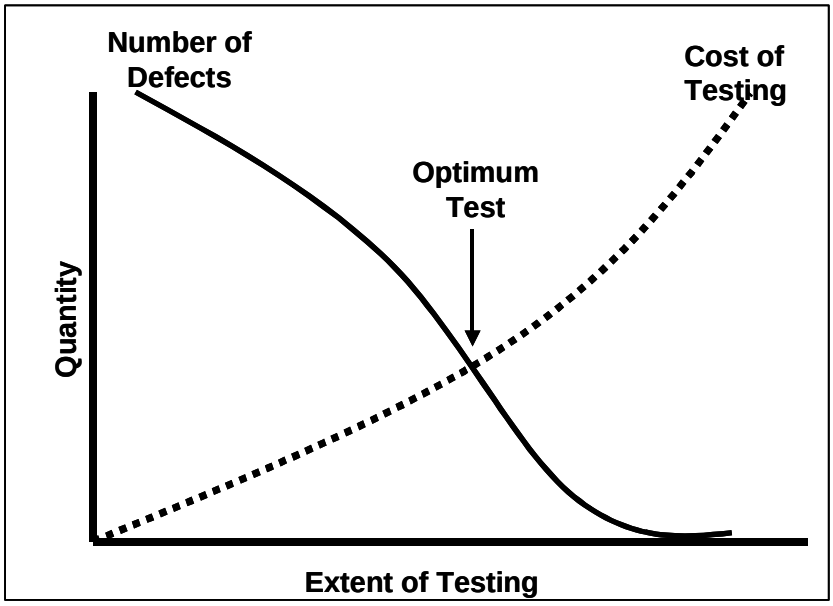

Figure 5–1. Duration and Scope of Testing

This Page Intentionally Blank

6. TEST DOCUMENTATION

Throughout this guide we have discussed the five types of tests encompassed by Integration and Test. Each test type will require some level of documentation. The exact structure and scope of your test documentation will vary based on your needs and resources. An illustrative example of a comprehensive set of documentation is shown in Figure 6-1. As the figure indicates, each type of test could be supported by four different types of documents:

