

# **SFA System Integration & Testing Approach**

## **SFA Modernization Program**

*US Department*

## Table of Contents

<b>1. Introduction</b>	<b>1</b>
<b>1.1. Purpose</b>	<b>1</b>
<b>1.2. Scope</b>	<b>2</b>
<b>1.3. Current State of System Integration and Testing</b>	<b>3</b>
<b>1.4. Document Organization</b>	<b>5</b>
<b>1.5. Document Development Process</b>	<b>5</b>
<b>2. System Integration and Testing Design</b>	<b>7</b>
<b>2.1. System Integration &amp; Testing Strategy</b>	<b>7</b>
2.1.1. Key Management Concepts	7
2.1.1.1. V – Model	7
2.1.1.2. Phase Containment	11
2.1.1.3. Metrics	11
2.1.1.4. Test Data Management	11
2.1.1.5. Version Control	11
2.1.2. System Integration and Testing Stages	11
2.1.2.1. Unit Test	13
2.1.2.2. Integration Test	19
2.1.2.3. Performance Test	24
2.1.2.4. User Acceptance Test	30
2.1.3. Testing Success Factors	35
2.1.3.1. Test Planning	35
2.1.3.2. Communication	35
2.1.3.3. Metrics	36
2.1.3.4. Managing the Testing Environment	37
2.1.3.5. Risk Management	39
2.1.3.6. Test Problem/Observation Logs Fix-It Process	39
<b>2.2. Organization and Responsibilities</b>	<b>40</b>
<b>2.3. Relationship to Other Processes</b>	<b>42</b>
<b>3. Next Steps and Implementation Plan</b>	<b>43</b>
<b>3.1. Next Steps</b>	<b>43</b>
3.1.1. Identify the Implementers of Enterprise System Integration & Testing	43
3.1.2. Define Scope of Implementation and Timeline	43
<b>3.2. Implementation Plan</b>	<b>44</b>
3.2.1. Mobilize Implementation Team	44
3.2.2. Prepare System Integration & Testing Environment	44
3.2.2.1. Establish Testing Environment(s)	44
3.2.2.2. Certify Environment	44
3.2.2.3. Support Environment	45
3.2.2.4. Training	45
3.2.3. Establish Metrics	46
3.2.3.1. How?	46

---

3.2.3.2.	Identify Metrics	46
3.2.3.3.	Define Purpose of each Metric	46
3.2.3.4.	Define how each Metric will be Collected, Analyzed and Reported	46
3.2.3.5.	Develop Process for Taking Corrective Action and Evaluating	47
3.2.3.6.	Develop Communication Methodology	47
3.2.4.	Define Test Resource Requirements	47
3.2.5.	Prepare Testing Environment(s)	47
3.2.5.1.	Define Test Environment Requirements	48
3.2.5.2.	Define Test Entry and Exit Criteria	48
3.2.6.	Test Script Generation and Scenario Planning	50
3.2.6.1.	Define High-level Test Conditions	50
3.2.6.2.	Cross Reference Test Conditions to Requirements Specifications	51
3.2.6.3.	Develop Product Test Cycles	51
3.2.6.4.	Group Test Conditions into Cycles	52
<b>4.</b>	<b>APPENDIX</b>	<b>53</b>
<b>4.1.</b>	<b>Testing Phases</b>	<b>53</b>
<b>4.2.</b>	<b>Incident Reporting</b>	<b>62</b>
<b>4.3.</b>	<b>Problem Detection</b>	<b>68</b>
<b>4.4.</b>	<b>Candidate Checklists and Forms</b>	<b>84</b>

## **1. Introduction**

### **1.1. Purpose**

In order for SFA to accomplish the objectives of a performance based organization, it will require reengineering of their technical processes and architecture. This level of activity will require a significant amount of effort to coordinate, monitor progress, track updates and validate the technical enhancements. To accomplish this, SFA requires a support structure to provide the oversight and coordination of capability release activities (to the SFA executive team and stakeholders). This is required so the right decisions can be made to achieve performance objectives related to planned application and/or system capability releases.

This approach identifies the need for an enterprise system integration and testing function that will maintain focus on the overall technical and functional objectives of the program. This enterprise system integration and testing function will also provide the continuous guidance needed to support the delivery of SFA's targeted business capabilities throughout the life of SFA Modernization Blueprint. Through rigorous system integration and testing criteria, SFA can maintain the integrity and quality of the system components being developed, revised, integrated and/or maintained. If system integration and testing is executed poorly it can result in (1) solutions which do not meet the Modernization Blueprint requirements, (2) incorrectly constructed / malfunctioning solutions, and (3) lost work, hence cost and schedule overrun. This approach will provide the model and guidance for successful integration and testing of products developed for the Student Financial Administration.

To develop quality applications, it is essential for project teams to follow a well defined testing strategy. The objective of the overall testing strategy is to ensure production ready, bug-free, quality applications; and to complete as much testing as possible early in the development life cycle.

The purpose of this document is to layout the testing strategy for SFA projects and to define each of the testing stages. This document will provide a structured testing framework throughout a project's development life cycle. The concepts discussed in this document will focus on laying a testing infrastructure and include a set of proven development processes and the architecture components required to support these processes.

As stated, these processes are intended to ensure SFA delivers quality, bug free application systems to their customers/clients. Adherence to these structured testing techniques should provide the following benefits:

- Ensures the application meets client quality expectations, as well as, SFA quality metrics

- Enhances SFA's reputation; SFA will be viewed as capable systems developers who consistently deliver quality in a timely manner
- Assists with maintaining schedules through structured format
- Improve relationships with customers; the clients will actively participate in the testing activities and develop first-hand knowledge of SFA's adherence to quality techniques
- Develop reusable testing materials; provides regression test plans; jump starts subsequent testing efforts
- Increases customer comfort and acceptance of applications by actually involving them in the development/test process; places responsibility for approving testing success on the customer
- Decreases development costs by introducing phase containment through testing segment entry and exit criteria

Detailed plans for testing will not be included as part of this documentation. The actual test conditions, scenarios, test data, and expected results will be developed during the test planning and preparation segments of an SFA project

Upon the conclusion of reading this approach, the reader will understand the necessity of creating an enterprise focused system integration and testing organization. To jump start this process, the reader has been supplied a series of next steps outlined at the conclusion of this approach. These steps will provide the reader with the requirements for implementing the organizational structure and the associated enterprise system integration and testing concepts.

## **1.2. Scope**

The system integration and testing approach applies to all information systems and related system engineering activities that might affect the achievement of the SFA Modernization effort. This would include hardware, software (COTS and/or custom), and documentation. In particular, the focus of this document is on the enterprise perspective of system integration and testing.

Some key system integration and testing objectives of this approach are:

- Test that the system enhancements and modifications meet the business requirements
- Perform a component/systems/product test of modifications
- Test and evaluate (validation/regression) whether the system is inadvertently impacted by modifications
- Test the system efficiently as possible, applying reusability wherever applicable
- Build business expertise within the testing team to support effective testing

### **1.3. Current State of System Integration and Testing**

The objective of testing is to ensure that delivered SFA systems satisfy their defined functional, technical and quality requirements. The testing process should include all activities required to conduct thorough and accurate tests of system parameters, customizations, interface modules, and business processes. The permeating philosophy should be to build testing into the development process, rather than make testing the last step before production. This philosophy should extend beyond traditional testing approaches by ensuring that all major deliverables completed during the SFA system implementation project, will be verified and validated throughout each stage of development. It is the lack of this consistent philosophy being followed at SFA, that signals the need for this system integration and testing approach.

A current state analysis was performed and those results have been summarized in the two representative examples. The current states depicted below, supports the suggestion that a standardized approach with defined outcomes or milestones, can provide SFA with a set of repeatable processes. Not only provide them but, provide a means for measuring them.

Following are two examples of the typical testing methodology being followed by the SFA eCommerce application development teams. These examples accentuate the need for consistency and repeatable system integration processes.

#### EDEExpress/EDEExpress Suite

The EDEExpress/EDEExpress systems development team produced five documents in the support of requirements management and development. The requirements began with the previous release as the baseline due to the annual release schedule. For example, year group software 1999-2000, is used to baseline year group software 2000-2001. Inputs for requirements come from three areas:

- Legislation (i.e. governmental requirements from Capital Hill, budgeting efforts in Congress, etc.)
- Configuration Management (i.e. software bugs)
- User group/focus group (i.e. state colleges, combination schools, 3<sup>rd</sup> party services, trade schools, etc.)

Contractors begin development once the technical specifications are drafted. At the completion of the technical specification for a module, the contractors will perform unit testing. Once completed, a version is released to SFA and passed onto the SFA customer service representatives to test. The customer service representative test provides a user perspective. There is no specified test scenario utilized during this step. A contractor, Macro International, performs the systems integration test. Macro International puts together a systems test plan, test procedures, and matrices package. The package is reviewed and inventoried by SFA. All shortfalls are fixed and

the plan is started. Macro International shares test data sets and files with SFA for SFA's acceptance testing. Macro International's test ensures that the customer requirements document requirements are met. SFA's acceptance test coincides with the beta test. The beta test allows 6-9 schools to work with the new software application. If any problems are discovered via the customer service representative test, Macro test, SFA acceptance test, or the beta test, the issues are logged into the reporting tool. A risk analysis is performed on the issue to determine if the issue will be fixed or documented on the requirement tracking summary for the next release.

Regression testing is then started to test every fix approved for the current release. After regression testing is passed, integration testing begins. This testing requires detailed coordination with interacting systems. The completion of the integration test signals the start for the production readiness review (PDR).

#### TIV WAN

The TIV WAN systems development team receives approved requirements that are used to build a functional specifications document. This document identifies what the software needs to do to satisfy the requirements. This document must be signed-off by SFA before NCS will begin any development.

As development progresses, National Computer Systems (NCS) will provide system flow charts and test cases to SFA. The flow chart shows where reports are generated and files are transmitted. Test cases are developed based on the requirements forwarded by NCS. There is a test representative assigned to the effort. They are responsible for coordinating the development and distribution of the test case scenarios (with the contractor). The test cases are reviewed with SFA before testing begins. After approval of test cases and the development of the code, NCS begins testing. The types of testing being performed in an adhoc fashion are: system, performance, Y2K compliance, acceptance and beta testing. NCS will send reports of test matrices to SFA. The reports identify the test owner, project manager, and quality assurance (QA) representative. It includes a test results (pass and failure rates).

SFA pays particular attention to the PC-based code. SFA reviews the code for documentation and performance aspects. Although, NCS provides additional documentation for the mainframe solution, it is reviewed by SFA but not checked as vigorously. The mainframe documentation includes a data dictionary, algorithm check, systems flow chart, JCL catalog, coding standards, etc.

The applications are rolled out to the customer service representatives once SFA approves the testing process of National Computer Systems (NCS). There is no standard entry/exit criteria established to determine the consistent readiness of applications to be migrated. The customer service representatives are used as testers. At the same time, SFA begins the acceptance testing of the PC-based application. The application is also distributed to 5-10 selected schools for beta testing. The tests are focused on the software, the interfaces, the install/uninstall capability, etc.

If an issue is found, a document is faxed to the NCS developers. The problems are fixed and if time and availability exist, a new beta version is released to the schools. If the application has already been released, a patch is implemented on the web site for users to download.

To summarize, SFA does not have a consistently followed standard approach to system integration and testing. There is no enterprise level organization within SFA that oversees and administers system integration and testing. These points suggest a need for establishing and/or redefining system integration and testing within SFA.

## **1.4. Document Organization**

The system integration and testing approach includes the primary responsibilities of system integration and testing in the system development life cycle process. Additionally, the steps required to fulfill these responsibilities, plus a high level design of the process and the organization structure that supports the system integration and testing are included.

- Section 1: Describes the overall purpose of system integration and testing of the scope of this approach
- Section 2: Describes the system integration and testing key concepts, stages (change request, plan release, migrate configuration items, administer system integration and testing environment, manage configuration item, implementation, tracking and monitoring configuration items. This section also describes the organization and the responsibilities allocated to each element of the organization for system integration and testing. The functions that system integration and testing tools need to fulfill is covered in this sections, as is the relationship of system integration testing with other processes and organizations.
- Section 3: Describes the next steps required to implement an enterprise-wide system integration and testing process.
- Section 4: This section includes an appendix of candidate worksheets, forms, templates and checklists that can be utilized during the system integration process.

## **1.5. Document Development Process**

The following organizations and individuals were source of information in writing this approach:

- SFA Enterprise IT Management: Denise Hill
- SFA Enterprise IT Services: David A. Elliott, Phillip Wynn, Jim Cunningham
- SFA E-Commerce Application Development: Mike Rockis

- SFA CIO Chief Scientist: Constance Davis
- SFA CIO Business Manager: Harry Feely
- CSC: Wayne Burgess
- NCS: Chris Ledman

The following sources were also used:

- Andersen Consulting System Integration and Testing Best Practices
- Andersen Consulting Business Integration Methodology

## **2. System Integration and Testing Design**

System integration and testing enables the controlled and repeatable management of IT architecture components as they evolve in both development and production environments. System integration and testing implements a process by which the various enterprise IT management organizations, project teams, and business stakeholders can determine the application system's readiness for deployment at SFA. When properly implemented, system integration and testing provides efficient and prompt handling of all incidents/problems throughout the development life cycle. In addition, this design stipulates the need for cooperation between the configuration management, quality assurance, application development and independent validation & verification teams. Thus, the purpose of this system integration and testing design is to establish a sound system integration and testing process that ensures the integrity of SFA systems prior to system deployment.

### **2.1. System Integration & Testing Strategy**

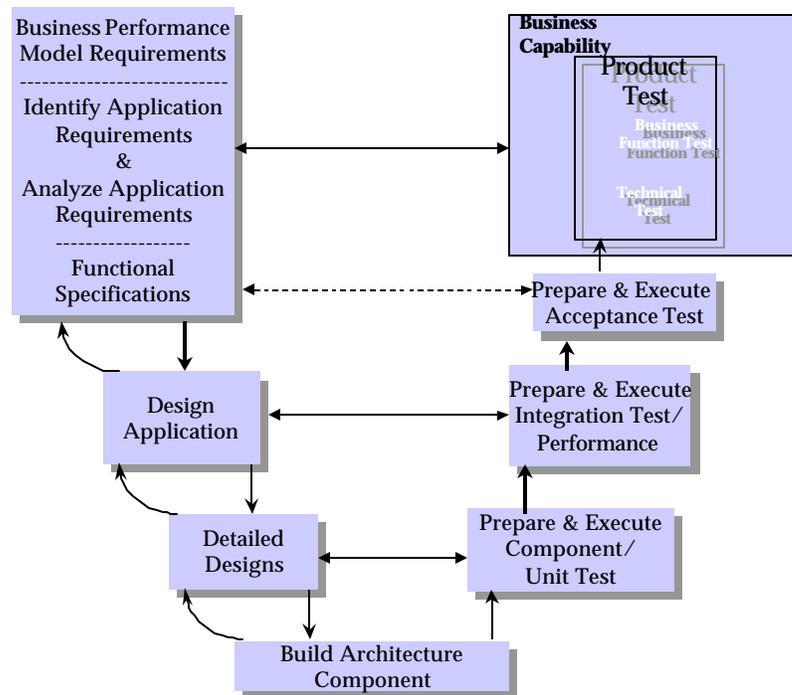
An enterprise perspective system integration and testing program requires a number of important steps be taken to realize its benefits. This section of the approach summarizes those steps. The backbone for any good approach are a set of guiding statements or principles. For the purpose of this document these statements are referred to as key concepts. These concepts are described briefly in the section to follow. The second step involves discussing the system integration and testing technical processes. Thirdly, there needs to be some discussion on the organizational requirements necessary for the success of the enterprise. And finally, after having understood the concepts, technical processes and required organizational structure, tools to do the job are the only things missing from the equation. The last section will provide a discussion on what types of tools are required to operate the enterprise system integration and testing organization.

#### **2.1.1. Key Management Concepts**

The key concepts of system integration and testing are the technical principles that a system integration and testing program are based on. This section describes those principles key to the success of an enterprise system integration and testing program at SFA. Phase containment, metrics, test data management and version control are key to managing the test process.

##### **2.1.1.1. V - Model**

To gain a head start in establishing a quality system integration and testing operation, some concepts of a proven methodological approach have been adapted for SFA use. These concepts come from the V-Model, which is an industry best practice in the area of solution development. Three concepts which are not unfamiliar, will be relied upon throughout this document, to explain how this approach will operate in the SFA environment. They, verification/validation/testing, have helped other organizations produce quality solutions that implemented their requirements. The goal is to utilize a combination of these proven methods and Andersen's best practices to provide a unique yet practical approach to system integration and testing. Below is a representation of the V-Model that is used in this system integration and testing approach.



To do this, you must first understand the V-Model basic concepts. The V-Model provides a structured development framework, emphasizing building quality in from the initial requirements stage through the final testing stage. The use of the V-Model structures the delivery processes to deliver a quality product because quality is delivered at every point in the process.

### **Basic Concepts**

The V-Model calls for each major deliverable to be **verified**, **validated**, and also **tested** for the implementation of each specification. The process of verification and validation is an attempt to catch problems as early as possible in the development life cycle and ensure that the specifications are complete, correct and adhere to standards. Testing ensures that the specifications have been properly and correctly implemented and that the solution meets the business and performance requirements.

**Verification** checks that a deliverable is correctly derived from the inputs of the corresponding stage and is internally consistent. In addition, it checks that both the output and the process conform to the standards in the project's quality plan. While the techniques used for verification and validation will vary based on the deliverable, verification is most commonly accomplished through an inspection. Inspections involve a number of reviewers, each with specific responsibilities for verifying aspects of the specification package, such as functional completeness, adherence to standards, and correct use of the technology infrastructure. An effective technique of verification is repository validation. Repository validation can be used when a design repository (via development workbenches, CASE tools, or even very strict naming conventions) are used and cross checks can be executed against the repository to ensure integrity of dependencies between deliverables. For a more extensive discussion of verification, see the Verification and Validation Guidelines job aid.

**Validation** checks that the deliverables satisfy the requirements specified in an earlier deliverable, and that the Business Case continues to be met; in other words, validation ensures that the work product, is within scope, contributes to the intended benefits, and does not have undesired side effects. While the techniques for validation will vary based on the deliverable, validation is most commonly accomplished through inspections, simulation, or prototyping. An effective technique of validation is the completion and review of traceability matrices. Validation of the design of the Technology Infrastructure may involve prototyping, while validation of a new organizational design may incorporate survey techniques. Inspections or formal reviews of design documents are frequently used for validation and verification in application design.

How validation is performed depends on the nature of the requirement in the specification document. Certain requirements can be traced directly from the specification to the implementation. In other cases, the specification concerns a quality factor or an emerging property of the implementation. Therefore, a direct comparison is not possible. In this case, validation can be done by analyzing a model of the implementation (for example, analyzing the workflow to ensure that head count does not increase and that cost is reduced), by creating and testing a prototype or by a peer or expert review (as in validating the design for maintainability criteria.)

**Testing** checks that a specification is properly implemented. Ideally, testing should only uncover problems made in translating the specifications into the product, rather than problems in the specifications themselves. The problems in the specifications themselves should be found as the result of verification and validation of the specifications when they were created.

If a deliverable fails to pass the verification, validation, or testing prescribed for it, it is demoted to the previous stage, or to the stage determined to have caused the defect.

### **2.1.1.2. Phase Containment**

The objective of phase containment is to identify and correct software defects at their source before they are passed on to a subsequent phase of development or testing. Problems become exponentially more expensive and difficult to fix the later in the development life cycle that they are detected. By concentrating on containment, the cost of fixing problems can be decreased and a quality product delivered. Phase containment is a project management style driven by the need to minimize the number of problems from development to implementation. A goal of phase containment is to minimize gaps and overlaps between the phases of testing while ensuring quality of delivery.

### **2.1.1.3. Metrics**

Metrics are performance measures that provide a mechanism to track how testing is proceeding compared to plan and how effective the development and testing phases are at containing errors (e.g., the number of problems identified and how long it takes to fix them). They can also help identify how effective the testing process is at discovering problems and areas that are error-prone. Data gathered through metrics provide input for scheduling subsequent releases and provides information to improve the processes in each phase.

### **2.1.1.4. Test Data Management**

Test data management (TDM) tools are controls and procedures that manage the quality of tests through the management of test data. The primary objective of TDM is to allow users to share and reuse test data throughout the many phases of testing. TDM provides the capability of creating, managing, maintaining and combining distinct versions of test data. Meaningful test data is essential for successful testing.

### **2.1.1.5. Version Control**

Version control is an essential tool in managing the development and testing process. The need for managing test data, different versions or releases of SFA and custom code is critical in the successful implementation and testing of SFA.

## **2.1.2. System Integration and Testing Stages**

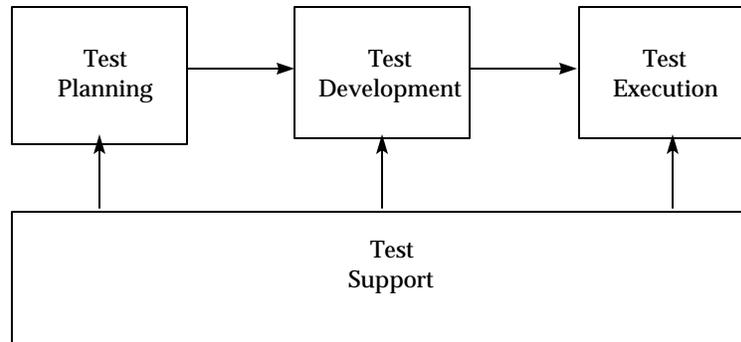
The overall objective of this testing methodology is to ensure that the new systems are production ready. At the end of a project, all team members need to feel the systems are ready to be used in a production environment. The goal of any methodology is to

accomplish these objectives in a reasonable amount of time. The following table summarizes objectives and lists an example for each stage of the testing strategy.

Stage	Objectives
Unit Test	<ul style="list-style-type: none"> <li>• Thoroughly test the units of work by focusing on all possible test conditions.</li> <li>• Test the functionality and technical components within the confines of a single unit of work.</li> <li>• Unit test is a single entity, but many unit tests occur during the testing of an application.</li> <li>• One unit test is performed for each module or program that is written or modified.</li> </ul>
Integration Test	<ul style="list-style-type: none"> <li>• Complete an environment test to insure communications and units of work are working correctly together.</li> <li>• Regression test using plans created for unit test.</li> <li>• Test the application in a simulated production environment.</li> <li>• Test dialog flows. (String Test)</li> <li>• Thoroughly test all possible business scenarios.</li> <li>• Test ties to interfaced systems.</li> </ul>
Performance Test	<ul style="list-style-type: none"> <li>• Ensure that the systems environment will support production volumes, both batch and on-line.</li> <li>• Ensure that the response times for the application are acceptable.</li> </ul>
User Acceptance Test	<ul style="list-style-type: none"> <li>• Test applications in a simulated production environment with other release applications and current production applications using business scenarios that integration systems and workflows.</li> <li>• Test new applications, interfaces from legacy systems, conversion procedures, on-line applications and batch application against requirements defined in the design stage in an integrated testing environment.</li> <li>• Ensure users verify processing is correct.</li> </ul>

Each of the stages detailed in this document will have different environmental needs. A discussion about the recommended environments and migration of applications through these environments is contained in the system integration and testing guidelines section.

In order to discuss each of these testing stages in more detail it is necessary to break them into phases. Each of the testing stages have been broken into four phases. This document will refer to these phases throughout each of the stages. The phases are as following:



The table below outlines the objectives we are striving for during each stage of testing

Phase	Objectives
Test Planning	<ul style="list-style-type: none"> <li>• Workplan has been developed.</li> <li>• Teams have been defined and rolls established.</li> <li>• Test scenarios and conditions have been planned.</li> </ul>
Test Development	<ul style="list-style-type: none"> <li>• Test scenarios, conditions and cycles have been designed.</li> <li>• Develop the test plan which includes assignments, development and execution dates.</li> <li>• Test model has been developed.</li> <li>• Test scripts (unit, string) have been produced.</li> <li>• Expected results have been defined.</li> </ul>
Test Execution	<ul style="list-style-type: none"> <li>• Unit and string tests are executed</li> <li>• Discrepancies and deficiencies have been annotated.</li> <li>• All scenarios have been executed and results verified.</li> <li>• If testing problems occur, modifications to test model are made.</li> </ul>
Test Support	<ul style="list-style-type: none"> <li>• Technical and functional support personnel are in place.</li> <li>• Testing environments have been established and verified to support the application testing support requirements.</li> </ul>

### 2.1.2.1. Unit Test

#### Overview

Once application coding is complete, the next step in development is a formal unit test. The purpose of unit test is to complete the testing of an application at the unit of work

level. The functionality and technical components are all tested within the confines of a single unit of work.

Once the objectives of one test stage are met, there is no need to repeat the same testing in subsequent stages. This is a key concept of the V-Model, and one that proves difficult to accept and use in practice. There is often a desire to retest just to "make sure everything is OK." Doing so inevitably leads to time-consuming effort. It also leaves less time to do the testing needed for the current stage of testing, ultimately resulting in minimal (if any) time available for the last stage(s) of testing. The goal is to minimize gaps and overlaps between the testing stages while ensuring quality of delivery. It is, however, very important that the testing done in each stage is well organized, documented, and repeatable.

The creation and execution of the unit test plan is a necessary step in the development of a quality application. A complete unit test plan should be driven off of the program requirements, a design deliverable, or other design specifications. The programmers will be responsible for creating these plans and should incorporate any suggested conditions from the design analyst, included in the design. Unit test plans should be reviewed by the programmers to ensure that all conditions programmed in the module are tested. Once the developer completes their unit test plan, the senior analyst reviews the unit test plan for completeness and consistency. Before proceeding with execution of the unit test, the senior analyst will sign-off on the test plan.

The test plans will be used as an environment/regression test of the application when it reaches integration test. The developers will be responsible for updating the plans with any changes made to the procedures after the plans are created.

Unit test will occur on the application development platform. The developers will create test data or revise production data. Once the test data has been created, the developer will carefully walk through each action on the script and the associated test data to produce the expected results. The expected results can be represented as file layouts as well as screen prints. In addition, the screen prints are an excellent tool to detail input to the windows as well as output.

Unit test execution begins once the developer has completed coding his/her assigned component and reviewed his/her test plan. The goal of this stage is to thoroughly test the functionality of the individual components. Each programmer will execute a unit test for each modification that is performed during programming. Unit test will also confirm the user interface looks and performs as expected. A single developer may be assigned multiple contiguous components, each of which will be tested individually and as a group.

Executing unit testing consists of executing scripts which satisfy all test conditions in the test plan. File layouts and screen prints of the actual results are then compared to the expected results. If inconsistencies are found, the application must be debugged and re-tested. As each cycle is successfully completed, it is noted as such on the unit test Status Sheet. The unit test is complete when all the cycles have been executed successfully.

When unit test is completed, the developer will collect the status sheet, unit test plan, actual results and expected results and place them in a unit test binder. It is important to organize and keep the unit test binder. This binder is the final deliverable for unit test, proof of a successful unit test, and the plans for regression testing any changes.

### **Entry/Exit Criteria for Unit Test**

A key concept inherent in the V-Model is that development and testing processes must be structured and repeatable. It is essential that the stages of the V-Model, and the processes to complete each stage, are well defined, structured, and standardized. Defined, standard processes are repeatable and measurable. Processes that are not repeatable are not measurable, and therefore they do not easily lend themselves to improvement. Tests should be reused within a release (for regression of fixes) and across releases (as enhancements are made), ensuring a consistent level of testing with a minimal level of effort. Finally, automation can be introduced to the repeatable testing processes to improve productivity and reduce human error.

A set of entry and exit criteria for each test phase will help facilitate a 'defect-free' production implementation. These criteria provide for quality management and control of the testing process through phase containment, allowing errors and defects to be identified and corrected before moving to the next testing segment. Portions of the unit test planning and test development phases will begin during the conceptual design and detail design phases of the project. However, all unit test entry criteria must be met before the unit test Execution can begin, and the unit test exit criteria must be met before the entire unit testing segment is complete.

#### **Entry Criteria for Unit Test Execution**

- Previously developed design packets are documented and available for the programmer.
- Programming of the unit has been completed. The program has had a clean compile and the programmer is confident all functionality has been programmed.
- All test conditions, cycles, scripts, test data and expected results are documented and available.

#### **Exit Criteria for Unit Test**

- All test scripts, cycles, and conditions have been executed successfully.
- Identified errors and defects have been corrected and re-tested.

- The unit test binder is completed and signed-off.
- The module has been approved for promotion to Integration Test.

### **Test Planning**

The following chart describes the planning tasks for unit test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Develop the Unit Test Plan	Application Development Teams	This document defines common testing conditions, outlines the approach for executing unit test, and describes the process and tasks to the programmers.	<ul style="list-style-type: none"> <li>• Unit Test Plan.</li> <li>• Create sample spec packet for a module.</li> <li>• Training packet for the programmers.</li> </ul>
Develop the Programming / Unit Test Schedule	Application Development Teams	This document shows the target and critical dates for completing Unit Test, and lists milestone dates for monitoring progress. The Test Schedule will be developed for both the overall Unit Test segment, and for each application.	<ul style="list-style-type: none"> <li>• Workplan detailing modules, milestones, estimated completion dates, and responsibilities.</li> </ul>
Confirm Technical Environment	Enterprise Architecture Team	The unit test technical environment must be setup prior to beginning unit test. This includes providing sample test data where possible.	<ul style="list-style-type: none"> <li>• Provides technical support for the creation/extraction of test data.</li> </ul>

### **Test Development**

The following chart describes the development tasks for unit test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Develop the Test Conditions	Application Development Team	Test conditions detail the specific program conditions and application functionality to be tested. Test conditions should be as comprehensive as possible to verify all program logic is tested, including exception conditions.	<ul style="list-style-type: none"> <li>• Test Conditions</li> </ul>

Develop Test Cycles	Application Development Team	Test cycles are logical groups of related test conditions. Test cycles help to set limits on the size of tests, thus reducing the complexity of reviewing and verifying the expected results. Test cycles should correspond to business flows, such as a valid transaction, or an error transaction.	<ul style="list-style-type: none"> <li>• Test Cycles</li> </ul>
Develop Test Data	Application Development Team	The test data that is created must test all conditions. For some conditions, invalid test data must be created to test exception conditions. Each test data record should be cross-referenced to the condition(s) it will test. Both master file data and transaction file data will be developed.	<ul style="list-style-type: none"> <li>• Test Data</li> </ul>
Develop Expected Results	Application Development Team	The expected results should be cross-referenced to test data records and test cycles. Expected results should be detailed enough so that any test team member can verify test results with little instruction, including expected report outputs, screen outputs, and file outputs.	<ul style="list-style-type: none"> <li>• Expected Results</li> </ul>

## **Test Execution**

The following chart describes the execution tasks for unit test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Verify all unit test Execution Entry Criteria Have Been Met	Application Development Teams	All unit test Execution Entry Criteria must be satisfied before executing the unit test cycles. A checklist will be completed to verify and document that all entry criteria have been met.	<ul style="list-style-type: none"> <li>Completed Unit Test Entry Criteria Checklist.</li> </ul>
Execute Test Cycles	Application Development Teams	Test cycles should be executed according to plans.	<ul style="list-style-type: none"> <li>Actual Test Results.</li> </ul>
Compare Actual Results with Expected Results	Application Development Team	Actual results are compared with expected results. If any discrepancies are found, the problem should be reviewed to identify if it is a programming error, an error in the expected results, or a user error. If there is an error in the module, it should be corrected and unit testing should be repeated. This process should be repeated until all cycles are completed without an error.	<ul style="list-style-type: none"> <li>Completed Unit Test Packet</li> </ul>
Manage Test Execution	Application Development Team	It is important that test are conducted according to the application schedule and that the status of each test is monitored carefully.	<ul style="list-style-type: none"> <li>Completed Unit Test Schedules</li> </ul>
Obtain Sign-Offs	Application Development Team	Once all test cycles have been executed successfully, and all actual results have been validated and verified, sign-off should be obtained from the functional analyst. Sign-off confirms that all user requirements have been satisfied.	<ul style="list-style-type: none"> <li>Unit Test Sign-off sheet</li> </ul>

### **Summary Unit Test Deliverable List**

Once the application components have been tested against the unit test plan and all errors corrected, the following deliverables are collected in a unit test binder. The unit test binder will also contain the unit test plan created by the developer.

- Unit Tested Application Procedures
- Test Script Packet created by the developers
  - Unit test scripts/cases.
  - Before views of the data.
  - After views of the data. (Expected Results)
  - Screen prints detailing actions and expected results.
- Unit Test Sign-off Sheet with the signature of the senior analyst and developer.
- Test data version stored for future use

### **2.1.2.2. Integration Test**

#### **Overview**

The main purposes of integration test are to:

- Test the ability of the system to meet the requirements defined by the users.
- Test the business processes being defined by the system
- Validate all possible conversation and dialog flows of the system.
- Test exception handling and error processing within the system.
- Test security handling.
- Validate the communications and technical architecture of the system.
- Validate all of the above within a simulated production environment.

Integration test concentrates on testing the completion of every possible business scenario within a given system. In contrast, unit test concentrates on testing every object-action condition within a given window or conversion. Integration test also validates unit test plans and confirms that environment changes have not affected the application. Integration test verifies that the business process design has been completed and correctly implemented, both functionally and technically. The Integration test plan will validate the system against the user requirements. In addition to the business and user requirements, integration test also covers system requirements such as security handling error handling, loss of connectivity, and performance measures.

The successful completion of integration testing requires a detailed planning and execution effort. The integration test planning team will be responsible for creating the test plans and coordinating the integration test execution. The team should create the test plans with the concept of creating the most effective and reusable plans possible. In order to finish integration test planning early enough, planning must occur parallel to or before programming and unit test. The planners will need to keep in mind that any changes to the design implemented during programming and testing must be incorporated into the integration test plan.

The integration team needs representation from both the business and technical communities within SFA. They should be rounded out with some business end user support. These requirements could be fulfilled by including the integrated product team (IPT) team members in business analyst and end user roles. The responsibilities outlined here require that the integration test planning team consist of users and functional analysts intimate with the screens, procedures and business functions of the applications.

The test development phase will generate test cycles and scripts, test data, and expected results. Together these scripts will cover all the procedures and business scenarios of the new system. The planning process will define a set of criteria or

test conditions from the user requirements documented in the requirements definition phase of the project. Test cycles will be created using the test conditions to test the system's ability to meet the criteria. Each integration test cycle should be related to a business scenario or a group of related scenarios. If a group of related scenarios exist then sub cycles should be assigned to each of the related scenarios. A script is assigned to each integration test cycle so that script and test cycle ID's will match.

Conducting integration testing will involve execution of test cycles developed in integration test planning phase. Executing the test cycles will require loading different versions of test data, executing the test scripts, and comparing the actual results with the expected results. The integration test team lead should monitor the planning completed for individual test cycles and the test data versions associated with the cycle. A version of test data will contain all the entity occurrences necessary to test the business scenarios within a cycle and its possible sub cycles. Integration test cycles address the more integrated and complex business processes within the system. During execution, if discrepancies are found between the actual and expected results, an incident investigation will be produced for each discrepancy, documenting information about the cause, symptoms and status of the discrepancy See the appendix 4.2 for details on an incident reporting process.

Once the entire integration test plan has been executed without generating any errors, integration test has been completed successfully. The completed deliverables will be collected and filed in the integration test binder. The team leader will sign-off on the Integration test completed line of an integration status report.

### **Entry/Exit Criteria for Integration Test**

A set of entry and exit criteria for the each test phase will help facilitate a 'defect-free' conversion. These criteria provide for quality management and control of the testing process through phase containment, allowing errors and defects to be identified and corrected before moving to the next testing segment. Portions of the test planning and test development phases will begin during the detail design and programming phases of the project, which is prior to all entry criteria being met. However, all entry criteria must be met before the execution can begin, and the integration test exit criteria must be met before the entire integration testing segment is complete.