- Test Plan
- Test Specifications
- Test Cases
- Test Reports

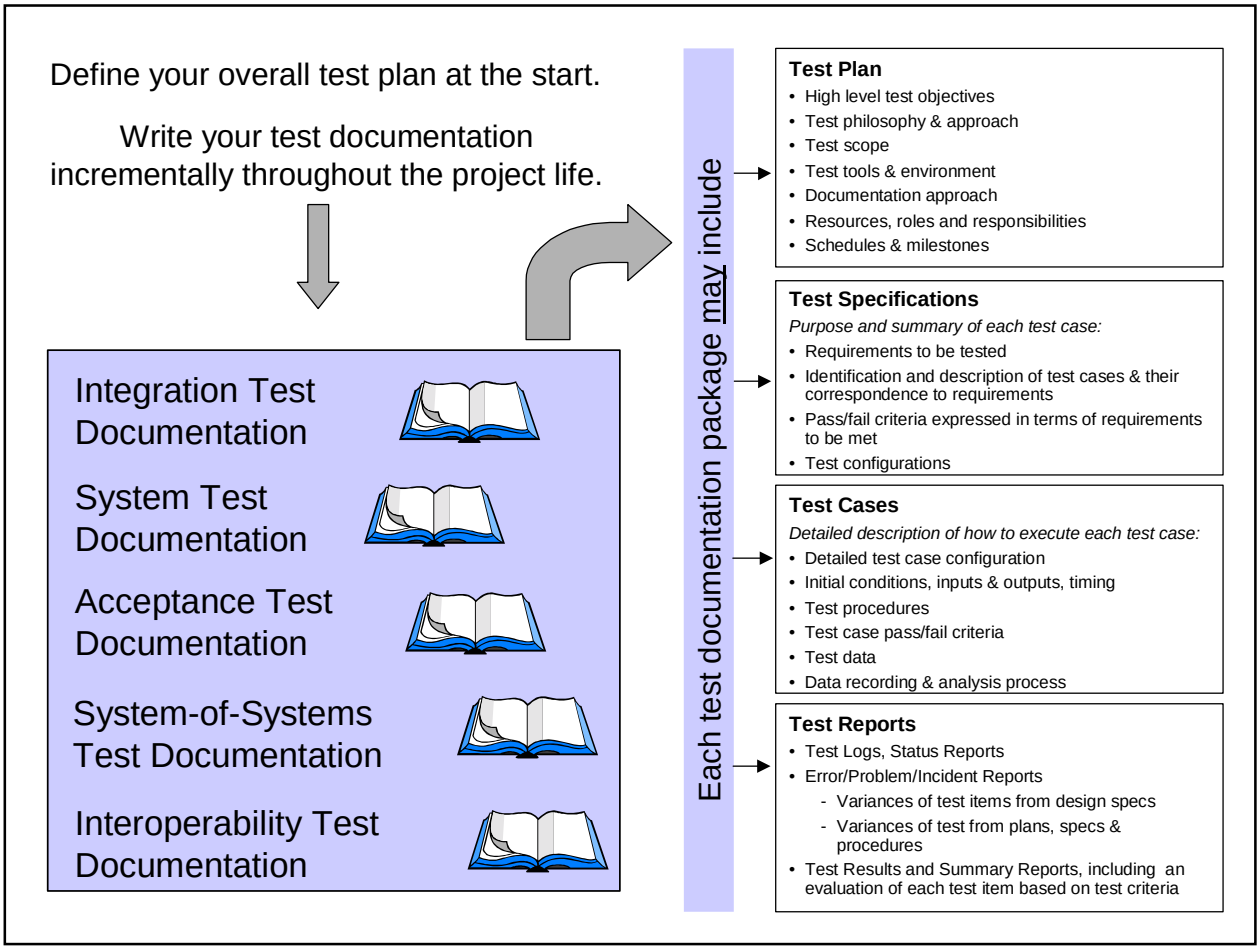

Figure 6–1. Framework for a Comprehensive Set of Test Documentation

It is unlikely that any state needs (or could afford!) to produce all the documents illustrated in the figure. As you develop your overall test strategy, you should define your documentation approach. At a minimum, you should have a master test plan, detailed test logs and good configuration management records. Some examples of how the framework shown in the figure could be adapted to your needs are listed below:

- Package all your test planning information into one overall Master Test Plan
- Combine two or more types of tests into one physical document. For example, Acceptance Test may be simply a subset of System Test and packaged as a section of the System Test Plan.
- Do not formally document detailed procedures. Use knowledgeable testers who are familiar with the system and user documentation to execute tests based on the documented test specifications.
- Produce the test report as a summary of System Problem Reports or Change Requests. Minimize the effort involved.

The "CVISN Operational and Architectural Compatibility Handbook, Part 5" [16], "ITS/CVO Interoperability Test Suite Package Parts 1 to 4" [6 to 9], and "ITS/CVO Architecture Conformance: Interoperability Testing Strategy" [10] provide most of what you will need for the Interoperability Test Documentation. These documents should be your starting point for interoperability testing. In fact, Interoperability Tests are just a special case of System Tests and System-of-Systems Test. Therefore, you may find that the Interoperability Test Documentation provides a good template for all of your system test plans. You, will of course, need to tailor the actual tests to your particular state's design and the stage of testing. Your system will probably not require all of the test cases that are described in these references, since the pair-wise tests are intended to verify specific standardized interfaces and your design may not use all those interface standards. In addition, the amount of detail included in this and the other plans is up to you. Your plans should be consistent in detail and scope with the magnitude, risk, complexity, and cost of your project. Complex projects with high risk and a large budget require careful planning, and your test plans should reflect this. By contrast, simple projects with low risk and small budgets can often get by with simple test plans.