#### **Entry Criteria for Integration Test Execution**

- Previously developed test packages are documented and available for the tester.
- All applications and application components necessary for cycle to be completed have been inventoried and promoted.
- Integration test plans are complete and approved.

- Resources to execute test are available.
- Results from prior testing phases are available (e.g. Unit Test).
- The integration test environment is in place.

**Exit Criteria for Integration Test**

- All test scripts, cycles, and conditions have been executed successfully.
- Identified errors and defects have been corrected and re-tested.
- All issues and incidents have been properly documented and worked through the resolution process.
- All reviews were conducted, and the appropriate sign-offs were obtained.
- All modules have been approved for promotion to performance test.

**Test Planning**

The following chart describes the planning tasks for integration test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Develop the Integration Test Plan	Integration Test Team	This document defines the approach for executing the Integration test, details the Integration test objectives, assumptions and potential risks. The plan should detail the necessary resources and test requirements.	<ul style="list-style-type: none"> <li>• Overall Integration Test Plan.</li> </ul>
Develop the Integration Test Schedule	Integration Test Team	This document shows the target and critical dates for completing Integration testing, and lists milestone dates for monitoring progress. This Schedule will include a timeline and detail events to be performed during Integration Testing.	<ul style="list-style-type: none"> <li>• Overall Integration Test Schedule</li> </ul>
Confirm Technical Environment	Enterprise Architecture Team	The Integration technical environment must be confirmed prior to beginning Integration Testing. All modules must be migrated into the Integration Test environment including all interfaces to existing systems.	<ul style="list-style-type: none"> <li>• Completed Integration Technical Environment Checklist.</li> </ul>

**Test Development**

The following chart describes the development tasks for Integration Test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
-------------	------------	--------------------	---------------------

Develop the Test Conditions	Integration Test Team	Test conditions detail the specific program conditions and application functionality to be tested. Test conditions should verify that all user requirements are tested and should be cross-referenced to the User Requirements Document and test scripts. Conditions should be developed that verify conversion changes, application changes, and interface changes.	<ul style="list-style-type: none"> <li>• Test Conditions (revived from Unit Test)</li> </ul>
Develop Test Cycles	Integration Test Team	Test cycles should correspond to business cycles, such as end of day, end of month, end of year, statement processing, exception processing, etc. Test cycles should be ordered according to the prioritization analysis outlined in the test plan and availability of the applications.	<ul style="list-style-type: none"> <li>• Test Cycles</li> </ul>
Develop Test Scripts	Integration Test Team	Test scripts organize test cycles for logical execution. Test scripts detail the steps required to test a business function/system modification (e.g. an on-line conversation or batch run)	<ul style="list-style-type: none"> <li>• Test Scripts</li> </ul>
Develop Test Data	Integration Test Team	The Integration test data should focus on testing data flow between applications. Test data should be documented in the test scripts to ensure what is expected to be tested is actually tested. Test data files should be created for both transaction files and master files.	<ul style="list-style-type: none"> <li>• Test Data</li> </ul>
Develop Expected Results	Integration Test Team	The expected results should be cross-referenced to test data records and test cycles. Expected results should be detailed enough so that any test team member can verify test results with little instruction, including expected report outputs, screen outputs, and file outputs. When documenting expected results, the output medium should be as close to production as possible (e.g. screen prints, report pages, or file dumps and listings).	<ul style="list-style-type: none"> <li>• Expected Results</li> </ul>

## **Test Execution**

The following chart describes the execution tasks for Integration Test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Verify All Integration Test Execution Entry Criteria Have Been Met	Integration Test Team	All Integration Test Execution Entry Criteria must be satisfied before executing the Integration Test cycles. A checklist will be completed to verify and document that all entry criteria have been met.	<ul style="list-style-type: none"> <li>Completed Integration Test Entry Criteria Checklist.</li> </ul>
Execute Test Scripts / Cycles	Integration Test Team	The tests are executing according to the Integration Test plans and schedules. The test cycles and scripts should simulate business cycles as closely as possible (e.g. daily, weekly, or monthly cycles)	<ul style="list-style-type: none"> <li>Actual Test Results.</li> </ul>
Compare Actual Results with Expected Results	Integration Test Team	Actual results are compared with expected results. Any discrepancies found should be logged as incidents or IR's and tracked until resolution. If there is an error in the module, it should be sent back to Programming, corrected and Unit Testing should be repeated. This process should be repeated until all cycles are completed without an error.	<ul style="list-style-type: none"> <li>Completed Integration Test Packets</li> <li>Logged System Incident Reports (IR's) in the incident request tracking system.</li> </ul>
Manage Test Execution	Integration Test Team Lead	It is important that test are conducted according to the Integration Test schedule and that the status of each test is monitored carefully. In addition, coordination and communication must be timely when responding to identified issues during the Execution phase.	<ul style="list-style-type: none"> <li>Completed Integration Test Schedules</li> </ul>
Obtain Sign-Offs	Integration Test Team	Once all test cycles have been executed successfully, and all actual results have been validated and verified, sign-off should be obtained from the functional analyst. Sign-off confirms that all user requirements have been satisfied.	<ul style="list-style-type: none"> <li>Integration Test Sign-off sheet</li> </ul>
Archive Test Results	Integration Test Team	Integration test data should be archived for later regression testing.	<ul style="list-style-type: none"> <li>Archived Integration Test data.</li> </ul>

## **Summary Integration Test Deliverable List**

When all the integration test cycles and scripts have been completed without error, the integration test for this release is considered complete. As each cycle is completed during Integration test, the date and number of errors found should be recorded on an integration

test status sheet. The status sheet is the final deliverable for integration test and should be filed and stored in the application test binder at the end of the application's Integration testing cycle.

- ❑ System Tested, Production Ready Application
- ❑ Integration Test Condition Checklist
- ❑ Test Cycle Documentation
  - Completed Test Cycle Status Sheet
  - Test scripts
  - Before views of the database
  - After views of the database (Expected results)
  - Screen prints detailing actions and expected results
- ❑ Integration Test Status Sheet with sign-off from Team Leaders
- ❑ Test Data Versions stored for future use

### **2.1.2.3. Performance Test**

#### **Overview**

Performance testing is an extension of the integration test and a volume/stress test of the system. During performance testing the system's processing and response time will be documented under peak transaction loads. In order to simulate these production sized volumes, performance test is executed in the user acceptance test environment.

The objectives of performance test are to:

- Test the system's processing and response time
- Test the back-up and recovery procedures

Both on-line and batch processing parameters will be verified. The technical team will perform necessary system tuning and disk space analysis as required prior to implementation.

The back-up and recovery testing will confirm that system back-up and recovery procedures perform to expectation and that corrupted files can be recovered from saved versions. These tests will be utilized for testing database, application, and system file backup and recovery time-frames and accuracy.

During the test planning phase of performance test, operations will work closely with the Integration testing team to develop an approach to test the system's performance. This will require simulating large volumes of transactions being processed by the system.

The first cycle of performance test is to perform a mock installation of the application. This mock installation involves collecting the production architecture components and application components, and attempting to install and configure them. The installability/configurability of these components and the reliability of the roll-out procedures are tested in this initial test cycle. (If data conversion is required for the implementation, a mock conversion of the data should be included. This mock conversion will test the reliability of the data conversion procedures and provide the data foundation for performance test).

After the mock installation is complete, the performance test will perform an exhaustive test of normal processing, batch processing, and error processing. These tests should follow normal cycles and cover all major processing periods (e.g. end of day, end of week, end of month, end of year). All critical response times and batch windows should be measured and recorded. After each of these cycles, backup and restoration procedures should follow to test the capability to backup and restore the system.

During execution, if discrepancies are found between the actual and expected results, an incident investigation request will be produced for each discrepancy, documenting information about the cause, symptoms and status of the discrepancy (see appendix 4.2 Incident Reporting).

Once all cycles have been complete and response times and batch windows are within the required metrics, performance test has been completed successfully. The completed deliverables should be collected and filed in a performance test binder.

### **Entry/Exit Criteria for Performance Test**

#### Entry Criteria for Performance Test Execution

- All applications necessary for cycle to be completed have been inventoried and promoted.
- Performance test plans are complete and approved.
- Resources to execute test are available.
- Results from prior testing phases are available (i.e. Integration Test).
- The performance test environment is in place.

#### Exit Criteria for Performance Test

- All test scripts, cycles, and conditions have been executed successfully.
- Identified defects have been corrected and re-tested.
- All issues and incidents have been properly documented and worked through the resolution process.
- All reviews were conducted, and the appropriate sign-offs were obtained.

## **Test Planning**

The following chart describes the test planning tasks for the performance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Develop the Performance Test Plan	Integration Test Team  Operational Support	This document defines the approach for executing the Performance Test, and details the objectives, assumptions and potential risks. Also included in the test Plan are the test requirements and the test resources. The test Plan will detail how to simulate large transaction volumes and critical response time areas to be tested.	<ul style="list-style-type: none"> <li>• Overall Performance Test Plan</li> </ul>
Develop the Performance Test Schedule	Integration Test Team  Operational Support	This document shows the target and critical dates for completing Performance testing, and lists milestone dates for monitoring progress. This Schedule will be critical for scheduling the necessary resources at the necessary times. Performance testing will use a large amount of the system resources, so scheduling will be very important.	<ul style="list-style-type: none"> <li>• Overall Performance Test Schedule</li> </ul>
Confirm Technical Environment	Enterprise Architecture Team	The performance technical environment must be confirmed prior to beginning performance Testing. All modules must be migrated into the performance Test environment including all interfaces to existing systems and the environment must simulate a true production environment. The data and resources to simulate large transaction volumes must be identified.	<ul style="list-style-type: none"> <li>• Completed Performance Technical Environment Checklist</li> </ul>

## **Test Development**

The following chart describes the test development tasks for the performance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Review / Modify Test Cycles	Integration Test Team  Operational Support	Test cycles developed for Integration Test will serve as the basis for test cycles developed for Performance Testing, and should be carefully reviewed to ensure that all key areas of concern will be tested within the allotted time frame. Any additional cycles should be developed along the same guidelines that the Integration test cycles were developed. Satisfying user response times and volumes, meeting batch window timeframes, and verifying backup and recovery procedures are the main focus of the test cycles.	<ul style="list-style-type: none"> <li>• Test Cycles</li> <li>• Test Cycle Approval Form</li> </ul>
Review / Modify Test Scripts	Integration Test Team  Operational Support	The test scripts developed for Integration testing will serve as the basis for test scripts developed for Performance Testing, and should be carefully reviewed to ensure that all key areas of concern will be tested within the allotted test time frame. Any additional scripts should be developed along the same guidelines that the Integration test scripts were developed. Satisfying user response times and volumes, meeting batch window timeframes, and verifying backup and recovery procedures are the main focus of the test scripts.	<ul style="list-style-type: none"> <li>• Test Scripts</li> <li>• Performance Test Script Approval Form</li> </ul>
Review / Modify Test Data	Integration Test Team  Operational Support	The test data developed for Integration test will serve as the basis for test data selected for Performance Test, but the test data for performance test should be large enough to simulate a heavy volume of traffic through the system.	<ul style="list-style-type: none"> <li>• Reviewed/Modified Test Data</li> </ul>
Configure Tools	Integration Test Team  Operational Support	If an automated testing tool is used to simulate production volume the tool is set-up for use by the Integration Test Team	<ul style="list-style-type: none"> <li>• Configured Tool</li> </ul>
Review Expected Results	Integration Test Team  Operational Support	Response time, volume, and batch window requirements from the User Requirements will serve as the expected results for Performance Test.	<ul style="list-style-type: none"> <li>• Expected Results</li> </ul>

## **Test Execution**

The following chart describes the test execution tasks for performance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Verify All Performance Test Execution Entry Criteria Have Been Met	Integration Test Team  Operational Support	All Performance Test Execution Entry Criteria must be satisfied before executing the Performance Test cycles. A checklist will be completed to verify and document that all entry criteria have been met.	<ul style="list-style-type: none"> <li>Completed Performance Test Entry Criteria Checklist.</li> </ul>
Execute Test Scripts / Cycles	Integration Test Team  Operational Support	The tests are executing according to the Performance Test plans and schedules. The test cycles and scripts should simulate business cycles as closely as possible (e.g. daily, weekly, or monthly cycles) and will simulate large transaction volumes.	<ul style="list-style-type: none"> <li>Actual Test Results.</li> </ul>
Compare Actual Results with Expected Results	Integration Test Team  Operational Support	Actual results are compared with expected results. Any discrepancies found should be logged as incidents or IR's and tracked until resolution.	<ul style="list-style-type: none"> <li>Actual Test Results</li> <li>Logged Incident Reports (IR's) in the IR tracking system.</li> </ul>
Manage Test Execution	Integration Test Team Lead  Operational Support	It is important that test are conducted according to the Performance Test schedule and that the status of each test is monitored carefully. In addition, coordination and communication must be timely when responding to identified issues during the Execution phase.	<ul style="list-style-type: none"> <li>Completed Performance Test Schedules</li> </ul>
Obtain Sign-Offs	Integration Test Team  Operational Support	Once all test cycles have been executed successfully, and all actual results have been validated and verified, sign-off should be obtained from the functional analyst. Sign-off confirms that all user requirements have been satisfied.	<ul style="list-style-type: none"> <li>Performance Test Sign-off sheet</li> </ul>
Archive Test Results	Integration Test Team	Performance test data should be archived for later regression testing.	<ul style="list-style-type: none"> <li>Archived Performance Test data.</li> </ul>

## **Summary Performance Test Deliverable List**

At the end of performance test, the users must sign-off on the response times and performance of the system. A log will be created containing Issues/Improvements for future releases of the system.

- ❑ Signed Performance Test Checklist
- ❑ Completed Issues/Improvements log

#### **2.1.2.4. User Acceptance Test**

##### **Overview**

The user acceptance test is the final functional test of the system prior to production, and should verify that the system meets all user requirements and expectations. User acceptance test is the actual simulation of the working conditions of the new/revised system and converted data. The test is conducted by end user personnel and the Development Team, with assistance from the Application Development Teams in providing direction and correcting errors.

The objectives of user acceptance test are to:

- Test all processing cycles in the system to ensure the results meet the users' requirements in as close to the business environment as possible.
- Obtain end user sign-off that the system meets all user requirements as detailed in the user requirements documentation.

As part of the application development life cycle, detailed workflows of the application under development will be available as inputs to user acceptance test planning. The functional design teams in conjunction with users of the application will be responsible for developing these workflows and maintaining them. Workflows in conjunction with detail design documents developed during detailed design and functional design will be the basis of detailed scripts which the user will execute as part of user acceptance testing. It is important to realize that user acceptance test should be testing against the business requirements drafted in the detail design stage rather than the "expectations" of the users. This is not the correct stage of the project to define new requirements.

During the test planning phase of user acceptance, users will be actively involved in formulating the business scenarios they wish to test. These scenarios may result from application walk-throughs and brainstorming sessions with the planning team. The scenarios should represent an integration between the system and process workflows and should simulate real-life operations. The final checklist which is signed off on will be comprised of these scenarios.

Detailed test scripts and data are developed and assigned with business scenarios so that users can check off business scenarios as they successfully execute test scripts. Integration test scripts can be re-used if they can be matched to these business scenarios. If the users are not satisfied with the testing scripts, they may create a separate set of user acceptance testing checklists, test scripts and data or any combination thereof.

The test execution of user acceptance test is the actual process where the acceptance test procedures and checklist, which were mutually agreed upon during the previous

planning phase, are used. The users are responsible for verifying, with the help of the test scripts developed, that the inputs are properly taken into the system, processing is correct, all error conditions are properly detected, and all outputs produced are accurate. Issues and problems are documented during this phase. If discrepancies are found within the system, the developers in concert with the user will determine the appropriate resolution.

The deliverables at the end of user acceptance test are the completed user acceptance checklist, user manuals (if necessary) and a log of issues / improvements that can be incorporated in new releases of the system. The users must sign-off on all deliverables individually to indicate that the system and all deliverables have been approved.

### **Entry/Exit Criteria for User Acceptance Test**

#### Entry Criteria for User Acceptance Test Execution

- Previously developed test packages are documented and available for the tester.
- All applications and application components necessary for user acceptance test have been inventoried and promoted.
- User acceptance test plan has been completed and approved.
- Resources to execute test are available.
- Results from prior testing phases are available (e.g. Integration and Performance Testing).
- The user acceptance test environment is in place.

#### Exit Criteria for User Acceptance Test

- All test scripts and cycles have been executed successfully and to the users approval.
- Identified errors and defects have been corrected and re-tested.
- All issues and incidents have been properly documented and worked through the resolution process.
- All reviews were conducted, and the appropriate user sign-offs were obtained.

## **Test Planning**

The following chart describes the test planning tasks for the user acceptance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Develop the User Acceptance Test Plan	Development Team  Integration Test Team	This document defines the approach for executing the User Acceptance Test, and details the objectives, assumptions and potential risks. Also included in the Test Plan are the test requirements and the test resources. The Test Plan will be similar to the Integration Test plan.	<ul style="list-style-type: none"> <li>Overall User Acceptance Test Plan</li> </ul>
Modify the User Acceptance Test Plan	IV&V Contractor  Quality Assurance	Augment User Acceptance Test with Independent Verification & Testing (IV&V) tasks.	<ul style="list-style-type: none"> <li>Overall User Acceptance Test Plan</li> </ul>
Develop the User Acceptance Test Schedule	Development Team  Integration Test Team	This document shows the target and critical dates for completing User Acceptance testing, and lists milestone dates for monitoring progress. This Schedule will be similar to the Integration Test schedule.	<ul style="list-style-type: none"> <li>Overall User Acceptance Test Schedule</li> </ul>
Confirm Technical Environment	Enterprise Architecture Team  Configuration Management	The User Acceptance technical environment must be confirmed prior to beginning User Acceptance Testing. All modules must be migrated into the User Acceptance Test environment including all interfaces to existing systems.	<ul style="list-style-type: none"> <li>Completed User Acceptance Technical Environment Checklist</li> </ul>

## **Test Development**

The following chart describes the test development tasks for the user acceptance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Review / Modify Test Cycles	Development Team  Integration Test Team	Test cycles developed for integration test will serve as the basis for test cycles developed for user acceptance testing, and should be carefully reviewed to ensure that all key user requirements will be tested within the allotted time frame. Any additional cycles should be developed along the same guidelines that the Integration test cycles were developed. Satisfying user requirements is the main focus when developing new test cycles.	<ul style="list-style-type: none"> <li>• Test Cycles</li> <li>• Test Cycle Approval Form</li> </ul>
Review / Modify Test Scripts	Development Team  Integration Test Team	The test scripts developed for Integration testing will serve as the basis for test scripts developed for user acceptance testing, and should be carefully reviewed to ensure that all key user requirements will be tested within the allotted test time frame. Any additional scripts should be developed along the same guidelines that the Integration test scripts were developed. Satisfying user requirements is the main focus when developing new test scripts.	<ul style="list-style-type: none"> <li>• Test Scripts</li> <li>• User Acceptance Test Script Approval Form</li> </ul>
Review / Modify Test Data	Development Team  Integration Test Team	The test data developed for integration test will serve as the basis for test data selected for user acceptance test, but the test data for user acceptance test should contain only converted production data. This will help to test actual business events, and provide the end user with realistic test results.	<ul style="list-style-type: none"> <li>• Reviewed/Modified Test Data</li> </ul>
Develop IV&V Test Scripts & Data	IV&V Contractor  Quality Assurance	Test scripts and data will be developed. It will be based upon industry best practices federal guidelines.	<ul style="list-style-type: none"> <li>• Test Scripts</li> <li>• User Acceptance Test Script Approval Form</li> <li>• Expected Results</li> </ul>
Review/Modify Expected Results	Development Team  Integration Test Team	The expected results developed for integration test will serve as the basis for expected results selected for user acceptance test, but will be modified to correspond to the converted production data. This will verify actual business events, and provide the end user with realistic test results. Included in the expected results will be report outputs, screen outputs, and file outputs.	<ul style="list-style-type: none"> <li>• Expected Results</li> </ul>

Confirm Application Flows	Development Team	The user acceptance test application flows should be reviewed to ensure that all jobs have been correctly modified for the user acceptance test environment.	<ul style="list-style-type: none"> <li>Confirmed User Acceptance Test Flows</li> </ul>
---------------------------	------------------	--	--

### **Test Execution**

The following chart describes the test execution tasks for user acceptance test:

<b>What</b>	<b>Who</b>	<b>Description</b>	<b>Deliverables</b>
Verify All User Acceptance Test Execution Entry Criteria Have Been Met	Development Team Integration Test Team	All user acceptance test execution entry criteria must be satisfied before executing the user acceptance test cycles. A checklist will be completed to verify and document that all entry criteria have been met.	<ul style="list-style-type: none"> <li>Completed User Acceptance Test Entry Criteria Checklist.</li> </ul>
Execute Test Scripts / Cycles	Integration Test Team	The tests are executing according to the user acceptance test plans and schedules. The test cycles and scripts should simulate business cycles as closely as possible (e.g. daily, weekly, or monthly cycles)	<ul style="list-style-type: none"> <li>Actual Test Results.</li> </ul>
Compare Actual Results with Expected Results	Integration Test Team	Actual results are compared with expected results. Any discrepancies found should be logged as incidents and tracked until resolution.	<ul style="list-style-type: none"> <li>Actual Test Results</li> <li>Logged System Incident Reports (IR's) in the IR tracking system.</li> </ul>
Manage Test Execution	Integration Test Team	It is important that test are conducted according to the user acceptance test schedule and that the status of each test is monitored carefully. In addition, coordination and communication must be timely when responding to identified issues during the test execution phase.	<ul style="list-style-type: none"> <li>Completed User Acceptance Test Schedules</li> </ul>
Execute Certification Test	IV&V Contractor Quality Assurance	Perform the independent verification and validation testing.	<ul style="list-style-type: none"> <li>Completed IV&amp;V Test</li> </ul>
Obtain Sign-Offs	Integration Test Team	Once all test cycles have been executed successfully, and all actual results have been validated and verified, sign-off should be obtained from the functional analyst. Sign-off confirms that all user requirements have been satisfied.	<ul style="list-style-type: none"> <li>User Acceptance Test Sign-off sheet</li> </ul>
Archive Test Results	Configuration Management	User acceptance test data should be archived for later regression testing.	<ul style="list-style-type: none"> <li>Archived User Acceptance Test data.</li> </ul>

### **Summary User Acceptance Deliverable List**

At the end of user acceptance test, the users must sign-off on all deliverables indicating that the system and all deliverables have been approved. A log will be created containing Issues/Improvements for future releases of the system.

- ❑ Signed User Acceptance Checklist
- ❑ Completed Issues/Improvements log

#### **2.1.3. Testing Success Factors**

##### **2.1.3.1. Test Planning**

Thorough planning is important for testing due to the interactions between the test team, enterprise architecture team, and development team(s). The integration testing lead must identify all activities that need to take place and the dependencies between these activities.

Coding and unit testing is done by the same person and is therefore tightly integrated. These tasks should be budgeted and scheduled separately. This way the schedule reflects the amount of time and effort necessary to code and to test. Therefore, the two tasks can be individually managed and phase containment can be tracked. Note, integration through user acceptance testing will be conducted by individuals (teams) solely responsible for these activities. They will not be performed by the same development/programming person(s).

Management will monitor the delivery of components from unit test to string test and from string test to user acceptance test in order to keep all three test stages functioning smoothly. They will adjust the delivery schedule, if necessary, as testing progresses. Management must also ensure that that migrations have occurred successfully.

##### **2.1.3.2. Communication**

Effective communications are important for testing due to the interactions between the test teams, technical teams, and development teams. The integration testing lead will need to keep the test team as well as other teams and project managers updated on the progress being made by:

- scheduling meetings
- updating all reports as needed

- determine frequency of meetings and reports
- determine who needs to receive information both internally and externally

The communications for a specific test should tie into the overall project's communication plan.

### **2.1.3.3. Metrics**

Metrics are performance measures that provide a mechanism to track how testing is proceeding compared to plan, and how effective the development and testing phases are at containing errors. They can also help identify how effective the testing process is at discovering problems and areas that are error-prone. Data gathered through metrics provides input for scheduling subsequent releases and provides information to improve the processes in each phase.

The following metrics should be tracked for the unit and string tests:

- Number of Errors Sorted by Type (i.e., how many problems are due to coding, system or design)
- Rate of Error Discovery (i.e., how many problems are being identified over a specified time)
- Number of Test Cycles Run per Day (i.e., how fast is the testing of the cycles proceeding)
- Number of Cycles Necessary to Complete the Test (i.e., how many cycles are required to test the unit or string)

It should be noted, the metric collection and evaluation activities are a full life cycle activity. It should become one of your ongoing process improvement duties. Additional information on metrics can be found in the reference section of the appendix.

### Test Reports

Using metrics, reports will be generated throughout the testing process. These will aid in communication the progress of the test and the status of dependencies. The two main report types that will be used are:

- **Test Progress Reports** - This weekly report, which updates the progress of the test , includes how many test conditions, scenarios and scripts have been documented and executed compared to plan. It also tracks the number and type of problems encountered. (See the appendix section E for an example.)

- **Test Summary** - This summarizes the testing effort and includes the numbers and types of problems logged, testing limitations, and who was involved. (See the appendix section F for an example.)

#### **2.1.3.4. Managing the Testing Environment**

##### **Test Data Management (TDM)**

###### Approach

TDM will be the responsibility of the testing team and technical support staff. Dedicated personnel are required, since a large number of test databases must be kept synchronized. Test data management (TDM) tools are controls and procedures that manage the quality of tests through the management of test data. The primary objective of TDM is to allow users to share and reuse test data throughout the many phases of testing. TDM provides the capability of creating, managing, maintaining and combining distinct versions of test data. Meaningful test data is essential for successful testing.

###### Common Test Data

Data for the unit test will need to be created specifically to test detailed test conditions. The string test should be able to reuse the unit test data with modifications.

###### Data maintenance

- Over time, changes will have to be made to the common test data. These changes may be due to modifications made to the physical data model, to defects found in the original test data or to additional common data needed to execute new test conditions.

#### **Version Control and Component Migration**

##### Version Control

Version control will be the responsibility of a dedicated person on the testing team with ongoing interaction with the technical team. Version control consists of managing the following components through the development lifecycle:

- SFA releases;
- Custom code;

- Other components (documentation, help files, report writers, designs, executables);
- Test model.

Regression testing will be enhanced by the fact that test components are maintained in multiple versions over time and thus reusable. The version control processes ensure that an accurate component audit trail is maintained, including modification time and date, the modifier's name, and the reason why the modification was made.

### Component Migration

The testing process will contain many testing phases. After the exit criteria have been met for a particular component in a particular phase, that component is ready for the next phase of testing. Some components are interrelated and should not be migrated to another environment without the others. Managing the migration of these components is very important to the testing model. This may require the component to be migrated into a different environment. This migration can be accomplished in one of three ways: physical migration, logical migration or a combination of the two.

The type of migration that is recommended is the combination migration. This type of migration allows for continuing development to occur in one environment while testing takes place in another environment. However, it requires far less management of additional components and physical space to store the additional components.

### **2.1.3.5. Risk Management**

Constant attention must be given to changes occurring within the project and the risk these changes present. Changes may include:

- design changes;
- code changes;
- scope;
- schedule changes.

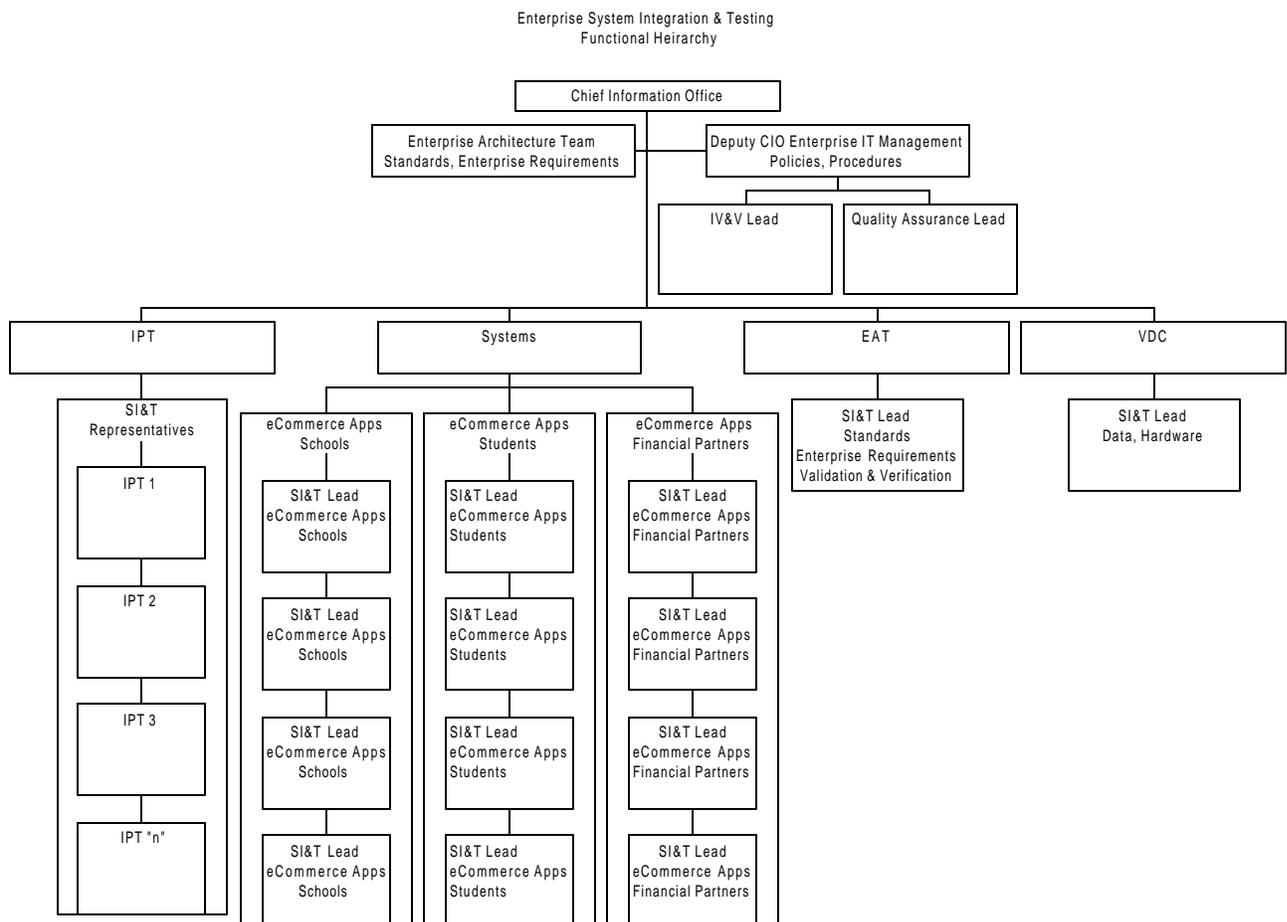
Decisions made by the integration testing lead must always reflect the risk presented by the situation and the relative risk of one action versus another, or the risk of no action.

### **2.1.3.6. Test Problem/Observation Logs Fix-It Process**

A “problem” is defined as anything that does not meet the expected result during execution of a test phase. As noted most problems identified in unit and string testing will be resolved by the developer doing the coding and testing. Problems that cannot be resolved by the developer should be turned over to the integration testing lead or team leader with appropriate documentation to allow for the problem to be recreated. The team leader/manager may need to convene a meeting with the appropriate personnel to resolve the issue.

## 2.2. Organization and Responsibilities

The SFA Modernization project organization is designed to provide management visibility into the technical work areas and ensure effective lines of authority, supervision, and communication. This section describes the system integration and testing-related project organization, details the authority and the specific responsibilities for system integration and testing for each element throughout the project life cycle, and identifies the specific resources necessary to perform the system integration and testing functions effectively.



Functionally and organizationally, the CIO has overall responsibility for instituting and leading a system integration and testing approach at SFA. However, the Enterprise Architecture Team (EAT) and the Deputy CIO Enterprise IT Management teams have co-responsibilities to support and provide guidance in the areas of standard practices, procedures and guidelines. The efforts. has overall responsibility for the integration and testing of SFA application systems. The

enterprise architecture team is responsible for defining the system integration and testing approach (this document) and establishing a format to measure the effectiveness of the approach (once implemented). The Deputy CIO Enterprise IT Management team will be responsible for monitoring the implementation of system integration and testing effectiveness.

The day to day implementation of system integration and testing policy and procedures is the responsibility of Deputy CIO Enterprise IT eCommerce Applications organization. Functionally, all SFA Modernization Blueprint system/project teams will come under this organization. This is illustrated in the organizational chart by the boxes indicating SI&T Lead. It is proposed there should be some specified individuals responsible for leading and coordinating the SFA application development project teams' testing efforts. They can be referred to as the integration testing lead. They would utilize the approved testing methodology and the tool suites.

There are two organizational functions that hold the responsibility to ensure the success of the SFA system integration and testing program: system integration and testing and QA. A validation and verification lead/team would independently evaluate and monitor deliverable items and analyze the metrics collected on project progress. The system integration and testing team would be responsible for the implementation of system integration and testing policy and practices as it applies to both documentation and software. The integrated product teams (IPT) shall have delegated responsibility for system integration and testing practices as applicable.

### **2.3. Relationship to Other Processes**

The CIO organization has ultimate responsibility for the system integration and testing program at SFA. They utilize their Deputy CIO Enterprise IT Management, Services and eCommerce Applications organizations to direct, coordinate and administer the policies, procedures and guidelines established by the enterprise architecture team (EAT). Integrated Product Teams (IPT), as the organization chart shows, have a vested interest in the smooth operation of the system integration and testing operation. They provide valued input into the process. SFA program management is responsible for the quality and integrity of the project. The prospective project leader approves the system integration and test planning, policy, and associated procedures to be implemented on their projects. A key task of management is to review and approve personnel, resources, and equipment required to implement a sound system integration and testing program. As problems occur, management must direct SFA team members to implement a corrective action and follow system integration and testing policy and procedures.

## **3. Next Steps and Implementation Plan**

### **3.1. Next Steps**

#### **3.1.1. Identify the Implementers of Enterprise System Integration & Testing**

Upon the decision to move forward, the first step required to fulfill the need of an enterprise focused system integration organization is to build the support organization that is going to drive it. However, to start, someone or some team must be defined to lead the charge. They should be tasked with the responsibility for obtaining and identifying the resources and implementing the organizational structure for the SFA. Tasked with this responsibility, one of the first decisions that needs to be made is whether the SFA would be better suited if the system integration and testing organization were composed of SFA staffers or whether it should be outsourced.

Our recommendation is the Modernization Partner Enterprise Architecture Team – Delivery QA unit, be tasked to work with the CIO Enterprise IT Management organization to implement the necessary structures, policies, procedures and guidelines. This is recommended because the potential time lag in identifying and mobilizing the appropriately skilled SFA personnel, could be offset by the readily available resources of Modernization Partner. Utilizing Modernization Partner personnel at least for the first phase of implementation would afford SFA ample time to conduct a personnel search (and training). It should be noted, SFA's CIO Enterprise IT eCommerce Applications group, would manage the system integration and testing effort. Upon the conclusion of the first phase and once the organization has become operational, at SFA's discretion, the system integration and testing organization could revert to a SFA operation.

#### **3.1.2. Define Scope of Implementation and Timeline**

The second step is to define the scope of the implementation and determine how much time should be allocated to implement it. This endeavor can not be accomplished overnight. The scope in this case could mean multiple things. For instance, the size of the integration testing team must be determined beforehand.

A phased implementation approach will be required to implement something of this magnitude. A detailed project plan must be developed containing well defined tasks/objectives with measurable milestones. One or two individuals working with the CIO Enterprise IT Management team could produce the detailed project plan in the next period.

## **3.2. Implementation Plan**

### **3.2.1. Mobilize Implementation Team**

The third step is to mobilize the implementation team. This would include obtaining the resources; conduct training, if required; make task assignments; and begin the development of the project plan tasks). See the organization structure outlined in section 2.2 of this approach.