Test planning, preparation, and documentation often receive less care and attention than they deserve. Test planning – and the preparation and documentation that goes along with planning – is critical to building solid, working systems. Test planning should start at the beginning of the project and be thoroughly integrated with the phases of development. Think of testing as all of the activities that are done during a project to inspect, verify, and assess the progress of the system being developed. Life cycle testing is specifically designed to capture errors as early as possible in the development cycle. Since most errors, perhaps as many as 60%, can be traced to the design, capturing these design defects early significantly reduces overall cost. Also, since testing represents a significant expenditure in time and effort, proper test planning will improve the efficiency and lower the cost of the testing effort.

Detailed guidance for test documentation outlines and content can be found in the IEEE Standard 829 [5], (IEEE Standard for Software Test Documentation).

APPENDIX A.

REFERENCES

This Page Intentionally Blank

REFERENCES

1. Hetzel, Bill, *The Complete Guide to SOFTWARE TESTING*, Second Edition, 1988, John Wiley & Sons.

2. Perry, William, *Effective Methods for Software Testing*, 1995, John Wiley & Sons.

3. Myers, G. J., *The Art of Software Testing*, 1979, John Wiley & Sons.

4. Beizer, B., *Software System Testing and Quality Assurance*, 1984, Van Nostrand.

5. IEEE Standard 829-1983 (reaffirmed September 1991), *Software Standard Test Documentation*, IEEE, Inc.

6. JHU/APL, ITS/CVO Interoperability Test Suite Package, *Introduction and Part 1, Test Specifications*, POR–98–7122, D.2, dated January 2000.

7. JHU/APL, ITS/CVO Interoperability Test Suite Package, *Part 2, Test Cases and Procedures*, POR–98–7123, D.2, dated December 2000.

8. JHU/APL, ITS/CVO Interoperability Test Suite Package, *Part 3, Test Tool Description*, POR–98–7124, D.1, dated July 1999.

9. JHU/APL, ITS/CVO Interoperability Test Suite Package, *Part 4, Test Data*, POR–98–7125, D.0, dated June 1998.

10. JHU/APL, *ITS/CVO Architecture Conformance: Interoperability Testing Strategy*, POR–98–7076 D.1, dated January 1998.

11. Collard & Company, *Systems Testing & Quality Assurance Techniques*, 1999, seminar from Advanced Information Technologies.

12. JHU/APL, *CVISN Operational and Architectural Compatibility Handbook (COACH) Part 1 – Operational Concept and Top-Level Design Checklists*, POR–97–7067 V2.0, August 2000, http://www.jhuapl.edu/cvisn.

13. JHU/APL, *CVISN Operational and Architectural Compatibility Handbook (COACH) Part 2 – Project Management Checklists*, POR–97–7067 P2.0, September 1999, [Note: The latest version will be available on the JHU/APL and CVISN Web Site http://www.jhuapl.edu/cvisn/.]

14. JHU/APL, *CVISN Operational and Architectural Compatibility Handbook (COACH) Part 3 – Detailed System Checklists*, POR–97–7067 V1.0, October 2000.

15. JHU/APL, *CVISN Operational and Architectural Compatibility Handbook (COACH) Part 4 – Interface Specification Checklists*, POR–97–7067 P2.0, October 2000.

16. JHU/APL, *CVISN Operational and Architectural Compatibility Handbook (COACH) Part 5 – Interoperability Test Criteria*, POR–97–7067 D1.0, July 1999.

17. Boehm, Barry W., *Software Engineering Economics*, 1982, Prentice Hall PTR.

18. Freedman, Daniel and Weinberg, Gerald, *Walkthroughs, Inspections, and Technical Reviews*, 3rd ed., (Boston: Little, Brown, 1977).

19. JHU/APL, ITS/CVO CVISN Glossary, POR–96–6997 V2.0, December 2000.

20. IEEE-1220-1998, IEEE Trial Use Standard for Application and Management of the Systems Engineering Process.