The next steps, after approval of this approach, should be to:

1. Determine the team structure which will work best for problem fixes. This will be done during the installation phase;
2. Assign individuals to each of the fix-it roles (test team fix-it coordinator, fix-it team coordinator, fixer and migration contact). These assignments will be made during the installation;
3. Determine the testing tool requirements, criteria and purchase, if the tool(s) are not currently available at SFA. These tools may be part of an automated testing tools suite or version control tool suite.

A problem tracking tool is a key component of an efficient process for fixing problems. The tool will enable the tracking of problems through their life cycle and support phase containment and tracking metrics. The tool and this approach are intended to make the fix-it process as efficient as possible and ensure that teams and individuals clearly understand their role in the process.

### **3.2.2. Prepare System Integration & Testing Environment**

#### **3.2.2.1. Establish Testing Environment(s)**

It is important to establish an environment where the enterprise system integration and testing model can be tested. Typically a proof of concept would be required to provide a level of acceptance for the approach. To do this a platform and representative SFA application system would be needed as part of a pilot program.

A pilot platform needs to be established. This can occur during the implementation of this approach. Representative SFA systems should be identified for inclusion in the pilot program. It is very important to review the proposed system integration and testing model prior to a full utilization by the SFA community.

#### **3.2.2.2. Certify Environment**

The objective of this step is to construct and certify technical environments in support of development and test activities. Here, components are created, linked and certified to ensure the technical environment is established. In addition, service level agreements (SLAs) are established and support procedures are defined.

#### **3.2.2.3. Support Environment**

It ensures that the technical environment functions as requested throughout the development or testing effort. In this phase, issues are addressed, change requests are processed, software code rollups are facilitated, software fixes are migrated and data is backed up, restored and refreshed. A combination of the following organizations and teams (CIO IT Services, configuration management team, virtual data center personnel) performs hardware, architecture and software support, initial investigation of issues/discrepancies, identification of change requests and resolution of discrepancies.

#### **3.2.2.4. Training**

Train Personnel in overall process

The first step is to ensure that everybody in the SFA organization is familiar with the system integration and testing process. It is critical everyone has a good level of understanding of the entry and exit criteria. Sufficient training beyond the basics should be established. Every person involved in the integration testing process should attend this training.

Establish Training

In an environment with multiple systems like SFA, organizational training is crucial. Individuals from different project teams need to work together to accomplish desired ends. The organization as a whole, when first established, will be inefficient, and an allowance should be made for this factor. One approach is to establish the integration testing presence with a core group of people (integration testing leads) before putting the new system integration and testing process fully in place. This allows the team leads to organize their teams, establish the detailed written procedures by which each individual team will operate, test the procedures out, and modify them to the point at which a person has a workable solution.

Specific Tool Training

Vendor tool training, if necessary, should take place early in the establishment of the system integration & testing program. Only those using the different system integration and testing tools should participate in vendor training.

### **3.2.3. Establish Metrics**

#### **3.2.3.1. How?**

In order to achieve these goals, the following approach to implementing metrics will need to be followed:

- Identify metrics;
- Define purpose of each metric;
- Define how each metric will be collected, analyzed and reported;
- Develop process for taking corrective action and evaluating;
- Develop communication methodology.

#### **3.2.3.2. Identify Metrics**

The first step in the approach is to identify which metrics to use. The approach will be to initially implement some metrics (instead of implementing all at once) that will improve management of the test process (goal 1) and improve the quality of the test process (goal 2). This piloting approach provides time to ascertain how these metrics work, identify adjustments and provide information to determine which additional metrics would be most useful. These metrics must be measurable and provide information that can be used to improve the testing process or to make planning and resource decisions.

#### **3.2.3.3. Define Purpose of each Metric**

The purpose of each metric should be defined, along with the benefit to be gained from it, how it will be calculated and what the target measurement will be. Additionally, each metric will include interpretation guidelines to help explain whether the actual measure indicates the process is working well or poorly. At this phase, stakeholders will need to be educated on the purpose of the metrics and to provide feedback to help refine the metrics.

#### **3.2.3.4. Define how each Metric will be Collected, Analyzed and Reported**

Data for most metrics should be collected, analyzed and reported weekly. Collection of the metrics data should “fall out” of the process and not add significant work to the process. A tool (Excel or other) will be used to track and display the data, as well as to show it graphically. Analysis of the data will include reviewing the variance between actual and target measurements for the period and analyzing the trend to see if there is improvement, decline, or no change in performance.

#### **3.2.3.5. Develop Process for Taking Corrective Action and Evaluating**

Once problems or opportunities for improvements are identified, the next step will be to determine what corrective action should be taken. Examples of corrective actions include: additional training, root cause analysis, process changes or estimate modifications. After actions are taken, the metrics should be reviewed to ensure that the desired results have been obtained.

#### **3.2.3.6. Develop Communication Methodology**

Throughout the development of the metrics, communication with other key stakeholders, including the test team, process teams and management, will be critical to assure that the metrics are appropriate and understood. The metrics will be communicated to the testing team and to other applicable stakeholders on a weekly basis. This information may also be posted so good progress can be celebrated and areas for improvement can be identified and understood. These postings will also help project management to quickly and easily assess testing status and quality.

### **3.2.4. Define Test Resource Requirements**

The purpose of this task is to determine the human resource requirements for the product test in order to satisfy the high level workplan developed, taking into consideration the risks (e.g. resource constrains, schedules) involved.

#### **Major Inputs**

- High-level Workplan
- Tasks list and tasks schedules

#### **Major Outputs**

- Skill sets required by key personnel
- Resource requirements for all tasks

### **3.2.5. Prepare Testing Environment(s)**

### **3.2.5.1. Define Test Environment Requirements**

The purpose of this task is to define the product test environment requirements and ensure that they model the production environments as closely as possible, including production size databases, production machine configurations, automated and manual processes and all interfaces. Additional equipment such as printers, terminals and other supplies may also be required for the product test. These additional resources must be available to avoid lost productivity in testing.

It is important to keep in mind, the requirements for each level of product testing as well as the existing requirements for the baseline testing and staging environments - the effort and expense required to establish and maintain these environments will be significant. As such, careful product test environment planning should ensure that environments can be shared (or transitioned from one test to the next) wherever is possible.

The product test environment issues are:

- Number of product test environments required (for both Business Function and Technical testing)
- Disk capacity requirements
- Processing power requirements
- RAM requirements
- Machine setup and configuration for each environment
- Database setup and configuration for each environment
- Interfaces with external systems (switches, customer care system, etc.)

#### **Major Inputs**

- Product Test Objectives, Scope and Risks
- SFA requirements and functional specifications for each release
- System requirements from SFA and application to support product test

#### **Major Outputs**

- Instructions to the environment Support team to create the necessary environments within the required timescales

### **3.2.5.2. Define Test Entry and Exit Criteria**

The purpose of this task is to define specific criteria that define :

- The conditions that need to be satisfied for product test to start and
  - The conditions that need to be satisfied for product test to be considered complete
- Within product test, there are separate entry and exit criteria for product test planning, preparation and execution :

- **Product Test Planning** can not begin until the business case (and models) are complete and verified, and the development of the release requirements (and application release plan) have been started. Product Test planning is not complete until the strategy and approach are finished, along with the test conditions and test cycles related to the specific release requirement specifications.
- **Product Test Preparation** can begin as soon as the following inputs have been verified:
  - Assemblies are fully tested
  - Root cause analysis of assembly test is complete
  - Inputs to execute application product test meet their exit criteria
    - Test approach
    - Test model
    - Technical architecture
    - Component and assembly tested work units
  - Test environment hardware is set up
  - Test environment software is configured
  - Test execution tool is installed and tested
  - Version control tool is installed and tested
  - Test data is obtained
  - Databases are populated
  - Responsibility is established for introducing code into environment
  - Promotion procedures are refined
  - Responsibility for running batch jobs is defined
  - Technical support is dedicated to the test environment
  - Standards for metrics collection are communicated to test team
  - Standards and procedures for application product test are communicated to test team
  - Test team is trained in test procedures and tools
- **Product Test Execution** may begin as soon as a baseline tested build of the release is passed by the baseline integration testing lead. Test execution is not be complete until all product test scripts have been executed once with no errors.
  - Actual results match expected results
  - All conditions successfully tested
  - Application product test packet has been reviewed following test by application product integration testing lead
  - Application product test packet has been reviewed following test by original analyst
  - Sign-off sheet is filled out completely
  - Data and migration requests are submitted as necessary
  - Test cycles and test conditions are updated with testing status

- Status of product is updated
- Checklist cross-references are updated
- Collect appropriate metrics
- Final sign-off by test management

#### ***Major Inputs***

- Status updates from the application development team and baseline integration testing lead
- High level Workplan

#### ***Major Outputs***

- Entry Criteria for product test to be started
- Exit Criteria for product test to be considered complete

### **3.2.6. Test Script Generation and Scenario Planning**

#### **3.2.6.1. Define High-level Test Conditions**

The purpose of this task is to define business and technical events which can then be decomposed into high level test conditions.

It is important that the defined test conditions address broader concerns than the Component, Integration and baseline tests which will have already been executed and ensure that the system's business processes are operating as specified in SFA requirements and functional specifications for the release.

The following types of test conditions should be identified :

- **Business Event Conditions** - Testing of business processes includes more than the system code. User Procedures, workflows and other process definitions should be used to help define test conditions that emphasize testing the systems business events from the user or business perspective, rather than testing every possible condition from the design perspective.
- **Technical Event Conditions** - Testing of technical aspects of the system including :
  - Error/ Restart/ Recovery** - Ensuring that the system is able to identify a technical error and that it includes a mechanism for recovery or restart at the point where the error occurs.
  - Performance** - Ensuring that the system meets expectations for performance. It is critical to identify potential performance problems with the network, the mainframe and client machines
  - Stress** - Ensure the accuracy and efficiency of the system when operating on large production file volumes and under extreme circumstances

**Interface** - Ensures that all external interfaces work from the technical perspective (i.e. valid information is successfully transferred across the interface). Note that Business Event Conditions should ensure that interfaces are tested from the functional perspective.

**Operations** - Ensures that all Operator applications and utilities work according to Operator requirements

### **Major Inputs**

- functional documentation
- SFA requirements and functional specifications for the release
- User Procedures
- Workflows
- Operator application requirements

### **Major Outputs**

- High-Level Product test conditions

#### **3.2.6.2. Cross Reference Test Conditions to Requirements Specifications**

The purpose of this task is to cross reference the high-level test conditions developed in the previous task to SFA requirements and functional specifications for the release to ensure that the complete scope of the release will be tested.

Cross references will ensure that the test conditions emphasize the user/business perspective, rather than testing every processing condition from the design perspective. The system should be proven reliable in response to all normal and critical SFA business events, including business events which results in invalid input to the system.

### **Major Inputs**

- High-Level product test Conditions and expected results
- functional documentation
- SFA requirements and functional specifications for the release

### **Major Outputs**

- Cross Referenced and reviewed high-level test conditions

#### **3.2.6.3. Develop Product Test Cycles**

The purpose of this task is to define product test cycles which are modular and large enough to include a meaningful set of functions or business processes to be tested, while being small enough to be easily repeatable for individual changes to functionality.

Modular and structured product test cycles will facilitate reuse of the test model within the current test (e.g. simple functional cycles maybe reused for baseline test, technical test, etc. ), but will also make the test model as a whole expandable and repeatable for future releases. All cycles, when combined, should meet the defined objectives and scope of the product test.

**Major Inputs**

- High-Level Product test conditions
- Product Test Objectives, Scope and Risks

**Major Outputs**

- Product Test Cycles

**3.2.6.4. Group Test Conditions into Cycles**

The purpose of this task is to group the define high level product test conditions into the defined product test cycles.

Related test conditions are grouped into test cycles according to the business processes that the system supports to facilitate test execution. Grouping related test conditions into cycles reduces the possibility of obscuring test results with unrelated test conditions. Test cycles also help to set limits on the size of tests, thus reducing the complexity of reviewing and verifying the expected results.

**Major Inputs**

- Product Test Cycles
- High-Level Product Test Conditions

**Major Outputs**

- High-Level Product Test Conditions grouped into Product Test Cycles

## 4. APPENDIX

### 4.1. Testing Phases

#### Test Planning

The test planning phase involves developing a workplan to support all activities within the test stage for the application being tested. The team members and roles are identified, and a high-level plan is developed to show when each of the activities will commence and complete. The designated testing teams are responsible for planning, preparing/developing and executing the unit and string tests. At this step, the test conditions, cycles and scenarios have not yet been developed, so the plan probably will not specify individual responsibilities relating to the development and execution of tests. The responsibilities will be assigned at the end of the design the test phase.

Planning the test also includes locating and inspecting all the documents and information necessary to design, develop and execute the unit and string tests. The program specification does not have to be finished for unit test modeling to start. However, the two should be developed in concert and the program specification should be completed prior to the start of the unit test plan.

#### Test Development

The test development phase combines two distinct activities. They are test design and development of the test model. They are discussed in detail below.

Test design includes the definition of test cycles, conditions, and scenarios from the business requirements and business case documents, and development of the test plan. The test plan is a "workplan" that details assignments, development dates, and execution dates of the test model. Early test design will ensure that the test model is thorough and that all technical environment requirements are finalized so that the test environment can be ready in time for test execution. A well-designed test will also establish confidence in what is being tested. The test plan will document what is to be tested, where it is to be tested, and what progress is being made; in addition, it helps determine when testing is complete.

#### Determine test conditions

Test conditions identify and document what to test to ensure that all logic branches are tested. A copy of the test conditions should accompany each unit and string that is developed. When each test is complete, every condition should either be checked or marked as not applicable. Test conditions are slightly different for unit tests than for string tests as described below:

### *Unit Test*

Test conditions are derived from the technical design specifications that were developed to meet the business requirements and are used to validate unit functionality. Unit test conditions should cover every statement in the program. All logic paths should be exercised including every WHILE loop, FOR loop, IF statement, and MOVE statement. Some other typical test conditions include high values, low values, empty files, end-of-file and error conditions. For windows based testing, GUI controls (such as drop-down boxes, push buttons, etc.) screen navigation and field exits will need to be validated.

### *String Test*

Test conditions should be derived from the requirements of the interface defined in the program specification and technical design. Test conditions should prove that related modules or windows can pass data back and forth. Every variation of interface communication, as well as communication error processing must have a test condition.

### Determine test cycles and scenarios

Test cycles and scenarios are logical groups of related test conditions and can be grouped according to business functions, processes or business cycles (i.e., cycles could be claims processing, exception resolution, or eligibility inquiries).

A test scenario describes the execution of a series of functions to be performed in chronological order. Scenarios are not applicable to the unit test since this test focuses on specific components. They are, however, applicable to the string test. Scenarios are used to ensure that the developed components can fulfill the designs that were developed to meet the specified business requirements.

Cycles should be designed to be modular and large enough to include a meaningful set of functions or business processes while being small enough to be easily repeatable for individual changes. Modular, structured cycles will allow the test team to reuse portions of the test model within the current test (for example, simple functional cycles may be reused for regression testing, technical testing, etc.), and also provide flexibility so that the test model, as a whole, is expandable and repeatable for future releases.

In the early cycles, test only the basic or mainline logic of the work unit. Subsequent cycles should test the exception logic and error logic paths. Every condition to be tested should be included in at least one cycle.

All cycles, when combined, should meet the following objectives of the test:

- Ensure proper communications between project components,
- Ensure the developed system complies with the documented program specifications;
- Test special end of cycle or time-dependent processing, as appropriate, including:
  - ⑥ End of day
  - ⑥ End of month
  - ⑥ End of fiscal period
  - ⑥ End of year

When designing cycles and developing the test execution schedule, consideration must be given to the balance between sequential and concurrent cycle execution. Sequential execution allows for continuity of data across business processes and streamlined, systematic detection of errors. Concurrent execution allows for multiple cycles to be executed simultaneously, providing for a more leveraged and independent execution -- reducing the risk of a single problem area stopping all execution. Cycle design should also account for other scheduling dependencies, such as when different data is loaded.

#### Complete test plan

The test plan consolidates and summarizes the information developed in the previous steps. The test plan will also include a bar chart that shows all test cycles scheduled dates for development, execution begin and add end dates, and the interdependencies. This will clearly depict which cycles can be executed concurrently, and which must be run sequentially. The chart will be updated to show actual dates as cycles are completed. This provides for easy monitoring of the test progress and simplifies the process of rescheduling scenarios and resources when required.

Once the conditions, cycles, and scenarios are identified, the test model can be developed. Test scripts are designed and created, including the steps, inputs, and expected results. This model will be used not only for the planned test execution, but will be maintained for regression testing as the project capabilities are modified.

A thorough, structured, well documented test model will help ensure adequate coverage and repeatability. Since test model preparation may be concurrent with changes to the environment, care must be taken to ensure that ongoing design changes are communicated to the test team by the project development teams.

The test model must be complete prior to beginning execution. The majority of the test model should be completed after business requirements are stable, to minimize maintenance as requirements are modified. However, it may be beneficial to create the more detailed test model components, such as test scripts and test data, later in the

development process. These components are reliant on detailed specifications, such as window layouts and reports, which may not stabilize until late in the development life-cycle. For package solutions, such as front-end provider practice management systems, window layouts and dialogue flows will be stable.

### Develop test scripts

Test scripts are a set of detailed instructions that describe the users' interaction with the system. The test cycles, scenarios, scripts, conditions, and data will be modular and independent of each other in order to facilitate test execution and increase maintainability and repeatability. These scripts may be automated, using a testing tool, to shorten the time needed to test and regression test. Scripts will only apply to testing of on-line (mainframe screens and GUI windows) work units. Batch work units will have job scheduling instructions (e.g. JCL) that will describe the steps for executing the application.

The test model (test data, scripts and expected results) will be designed so that all tests can be repeated. Repeatable tests are necessary for the following reasons:

- To capture the functional and technical expertise involved in creating a test model for use in future testing;
- To reproduce any error found, for debugging purposes;
- To reproduce the entire test, once an error is found and corrected, to ensure that the correction has not introduced additional errors (regression testing);
- To save considerable future work developing a test model for any subsequent releases, if applicable.

Script development is described below for unit and string tests.

#### *Unit Test*

The level of detail in the component test scripts must be detailed enough to ensure that they may be reused and maintained.

#### *String Test*

The string test scripts should consist of the steps necessary to pass information from one component to another or from a component to the technical architecture. The string test scripts can, with modifications, be based on the unit test scripts. In the unit test, script execution tested the component up to the interface. By modifying the unit scripts, they can be reused to test the interface in the string test.

### Develop expected results

The expected results completely describe the expected outcome of the test; thus allowing the test executor to easily determine whether the test is successful, and to identify problem incidents as necessary. A test execution schedule will be developed that will detail the timeframes for executing the scheduled passes of each test cycle. This schedule will demonstrate the dependencies between cycles and display changes as execution is adjusted and completed.

Expected results must be prepared after the test data has been developed but before the testing begins.

### **Test Execution Phase**

The test execution phase involves executing each of the test scripts and cycles and comparing the results. It is in this phase that the actual testing of the application occurs. Execution of the scripts sometimes occurs more than once during the test execution phase, since discrepancies are identified and fixed during this phase. These scripts are executed until no discrepancies are found.

Execution of the unit and string tests will be done by the development person doing the coding. Key steps within test execution include:

- Accept components that have met applicable exit criteria;
- Execute test scenarios;
- Verify test results;
- Log the discrepancies;
- Identify the cause of the defect;
- Modify test model as appropriate;
- Re-test fixes;

Accept components that have met applicable exit criteria

At this stage, it needs to be confirmed that all required resources are in place for the test environment. This environment is established and maintained by the Application Architecture team and dedicated test resources. Prior to completing the string test of the configuration, the conversion test should be completed.

#### Execute test scenarios

Testing is executed cycle by cycle, as outlined in the test plan. In the early cycles, test only the basic or mainline logic of the work unit. Subsequent test cycles should test the exception logic and error logic paths. Every condition to be tested should be included in at least one cycle. Unit test execution can begin when the test model for a specific unit is

complete. String tests can begin when the model is completed for the applicable components that make up the string test.

### Verify test results

Actual test results are compared to expected results. The actual results consist mostly of files, database tables and screen or window images. A testing tool will be used, where appropriate, to automatically compare the actual results to the expected results.

To ensure accurate comparisons of actual versus expected results, the following should be done:

#### *Unit Test*

- Output records should meet the requirements of subsequent programs and should reflect all record types in a logical sequence.
- The unit test for a client program should send messages to a component-tested server program or to a test stub simulating the actions of the server program. Testing will require a check of the output messages, and the sending of test reply messages back to the client program. Test the client program with both valid and invalid replies.
- The unit test for a server program tests the response to messages sent from the client program (or test stub). Check the messages returned to the client program (or test stub) against the expected results.

#### *Unit and String Tests*

- Verify that database fields are correctly added, updated or deleted.
- Edit the records and verify them byte for byte against the record layouts and expected test output. If several records with identical formats are created, check at least two of them.
- Verify that all lines have been printed on reports. Check editing for significant digits, decimal points, monetary signs, plus or minus signs, and spacing. Verify the page count and the place at which page breaks occur. Check accumulations and totals for foot totals, as appropriate. Verify control breaks and totals.
- Verify that all window output formats are properly displayed.

### Log the discrepancies

Discrepancies should be tracked by the developer who is coding and testing. Logging should include a count of problems by type (e.g., design errors, code errors, technical problems, etc.). The process for logging problems encountered during the unit and string tests is less formal than the process used for the System test. A less formal process is needed due to the trade-off between time required to manage the process with so many things changing during these phases. If a problem cannot be resolved by the developer, it should be documented and reported to the test methodology expert. See section VI. F. for how an unresolved problem will be handled.

### Identify the cause of the defect

#### Unit Test

The majority of problems encountered during unit testing are the result of coding errors. Coding errors include both incorrectly implementing the design and coding incorrect logic. Problems may also result from system errors or design errors.

#### String Test

Issues encountered in the string test will most likely be defects in the code or errors in the string test model, but could also be problems with the technical design or architecture. Defects must be properly documented, raised, and assigned to the appropriate personnel for correction. This may entail returning the offending component(s) to the development or unit test phase.

### Modify the Test Model as Appropriate

Changes resulting from identified discrepancies are not solely restricted to application errors; it may be necessary to change reports, the system design or systems software. Depending upon those changes, it may be necessary to modify parts of the test model. The test model may also require changes if the discrepancy resulted from errors in the expected results or test data.

### Retest fixes

As the fixes are completed, a retest will need to be done. A cycle should not be executed again until all known fixes affecting the components of that cycle are complete. These fixes may have originated from an earlier pass of the same cycle, or from another cycle that has one or more components in common. The application teams will group related fixes together so as to minimize the delay in executing the next pass of the test cycle. The cycle is complete when the expected results are obtained. The unit/string developer should have the team lead sign-off on the completed cycle.

Entry and exit criteria have been defined in this document for beginning and ending the test execution phase of each testing stages. These criteria are a part of phase containment, which attempts to contain errors to the appropriate testing phase. For example, all programming logic problems should be solved in unit test and should not appear in integration test. If a unit has not been correctly unit tested, it should not be accepted into integration test, but should be fixed and correctly unit tested by the responsible developer.

### **Test Support**

The test support phase consists of the technical and functional support provided to the testing teams. Technical support includes supporting the testing environmental, application support, and providing assistance with test data. Functional support will be provided by the application development organization. This would include answering design questions, answering business questions, and providing support on developing accurate business scripts.

For each of the testing stages (Unit, Integration, User Acceptance, and Performance Testing), this document will detail the purpose, objectives and scope of the phase and entry/exit criteria. In addition a listing of tasks and responsibilities will be included, broken out by phase ( test planning, test development, and test execution).

### **Other Testing**

Periodically, there will also be the need to do regression and acceptance tests.

#### Regression testing

If a cycle has been signed off on and a change is made to another part of the system, then a regression test may need to be run to assure that all previously tested aspects of the system remain functional after the change. Regression testing minimizes the risk of accidentally changing something when doing a fix on another area. The test model should be well documented and maintained to allow for regression testing.

Regression testing should be used when there is a high risk that new changes may affect unchanged areas of the application system. In the developmental process, regression testing should occur after a predetermined number of changes are incorporated into the system. The determination as to whether or not to conduct regression testing should be based upon the significance of the loss that could occur due to improperly tested applications. For example, if a code change is made to a module during system test, and that change potentially impacts a business function across applications, the module must be unit tested and string tested again.

#### Acceptance Testing

SFA will develop new releases and patches to identified code errors on an on-going basis. SFA will do an alpha test prior to releasing the changes to SFA. These releases and patches should initially be given to the SFA version control coordinator. The version control coordinator will place the new release under version control and notify the technical architecture team and the applicable unit and string testers for them to do an acceptance test.

## 4.2. Incident Reporting

The purpose of this section is to present recommendations for logging and resolving incidents found during one of the testing or training phases in the application development lifecycle. The process should provide the following features:

- Be flexible enough to address incidents found in any post-unit test development stage (integration testing, performance testing, acceptance testing, and training)
- Provide an automated tracking and reporting capability
- Ensure thorough documentation of incidents and subsequent resolution
- Provide a mechanism to ensure that all incidents, regardless of severity level, are dealt with in a timely manner
- Provide communication guidelines to ensure that all interested/involved parties are aware of the status of a given incident
- Provide controls to ensure that incidents are accurately reviewed and resolutions approved

An incident occurs when a business or technical requirement is not met during integration test, user acceptance test, performance/volume testing, or training conduct. Incidents can also include general implementation issues that are preventing or prohibiting completion of a testing a training segment.

Benefits of a well defined, tightly managed incident tracking process include:

- Assists with identifying potential schedule slippages or the need to develop alternative implementation paths
- Verifies that quality controls are in place; provides mechanism for capturing development metrics
- Assures that incidents are dealt with in a timely manner
- Assures that incidents are not overlooked and incorrect applications migrated to the next testing/training segment or even production
- Verifies that all components are ready for implementation

### Tools

In order to accurately track and report the status of incidents, it is recommended that an automated tracking system be used. Packaged software exists to facilitate this process, however, many development “shops” opt to design and develop a tailored application using tools such as Excel, Access, or Lotus Notes. A database application package such as Access or Lotus Notes is preferred because it allows a more user friendly data entry mechanism and flexibility in viewing data (online vs. hardcopy reports). If significant remote testing will occur, Lotus Notes is the more viable option. The tracking and reporting application should capture at a minimum the following information:



<b>What</b>	<b>Description</b>
Reported by	The individual who discovered the incident
Date reported	The date the incident was discovered
Application	Application area that contains the incident (if known). Examples include: GMS, BEST, Product Manager
Testing Phase	The testing phase (or training) where the incident was discovered
Scenario or Script no.	A cross reference to the script, cycle, or scenario that was being executed when the incident occurred
Brief Description	Short (one sentence) description of the incident
Long Description	Long description of the incident
Classification	Classification of the incident as either a bug/problem, enhancement, or design change
Estimate	Estimated number of hours it should take to resolve the incident (should be entered by the application development team)
Assigned to	The name of the analyst or developer who will investigate incident
Date Assigned	Date the incident is assigned
Resolution	Comprehensive description of the resolution to the incident
Resolved by	Individual who resolved the incident
Date resolved	Date the incident was resolved
Actual	Actual number of hours spent to resolve the incident (including investigation, analysis, and coding/unit test)
Retested by	Individual responsible for retesting the script, cycle, etc. where the incident occurred
Date retested	Date the incident was retested
Incident status	Open, Assigned, Resolved, Approved
Incident Number	Identifies the incident with a unique number to the system
Incident priority	H/M/L

## Processes and Procedures

The processes and procedures defined in this section of the testing approach assume that an application such as the aforementioned is developed and in place. Figure A presents an overview of the incident tracking and reporting process. Specific steps of the process are detailed following figure A. Each task identified is a high-level step and may in fact be comprised of multiple detailed tasks. The diagram is not meant to stand-alone and should be reviewed with the descriptions presented which follow the diagram.

Step	What	Who	Description
1	Incident Reporting form is submitted	Integration Test Team member; training participant	Individuals who do not have access to the incident reporting and tracking system should complete an incident tracking form and submit the form to the Integration testing team. A sample form is included in Page 1. Supporting documentation such as screen prints, expected and actual results, etc. should be attached to the form to assist with issue resolution.
2	Log Incident in Incident Tracking System	Integration Test Team	It is recommended that the integration testing team retain responsibility for updating the incident tracking system. The following items should be entered when an incident is initially logged: Reported by, Date Reported, Application, Scenario/Script no., Testing Phase, Estimated Hours, Brief Description, Long Description, Classification, Status = O, Incident number, and Incident priority. Any supporting documentation should be copied and filed.
3	Review Incident	Integration Test Team	The integration testing team should review all incidents regardless of the testing phase/training phase in which they occurred. It is the responsibility of this team to discern whether adequate information exists to assign and resolve the issues and to gather additional information if necessary.
4	Discern Incident Reason	Integration Test Team	The integration test team must discern whether the incident is in fact a bug, design change request, or system enhancement.
5	Review by project management office	Project Management Office	Assuming the incident is in fact an enhancement or design change, the project management office should review the request against project plans and ensure that a cost/benefit analysis was performed and that the modification is justified and scheduled appropriately.
6	Make Assignments	Project Management Office and Integration Test Team	Defects and approved modifications should be assigned to an appropriate analysts for investigation. The Incident Tracking System is also updated to reflect this assignment.
7	Update Incident Tracking System	Integration Test Team	If an enhancement or design change is denied or deferred, the Integration Test Team will update the Incident Tracking System and communicate the decision and rationale for the decision to the appropriate parties.
8	Update Incident Tracking System	Integration Test Team	This update occurs in the event that a corrected incident undergoes integration testing and has not been resolved adequately. The log is updated to show the failed resolution and its status remained at assigned.

9	Enter Resolution in Incident Tracking System	Integration Test Team	This update occurs when a corrected incident is resolved and retested successfully. The resolution is captured, individual responsible for the resolution, date of resolution, date of retesting, and the status is set to R.
10	Obtain Approval for Resolution	Integration Test Team Leader	The integration test team leader will discern whether an incident is truly resolved.
11	Incident is closed	Integration Test Team Leader	If the integration test team leader approves an incident's resolution, the incident tracking system is updated to reflect a status of Closed ("C").
12	Integration Testing team resolves	Integration Test Team	If the integration test team leader does not approve an incident's resolution, the integration team must discuss the incident and recommend a course of action.
13	Update Incident Tracking System	Integration Test Team	This update occurs in the event that a corrected incident passes regression testing in integration test but does not pass acceptance test. The log should be updated to show the failed resolution and its status remained at assigned.
14	Enter Resolution in Incident Tracking System	Integration Test Team	This update occurs when a corrected incident is resolved and retested successfully in the acceptance test environment. The resolution is captured, individual responsible for the resolution, date of resolution, date of retesting, and the status is set to R.
15	Obtain approval for resolution	Integration Test Team Leader	The integration test team leader will discern whether an incident is truly resolved.
16	Incident is closed	Integration Test Team Leader	If the integration test team leader approves an incident's resolution, the incident tracking system is updated to reflect a status of Closed ("C").
17	Integration Test Team resolves	Integration Test Team	If the integration test team leader does not approve an incident's resolution, the integration team must discuss the incident and recommend a course of action.

As identified in the process flow diagram and the detailed descriptions above, the integration test team should track and report the status of all incidents. This responsibility spans beyond the boundaries of the integration testing segment in the development life cycle. The integration testing team should work with the business enablement team to facilitate user acceptance testing and training activities. The integration testing team's knowledge of test scripts and business cycles will jump start and supplement subsequent testing and training activities. Additionally, centralization of control and monitoring responsibilities will ensure that management maintains an accurate view of testing progress and the production readiness of application components. It is important, therefore, that the integration testing team keep the Incident Tracking System updated and frequently communicate incident status to application development teams, the integration team, business enablement team, and the project management office. Additionally, if concurrent development efforts are underway, the integration testing team may have other project teams who require status. Communication can be enhanced by supplementing integration testing team meetings and integration teams with frequently distributed incident tracking logs.

### Summary

While implementation of these processes and procedures will help achieve the before mentioned benefits, the incident reporting tasks identified here should be integrated with testing phase containment (entry and exit criteria) and migration controls to ensure comprehensive quality controls are in place through-out the development lifecycle. Full implementation of these various tools and techniques will assist in assuring quality, defect free, timely implementations.

### 4.3. Problem Detection

This process description assumes that you are familiar with the system integration & testing model strategy, configuration management change control process, version control tool and the source code repository structure.

#### Source of Incident Reports (IR)

The majority of incident reports are created during development and during production. However, incident reports can come from other sources.

#### Development

The majority of incident reports will be created during development and testing. A tester or developer will locate a problem and open up a incident report and document that problem.

#### Post Deployment Maintenance

The customer service group receives calls and creates incidents to be investigated in the production environment. As part of investigating the incident, someone may identify a change to an application. This change will require a incident report to be opened.

Within the application databases, there may be data in tables that is under version control. The reason for the control on the data is because of the dependency between the data and the source code in the application; if the data changes, then the source code changes.

Data changes are sub-changes to the code changes. *Note:* It is important to remember that the master database is the end result of changes to the application. When a change has been successfully migrated to production, the changes are then made to the master database. The steps must occur in this order to ensure that the data is kept in sync with the source code.

#### Database Structure

Database structure changes come from two sources; the database team and the development team. The database team may want to modify the structure of the database (tables, indices, etc.). To do this, they would request a change and apply it to development. When they are certain that the change will be successful, it is migrated through the change process, the same as any change.

Database structure changes identified by the database team are entered directly into CM tool. They do not go through the customer service group and they are not sub-changes.

The development team may identify database changes when making a code change. These changes are related or sub-changes and are linked to the original (parent) change in the incident report log. The changes can then be migrated together so any database change flows with the code change. The application will work correctly as the two changes are synchronized. Database structure changes identified by development are sub-changes to the code changes.



### Database Files in version control tool

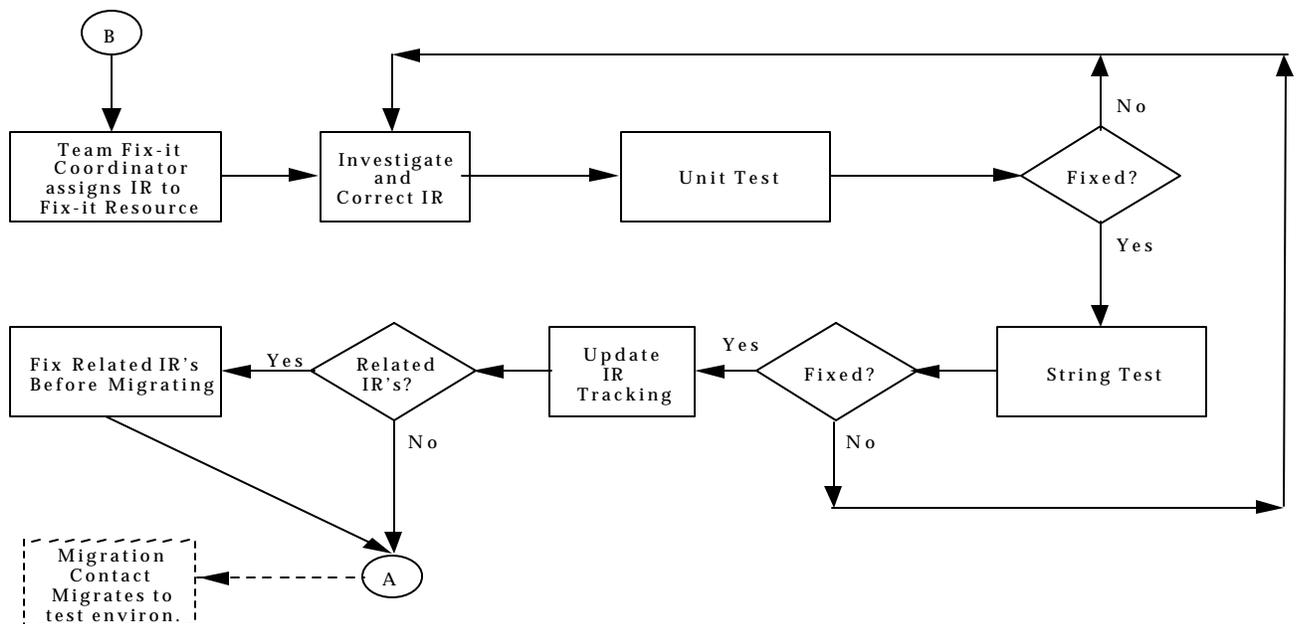
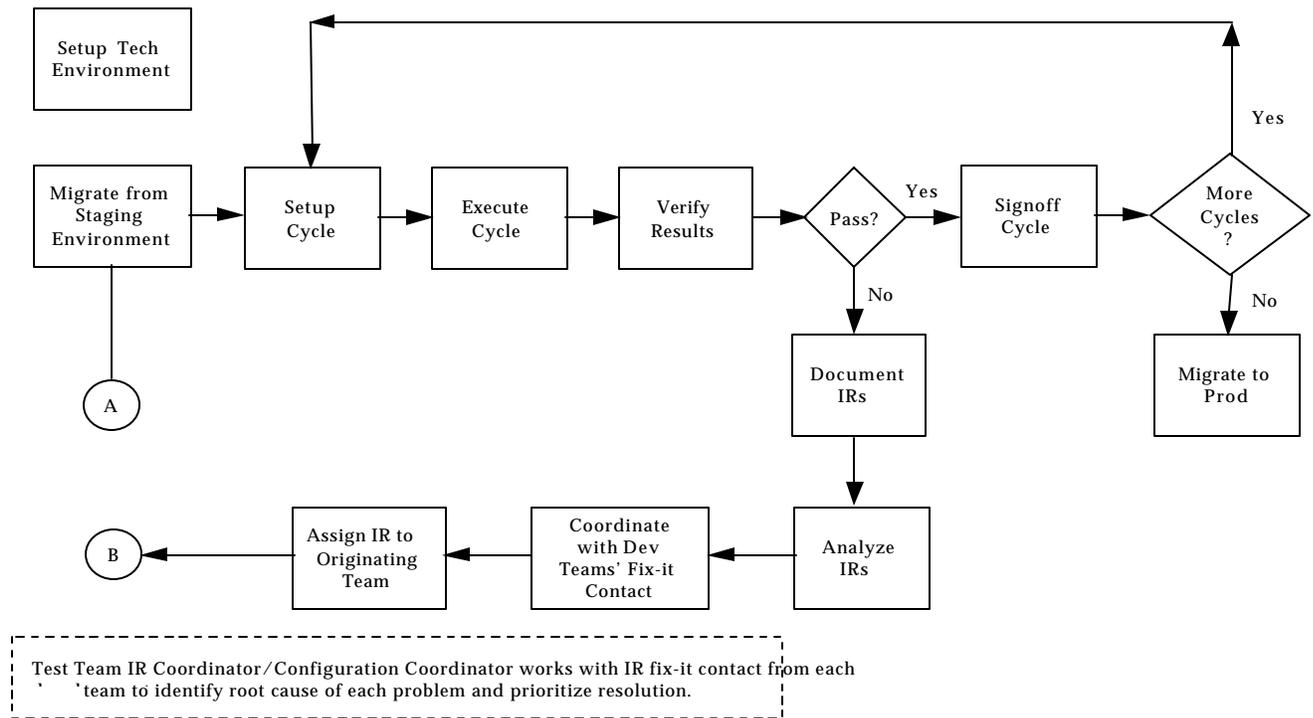
Ideally the project should keep all files for the application under version control in version control tool. However, practically this is not possible for the databases. Some databases are too large and for some it is not practical. The process has been documented as if version control tool is used but this is not always the case.

The development team may identify modifications to error string or messages when making a code change. As well they may require new error or messages to complete their code changes. The development team requests a change, noting the parent incident report. The changes can then be migrated together. The application will work correctly as the two changes are synchronized. Error and message changes are sub-changes to the code changes.

### SQL / Service

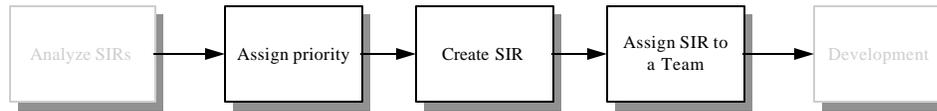
The development team may identify modifications to SQL services when making a code change. The development team requests a change, noting the parent incident report. The changes can then be migrated together. The application will work correctly as the two changes are synchronized. SQL/Service changes are sub-changes to the code changes.

Problem Detection General Flow



<b>Document Incident Report</b>	Document the fix or enhancement to a incident report. The change is then analyzed by the Test Team Fix-It Coordinator/CM Coordinator.
<b>Analyze Incident Report</b>	<p>The Test Team Fix-It Coordinator/CM Coordinator investigates the incident report. There are three possible results of the investigation:</p> <p>No change required. The resolution to the problem does not require a change to the application code or the environment.</p> <p>The solution to the incident report requires a change to the application or the environment. The change is either a Fix or an Enhancement;</p> <p>Fix - The fixer needs to modify the application to correct a problem. The Fix occurs when Development did not implement an approved design correctly or the change caused an error in the application</p> <p>Enhancement -The fixer needs to modify the application to add additional functionality to the current design or to implement the current design in a different way. This is a change to the current design.</p> <p>During the analysis of the incident report the Test Team Fix-It Coordinator/CM Coordinator will coordinate with the Development Teams' Fix-it Contact, the Test Team Fix-It Coordinator/CM Coordinator will assign the incident report to a Fix-It Team Coordinator, and the Team Fix-It Coordinator will assign the incident report to a Fix-It Resource.</p>
<b>Investigate and Correct Incident Report</b>	The Fix-it resource investigates and then corrects the incident report.
<b>Unit/String Test</b>	Incident reports that have been Fixed are tested to make sure the changes have correctly implemented the program specifications. At the end of unit/string test, all segments of code should have been exercised and proven to meet the specified functional and quality requirements. Because of the table-driven nature of the SFA system, each rule represents a branch of processing logic and therefore must be tested to ensure that the incident report results are achieved. If the incident report is not corrected, it must be investigated and fixed by the Fix-It Resource and then Unit/String tested again. This process continues until the error has been corrected.
<b>Related Incident Reports</b>	Before a component can be migrated all incident reports associated with that component must be fixed.
<b>Change Migration</b>	Incident reports that complete a stage in the Development lifecycle process are migrated to the next level.
<b>Integration Test</b>	Incident reports that have been Fixed are tested to make sure the change a) corresponds with the design and b) did not impact other parts of the application (Regression Testing). For data or database structure changes, the changes are applied to the Integration Test environment.
<b>Failed Incident Reports</b>	Incident report that do not pass a stage or are not accepted by the Test Team Fix-It Coordinator/CM Coordinator. These changes are returned to a status of <b>Assigned</b> and are re-addressed by the Team Fix-it Coordinator.
<b>Production</b>	The change is ready for production and is part of a release or version of the executable.

## Create Incident Report



Assign Priority	Determine the Severity and Importance of the incident report..
Create Incident Report	To submit a Incident report record, follow these steps: Submit change request.
Assign Incident Report to a team	Assign the incident report to a team who will be responsible in resolving the error.

Field	Definition
Team	Team that identified the problem. Limit the selection to the following: Integration Conversion Technical Architecture Testing
Project	Name of project that is the probable source of the incident report
Initiator	Select cycle coordinator's name
To Be Resolved By	- If internal problem, select name of cycle coordinator - If external problem, select appropriate contact from the team referenced in "Project" field
Problem Type	:Code Error - program working incorrectly Design Error - system design doesn't satisfy business requirements Data Prop Error - data conversion problem SFA Bug - if SFA does not work as it should Processing Rule Error Scripting Error - script or expected results were wrong Technical Error - such as Oracle, UNIX Data Error
Researched By	This field is selected by the project team's fix-it coordinator to document the name of the person who will investigate/fix the problem.
Tracked By	Select cycle coordinator's name
Phase ID	Select the phase the incident report was located in
Task/Script	Enter cycle number, pass number, script number (ex. Cycle 2 pass 3 script 17 would be entered as C2_P3_SC17)
Problem #	
Create Date	
Actual Closed Date	This appropriate date will be entered when the problem is closed.
Estimated Closed Date	Estimated time of the problem closing.
Date Required	Date when the problem needs to be resolved - based on when the next pass is scheduled.
Status	Open - Problem has been identified Assigned - Test Team Fix-It Coordinator/CM Coordinator assigns the incident report to the

	<p>appropriate fix-it team</p> <p>In Analysis - Test Team Fix-It Coordinator/CM Coordinator assigns the incident report to a fixer on the team</p> <p>In Process - This step indicates that a proposed course of action has been determined and that the fixer is in the process of making the change</p> <p>Fixed - Once the fixer has tested the fix and is satisfied that a incident report has been resolved</p> <p>Ready for Migration - Once the problem is fixed and ready for migration, the team's migration contact will migrate the effected components to the next phase of testing</p> <p>Ready for Retest - Once the incident report has been migrated, the components are ready to be retested. A cycle will not be executed again until all known fixes affecting the components of that cycle are complete.</p> <p>Reopen - The incident report has been fixed but a problem still exists</p> <p>Closed - Once the retest has verified appropriate results, the problem's status is changed to "Closed". Only the team responsible for the retest can determine when a incident report should be closed.</p> <p>Deferred - To ensure that all modifications to the system are documented, any change requests or problems to be resolved in future phases should be assigned a status of "Deferred". This classification will help ensure that system enhancement requests are captured and will be worked on at a later date.</p>
Severity	Select the appropriate severity 1-5 (1 = most severe, 5 = least severe)
Description	Short description of problem
Title	Title of incident report
ATC	The Actual hours to complete the change (ATC). This time can include meetings, revising documentation, programming, performing unit testing, etc.

### Attaching Files to Incident Reports

Each incident report created due to a error found after unit/string test will require a file to be attached to it. After unit/string testing, An automated test tool will be used to run test scripts automatically. If an error occurs during the script execution, automated test tool will be able to save the test script, with it's actual and expected results, as either a text file or bitmap file. After the automated test tool file has been saved, it should be attached to the incident report that was opened due to the error.

### Attaching Modules to Incident Reports

This allows version control tool to exchange and store information. The tool provides two ways to create associations between incident reports and modules:

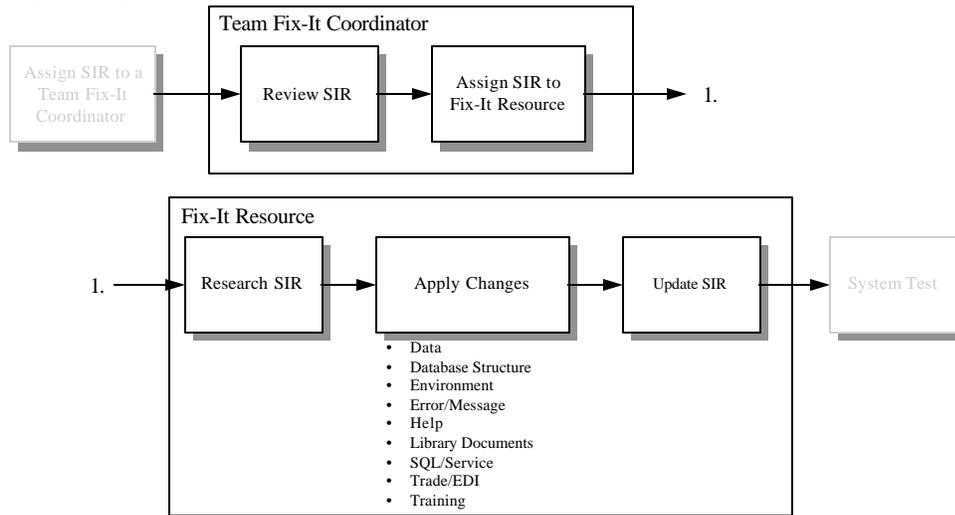
A single incident report record can be associated with many source modules, or  
Many incident report records can be associated with a single source module.

To create the association, select one or more open incident reports. Then use the check out option to check out the relevant source modules from their version manager archives. After the modifications are complete add the new revisions to the original association.

### Notifying Resources of Incident Reports

E-mail messages can be sent automatically via a third party E-mail package when certain conditions are met.

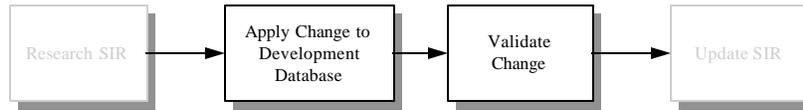
### Applying Fixes



Review Report	Incident	Before assigning a incident report to a team member, the Team Fix-it Coordinator reviews the incident reports and estimates the time to complete the change. The estimate and the date of the estimate (Complete Date) are entered in the incident report. The Team Lead changes the status to <b>Assigned</b> .
Assign Report to team member	Incident	For the current version of the application, the Team Fix-it Coordinator groups incident reports for the same module together and assigns them to the Fix-it Resource. The Team Fix-it Coordinator changes the status to <b>In Analysis</b> .
Research Report	Incident	The Fix-it Resource reviews the incident report. If the incident report requires clarification or additional details, the Fix-it Resource gathers the needed information.
Apply Changes		See the Section <i>Apply Changes</i> below. The Team Fix-it Coordinator changes the status to <b>In Process</b> .
Update Report	Incident	The Fix-it Resource moves the status of the incident report to <b>Fixed</b> . They enter the date the incident report was completed and the Actual hours to complete the incident report (ATC). The ATC includes any time spent on the incident report. This time can include meetings, revising documentation, programming, performing unit testing, etc..

### Apply Data

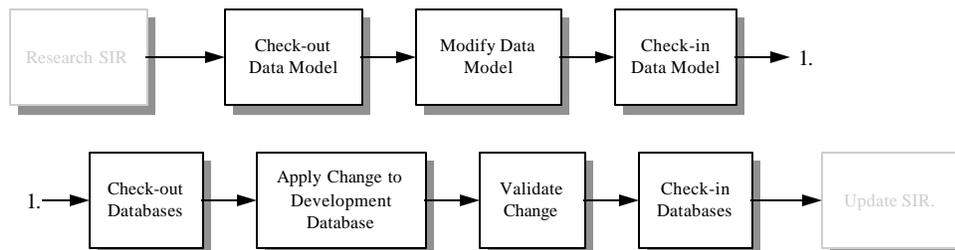
Data changes are put through the change control process to test the change and to ensure the data change remains synchronized with any code changes. When a data change has been successfully tested, then it moves to production. *It is very important to remember that the master database is the **last** database to be updated as it is the database that is copied and distributed.*



<i>Apply Change to Development Database</i>	The change is applied to the development database.
<i>Validate Change</i>	Data changes are typically sub-changes to code changes. After applying the change to development, check with the developer that the change is working correctly before updating the incident report.

### Database Structure

Database structure changes are put through the change control process for the same reasons as data changes. When a database change has been successfully tested, it then moves to production. This is the version that will be distributed when the changes are complete.



Check-out Data Model	If required, after researching the incident report, the Database Analyst checks out the existing data model from the version control tool application repository.
Modify Data Model	The database structure changes are applied to the data model.
Check-in Data Model	The new version of the data model is put back in the version control tool repository. With the data model updated, the Database Analyst can apply the modifications to the database. The Database Analyst uses the incident report number to check-in the data model and adds a comment to the file.
Check-out Databases	If required, after updating the data model, the Database Analyst checks out any databases from the version control tool application repository.
Apply Change to Development Database	The change is applied to the development database.
Validate Change	Data changes are typically sub-changes to code changes. After applying the change to development, check with the developer that the change is working correctly before updating the incident report.
Check-in Databases	The new version of the databases is put back in the version control tool repository. The Database Analyst uses the incident report number to check-in the data model and adds a comment to the file. <sup>1</sup>

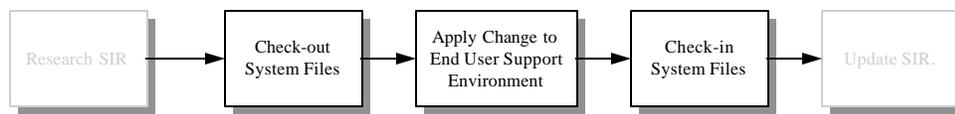
### Environment

Environment changes are unique for the following reasons:

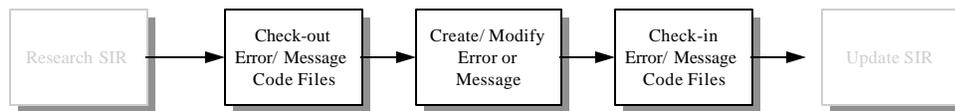
Based on the PC Configuration impacted, changes are applied to the End User Support environment, Development or System Test.

Depending on the type of change, System Testing may or may not be performed by the System Test team. However, upgrades to package software, such as MS Office or RoboHelp would not be explicitly tested. However, End User Support and a test group would test the installation and rollout of the new configuration.

Final distribution of the new configuration would be managed and completed by the End User Support Team.

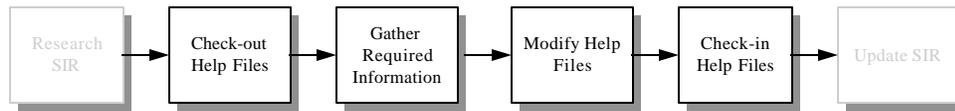


### Error / Message



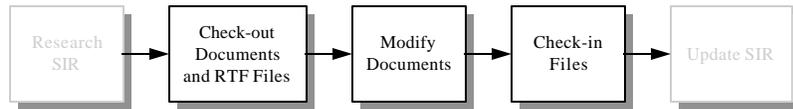
Check-out Error/ Message Code Files	If required, after researching the incident report, the database analyst checks out the existing Error and Message code files from the version control tool application repository. This step is only required for common error or messages strings or for application global strings. This step is not required for module level errors or messages.
Create/ Modify the Error or Message	With the Error Message Maintenance tool, the error or message is added to the Global database of strings or a current string is modified. After the changes are made, the code files are regenerated.
Check-in Error/ Message Code Files	The new versions of the code files are put back in the version control tool repository. The database analyst uses the incident report number to check-in the files and adds a comment to the files.

## Help



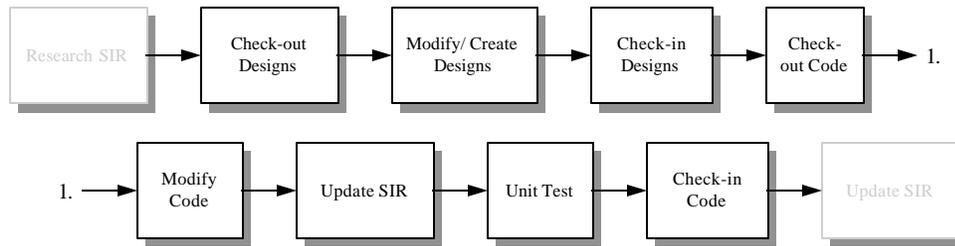
Check-out Help Files	After researching the incident report, the training analyst checks out the existing help files from the version control tool application repository.
Gather Required Information	Before modifying the Help files, the Help File developer needs to make sure that they have the latest executable or screen prints (bitmaps) for the help files.
Modify Help Files	The training analyst modifies the Help files.
Check-in Help Files	The new versions of the help files are put back in the version control tool repository. The Help File developer uses the incident report number to check-in the help files and adds a comment to the files.

## Library Documents



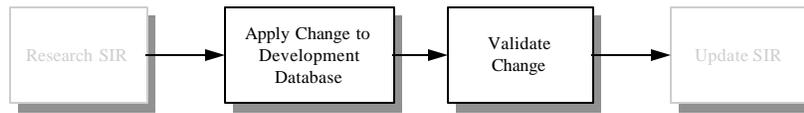
Check-out Document and RTF Files	After researching the incident report, the Fix-It resource checks out the existing Microsoft Word documents and the corresponding Rich Text Format (RTF) files from the version control tool application repository.
Modify Documents	The documents are updated based on the description for the incident report. The RTF files are created from the Microsoft Word documents.
Check-in Files	The new versions of the Microsoft Word documents and RTF files are put back in the version control tool repository. The Team Analyst uses the incident report number to check-in the help files and adds a comment to the files.

**EDI**



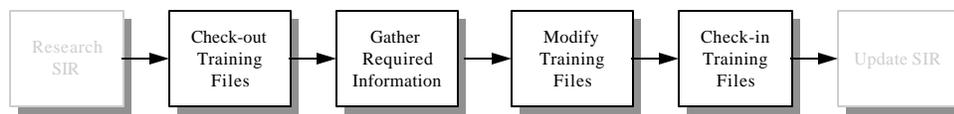
Check-out Designs	If required, after researching the incident report, the developer creates a Problem Resolution Request or a Library Action Request to check out any existing Functional or Technical Design documents from the CMS repository.
Modify / Create Designs	The team member revises the designs according to the description on the incident report. For large changes and enhancements, the Team Lead reviews the designs for completeness.
Check-in Designs	The new versions of the designs are put back into the CMS repository. With the designs complete and reviewed, the developer can begin modifications to the code. The developer uses the incident report number to check-in the designs and adds a comment to the files.
Check-out Code	The latest version of the code is retrieved from the application repository.
Modify Code	The source code is modified per the description on the incident report.
Update Incident Report	<b>Optional.</b> The developer updates the status of the incident report to <b>Unit Test</b> . This shows that the code modifications are complete and that the testing process has begun.
Unit Test	The developer tests the new changes to the application. As well, they perform regression testing to make sure that other parts of the application were not impacted. If a code review is required, the Team Lead completes the review while the change is being unit tested. The code review must be completed before the changes are checked in.
Check-in Code	The new version of the source code is checked back into the CMS library. The CMS administrator uses the incident report number to check-in the designs and adds a comment to the files.

### SQL / Service



Apply Change to Development Database	The SQL changes are applied to the development database. For a new service, an entry is added to the application's local database.
Validate Change	SQL/Service changes are sub-changes to code changes. It will be very rare to have a change to SQL or a Service without a code change. After applying the change to development, check with the developer that the change is working correctly before updating the incident report.

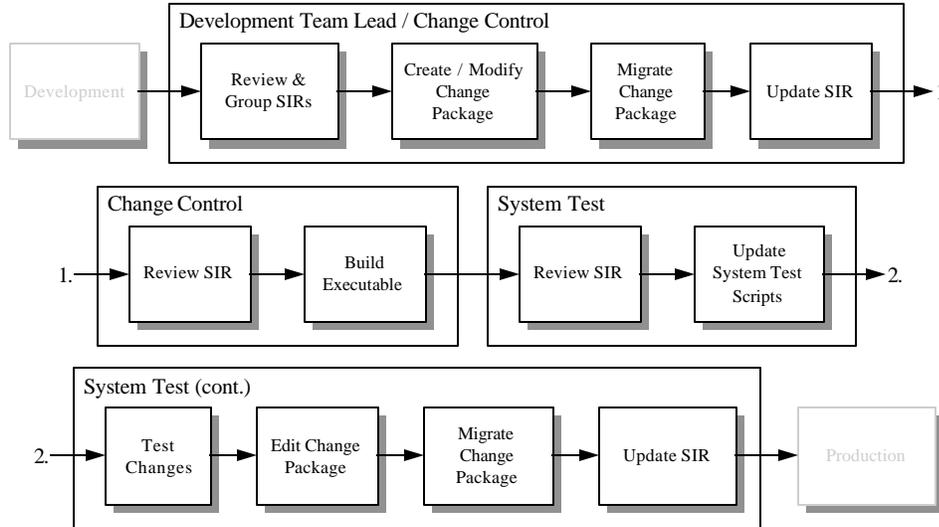
### Training



Check-out Training Files	After researching the incident report, the training analyst checks out the existing training files from the version control tool application repository.
Gather Required Information	Before modifying the training files, the training analyst needs to make sure that they have the latest executable and review the incident reports for the current version of the application.
Modify Training Files	The training files are updated to reflect the new functionality of the software. The training scripts need to be re-run to make sure data or database changes have been applied to the training database.
Check-in Training Files	The new version of the training files is put back in the version control tool repository. The training analyst uses the incident report number to check-in the training files and adds a comment to the files.

### Integration Test

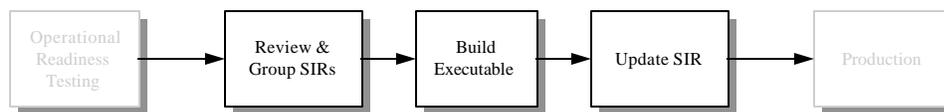
The system test process has two components; preparation and testing. The integration test preparation is completed by the development team lead. Configuration management helps the team lead by ensuring changes completed by other teams are included in the change package. When the preparation is complete, the integration test team performs the testing.



Review and Group Incident Reports	For the current version of the application, the Development Team Lead lists out all the incident reports that have been fixed by the Development Team. For 'non-Development' changes such as data or service changes, the Configuration management lead ensures that the changes are applied to the Integration Test environment.
Create / Modify Change Package	The Development Team Lead creates a change package according to the Configuration management Naming Standards. If a package already exists for the current version of the application, the existing package is modified. Changes that have been fixed, according to the incident report, are added to the Change Package.
Migrate Change Package	The Team Fix-it Coordinator changes the status to <b>Ready for Migration</b> The Change Package is migrated from Development to the Integration Test.
Update Incident Report	The status of the migrated changes are moved to a status of <b>Ready for Retest</b> .

Review Incident Report	Configuration management reviews the log to ensure that changes and their sub-changes were migrated together. They also validate that all Fixed changes were migrated.
Build Executable	Configuration management creates the executable for Integration Test.
Review Incident Report	Integration Test lists all the incident reports to be tested and reviews the changes. They research any changes that need more detail and identify any changes that they feel they can not test. For incident reports they cannot test, they are reviewed with the configuration management lead. They first try to develop a test scenario or, if there is no test scenario, the configuration management lead can decide to move the change forward. The Configuration management lead moves the incident report to a status of <b>Deferred</b> , which ensures the incident report will be worked on at a later date.
Update Integration Test Scripts	The test scripts are updated as well as an scripts required for the automated testing tool.
Test Changes	All changes are tested.
Edit Change Package	For incident reports that fail Integration Test, do not go any further in the process. They are removed from the Change Package before it is migrated.
Migrate Change Package	The Change Package is migrated from Integration Test to Operational Readiness Test.
Update Incident Report	Incident reports that are successfully tested are moved to a status <b>Closed</b> . Incident reports that fail are moved back to a status of <b>Reopen</b> to be re-addressed by the Development Team. Sub-changes are kept in the same status as their parent; if a change is tested successfully, then all the sub-changes are successful as well.

## Production



Review and Group Incident Reports	For the current version of the application, Configuration management lists out all the changes that are ready for the Production Executable (status of <b>Migration</b> ). For 'non-Development' changes such as data or service changes, the Configuration management lead ensures that the changes are applied to the Production environment.
Build Executable	Configuration management creates the executable for Production.
Update Incident Report	Update the log to record the production build. All incident reports included in the production executable for the current release are moved to a Status of <b>Closed</b> . The incident reports are updated to include the Build Number of the executable.

## **4.4. Candidate Checklists and Forms**

**Incident Reporting and Tracking Form**

**Application Folder Checklist**

**Integration Test Team Script Control Sheet**

**Integration Test Team Detailed Schedule**

**Integration Test Team Condition Tracking Sheet**

**IT Test Plan Approval Form**

**IT Test Schedule Approval Form**

**IT Technical Environment Checklist Approval Form**

**IT Test Conditions Approval Form**

**IT Test Cycles Approval Forms**

**IT Test Scripts Approval Forms**

**IT Entry Criteria Approval Form**

**IT Test Script and Cycle Execution Approval Form**

**IT Exit Criteria Approval Form**

**User Acceptance Test Plan Approval Form**

**User Acceptance Test Schedule Approval Form**

**User Acceptance Test Technical Environment Checklist Approval Form**

**User Acceptance Test Cycles Approval Form**

**User Acceptance Test Scripts Approval Form**

**User Acceptance Test Entry Criteria Approval Form**

**User Acceptance Test Script and Cycle Execution Approval Form**

**User Acceptance Test Exit Criteria Approval Form**

**Incident Reporting and Tracking  
Incident Investigation Report Form**

Incident Number: _____ Incident Priority: _____ (H/M/L)
--

Reported by: \_\_\_\_\_

Date Reported: \_\_\_\_\_

Application: \_\_\_\_\_

Script no: \_\_\_\_\_

Cycle no: \_\_\_\_\_

Testing Phase: Unit: \_\_\_\_\_ Integration: \_\_\_\_\_

Integration/Performance: \_\_\_\_\_

User Acceptance: \_\_\_\_\_

Training: \_\_\_\_\_

Incident Reason: Bug: \_\_\_\_\_ Design Change: \_\_\_\_\_ Enhancement: \_\_\_\_\_

Brief Description: \_\_\_\_\_

Long Description: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Assigned to: \_\_\_\_\_

Date Assigned: \_\_\_\_\_

Resolution: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Resolved by: \_\_\_\_\_  
Date Resolved: \_\_\_\_\_

Retested by: \_\_\_\_\_  
Date Retested: \_\_\_\_\_

Approved by: \_\_\_\_\_

### Application Folder Checklist

The Application Folder serves as the package for all testing deliverables and test results for each application. The information for each application may not all be included in a “single binder”, but should be packaged together to provide continuity of test results across each application. An Application Folder should be prepared for each test phase.

The following tables provide a checklist for ensuring that all testing-related materials are contained in the Application Folder. While this checklist is not a formal deliverable, it is recommended that the checklist be included as a table of contents for each Application Folder.

<b>Test Planning</b>	<b>UT</b>	<b>IT</b>	<b>IT Perf.</b>	<b>UAT</b>
Test Plan				
Test Plan Approval Form				
Test Timeline				
Test Detailed Schedule				
Test Schedule Approval Form				
Technical Environment Checklist				
Technical Environment Approval Form				
Test Planning meeting notes and working papers				

<b>Test Development</b>	<b>UT</b>	<b>IT</b>	<b>IT Perf.</b>	<b>UAT</b>
Test Conditions Tracking Sheet				
Test Conditions Approval Form				
Test Cycles Worksheet				
Test Cycles Approval Form				
Test Scripts with Attached Script Control Sheet				
Test Scripts Approval Form				
Application Flows				
Test Development meeting notes and working Papers				

<b>Test Execution</b>	<b>UT</b>	<b>IT</b>	<b>IT Perf.</b>	<b>UAT</b>
Entry Criteria Checklist				
Entry Criteria Approval Form				
Test Script and Cycle Execution Approval Form				
Exit Criteria Checklist				
Exit Criteria Approval Form				
Execution meeting notes and working papers				

<b>Support</b>	<b>UT</b>	<b>IT</b>	<b>IT Perf</b>	<b>UAT</b>
Daily integrated testing status meeting notes and status reports				
Incident Reporting System Reports				
Issue Tracking System Reports				

**Integration Test Team  
Script Control Sheet**

Application area being tested: \_\_\_\_\_

<b>Script #</b>	

<b>Execution Date</b>	<b>Executed By:</b>	<b>Results</b>

### Integration Test Team Detailed Schedule

Application area being tested: \_\_\_\_\_

<b>Script #</b>	
---------------------	--

<b>Execution Date</b>	<b>Executed By:</b>	<b>Results</b>

### Integration Test Team Detailed Schedule

Application area being tested: \_\_\_\_\_

<b>Phase:</b>	
---------------	--

Step	Time	Cycle/Day	Event	Responsibilit y	Phone	Duration



**IT Test Plan  
Approval Form**

The undersigned certify that they have reviewed the IT Test Plan. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Integrated Testing Team

Date

Andersen Consulting

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

**IT Test Schedule  
Approval Form**

The undersigned certify that they have reviewed the IT Test Schedules. It has been determined that the deliverables meet requirements, meet requirements with concerns, or do not meet requirements. If the deliverables meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

---

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

\_\_\_\_\_

Integrated Testing Team

Date

- Meets Requirements

---

Andersen Consulting

- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

### **IT Technical Environment Checklist Approval Form**

The undersigned certify that they have reviewed the IT Technical Environment Checklist. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

**IT Test Conditions  
Approval Form**

The undersigned certify that they have reviewed the IT Condition Tracking Sheet. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

\_\_\_\_\_  
Integration Testing Team Manager

Date

---

Andersen Consulting

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

**IT Test Cycles  
Approval Form**

The undersigned certify that they have reviewed the IT Cycle Tracking Sheet. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

### **IT Test Scripts Approval Form**

The undersigned certify that they have reviewed the IT Test Scripts. It has been determined that the deliverables meet requirements, meet requirements with concerns, or do not meet requirements. If the deliverables meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns

Andersen Consulting

Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

**IT Entry Criteria  
Approval Form**

The undersigned certify that they have reviewed the IT Entry Criteria Worksheet. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

Andersen Consulting

-----  
-----  
Integration Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- -----  
-----

-----  
-----  
Integration Testing Team Manager

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- -----  
-----

-----  
-----  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

**IT Test Script and Cycle Execution  
Approval Form**

The undersigned certify that they have reviewed the executed IT Test Scripts and Cycles. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

---

-----  
Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

---

-----  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

---

-----  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

**IT Exit Criteria  
Approval Form**

The undersigned certify that they have reviewed the completed IT Exit Criteria Checklist. It has been determined that the results meet requirements, meet requirements with concerns, or do not meet requirements. If the results meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

---

Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

IS Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

\_\_\_\_\_

Integrated Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

**UAT Test Plan  
Approval Form**

The undersigned certify that they have reviewed the UAT Test Plan. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

---

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

---

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
  - Meets Requirements with concerns
- 

Andersen Consulting

Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements  
 Meets Requirements with concerns  
 Does Not Meet Requirements

---

---

---

**UAT Test Schedule  
Approval Form**

The undersigned certify that they have reviewed the UAT Test Schedules. It has been determined that the deliverables meet requirements, meet requirements with concerns, or do not meet requirements. If the deliverables meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements  
 Meets Requirements with concerns  
 Does Not Meet Requirements

---

---

---

\_\_\_\_\_  
Integration Testing Team Manager

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

**UAT Technical Environment Checklist  
Approval Form**

The undersigned certify that they have reviewed the UAT Technical Environment Checklist. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_  
Integration Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 

Andersen Consulting

\_\_\_\_\_  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

### **UAT Test Cycles Approval Form**

The undersigned certify that they have reviewed the UAT Test Cycles. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

-----  
Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

-----  
Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

**UAT Test Scripts  
Approval Form**

The undersigned certify that they have reviewed the UAT Test Scripts. It has been determined that the deliverables meet requirements, meet requirements with concerns, or do not meet requirements. If the deliverables meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

Andersen Consulting

**UAT Entry Criteria  
Approval Form**

The undersigned certify that they have reviewed the UAT Entry Criteria Checklist. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_

Integrated Testing Team

Date

Andersen Consulting

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

### **UAT Test Script and Cycle Execution Approval Form**

The undersigned certify that they have reviewed the execution of the UAT Test Scripts and Cycles. It has been determined that the deliverables meet requirements, meet requirements with concerns, or do not meet requirements. If the deliverables meet requirements with concerns, or do not meet requirements, an explanation must be provided along with necessary documentation.

---

\_\_\_\_\_

Integration Testing Team

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

---

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
-

---

---

\_\_\_\_\_

Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

**UAT Exit Criteria  
Approval Form**

The undersigned certify that they have reviewed the UAT Exit Criteria Checklist. It has been determined that the deliverable meets requirements, meets requirements with concerns, or does not meet requirements. If the deliverable meets requirements with concerns, or does not meet requirements, an explanation must be provided along with necessary documentation.

---

\_\_\_\_\_

Integration Testing Team Manager

Date

- Meets Requirements
  - Meets Requirements with concerns
  - Does Not Meet Requirements
- 
- 
- 
- 

---

\_\_\_\_\_

Functional Executive

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---

\_\_\_\_\_

Integrated Testing Team

Date

- Meets Requirements
- Meets Requirements with concerns
- Does Not Meet Requirements

---

---

---

---