COMMONWEALTH OF PENNSYLVANIA
DEPARTMENT'S OF PUBLIC WELFARE, INSURANCE, AND AGING

INFORMATION TECHNOLOGY STANDARD

Name Of Standard: **Software Quality Assurance Tests**	Number: **STD-EASS009**
Domain: **Application**	Category: **Testing / Software Quality Assurance Tests**
Date Issued: **05/04/2010**	Issued By Direction Of:
Date Revised: **02/25/2014**	Shirley A. Monroe, Dir of Div of Technical Engineering

Abstract:

Application software testing is the process used to identify the correctness, completeness**,** and quality of an IT system. It involves any activity performed to evaluate an attribute or functional capability of a system or component of a system and determine if it meets its expected and required results. Testing application systems is a quality assurance function to facilitate the assessment of end product quality and performance as well as costs to avoidance in capturing software errors and systems anomalies prior to the software being released into a live production environment.

The purpose of this standard is to establish processes used by the Department of Public Welfare (DPW) to define criterion regarding software test methods and types to eliminate or minimize defects impacting the intended use of software application systems delivered to the customers.

The contracted solution providers performing work for DPW are required to establish a comprehensive test plan with well designed execution strategies. The contracted solution providers should identify specific constraints in the test plan document that would limit their ability to thoroughly examine all the functionality and components associated with the application. The contracted solution providers are also required to identify specifically what functionality and/or components that cannot be tested and any areas where test coverage or effectiveness will be diminished due to a particular constraint. The contracted solution providers will coordinate with the

Department to validate the constraint and if required outline mitigation strategies for these specific constraints where feasible.

Adherence to these policies is mandatory. When a DPW and/or contracted solution provider believe there is a need to deviate from these standards/policies, the agency and/or Offeror must first receive written approval to do so from the DPW, Chief Information Officer (CIO), Contract Administrator, or designated BIS Director.

General:

Software testing is a vital but integral component of the systems development life cycle (SDLC). The objectives of software testing are to: 1) Identify and correct as many defects as possible in the development phase; 2) Improve software integrity, reliability, and end product quality; 3) Validate software meets all specified business and technical requirements; 4) Assess operational readiness to support business operations; 5) Evaluate and leverage test results to improve software development and delivery processes and procedures; 6) Lower down stream maintenance costs. For software testing to be effective, it must be carefully planned, designed, executed, and accurately documented. Software testing is a key component of any sound quality assurance and quality control program, software engineering process, and ITIL release, deployment, and continual service improvement processes.

Standard:

Test Methodology

The following subsections describe test methodologies and types of testing to be performed during project SDLC phases.

Testing requires both a standard for correctness and a means of determining correctness. The standard for correctness is typically a requirement. A requirement may include a specific description of functionality, a defined level of service, or a prescribed format. The means of determining correctness is described below.

Elements of Testing

According to accepted best practices, four specific elements of testing are used to determine correctness: inspection, analysis, demonstrations, and test.

- **Inspection**: Inspection is the direct perception of the correctness of the item under test. Examples include viewing application input based entry forms to ensure that the field labels are correct, viewing a printed report to ensure that the format and content are correct or other examples where the immediate perception of the item under test will reveal whether it is correct.
- **Analysis**: Analysis is the assessment, calculation, or breakdown of an item under test to determine that the perceived information is correct. Analysis is the process of taking the immediate data or presentation and either decomposing it or combining it with other information to ensure that the item

under test is correct. Examples of analysis include the manual calculation of a result displayed by the item under test, using the source data used by the test item to ensure that the item under test performs the calculation correctly, the investigation to determine whether a business rule has been applied correctly, or other examples which validate that information provided by or functions performed by the item under test are correctly derived.

- **Demonstration**: Demonstration is the simple execution of a function in the item under test. In demonstration, the item under test initially resides in one state and is triggered to perform some function that either results in an output and/or causes the item under test to move to a new state. Examples of demonstration include accepting a form, generating a report, posting to an account, saving a file, or other actions that are the direct result of triggering an event. Like inspection, the correctness of a demonstration can be immediately perceived; but unlike inspection, an event must occur to cause the item under test to perform its function. This includes Business and Technical walkthroughs by development teams with project stakeholders.

- **Test**: Test is the methodical (and typically documented) combination of a set of inspections, analyses, and demonstrations that provide a well-defined test procedure to ensure the correctness of some functionality in the item under test. Usually, a complex function cannot be assessed for correctness by a single inspection, analysis, or demonstration. Most commonly, a series of steps must be performed that include some ordered set of demonstrations with inspections and analysis to verify functionality.

The elements of testing are applicable to each of the types of tests described below.

Functional Testing

Functional testing ensures that the system or system component correctly performs its intended function. In addition to a description of the function to be performed, functional requirements may include requirements for allowable or unallowable inputs and outputs and specific operations to be performed in satisfying the function.

If functionality will be tested against use cases, the actor, pre-conditions, post-conditions, scenarios, and alternate scenarios should be validated in testing to ensure that the functionality satisfies the use case.

Functional testing should address these factors for each function to be tested. Each functional test should have a unique identifier and each functional test should trace to the function or design element (and its unique identifier) to be tested.

Data Testing

Testing of data includes both the testing of the data contained within the system and the testing of the data used by the system.

Internal data may be in the form of constants, rules, or other data that is either static or seldom changes during the production use of the system

External data includes both data that is accepted by the system and data that is generated by the system, such as user input, communications with other applications, database sources, constructed data, and reports.

Testing of data ensures that the item under test accepts or delivers all and only the data intended for that item, in the form, format, and frequency that is correct for that item.

Each data test should have a unique identifier, and each data test should trace to the function (and its unique identifier) that describes the data to be tested.

User Testing

User testing ensures that the item under test meets usability, accessibility, and user documentation requirements.

User testing for usability and accessibility addresses the required behavior of the user interfaces. These should be documented in a set of specific requirements that ensure the testability of the user interface. The ultimate success of systems that are heavily dependent upon user interfaces often rests upon the clarity of the requirements for the user interface and the success of the system in meeting those user interface requirements.

User interface testing should ensure that the user interface meets the quantified and specific look and feel requirements, including any performance or ease-of-use requirements that are documented in specific, testable terms (e.g., requiring a system to be easy to use is a highly subjective and therefore less testable requirement than requiring a system to provide a display response time of less than or equal to one second.).

Testing of user documentation includes testing of online documentation, help, and other support text to ensure it is correct and adequately supports the needs of the typical users. User documentation testing also includes testing of the system according to any written documentation to ensure that the documentation correctly describes the component or system functionality (e.g., installation instructions for the system should be tested by performing an installation according to the instructions).

Each user test should have a unique identifier, and each user test should trace to the user requirement (and its unique identifier) that describes the user function to be tested.

Non-functional (Systems Requirements) Testing

In the following subsections, describe the non-functional or systems requirements testing for the project.

Non-functional or systems requirements testing validates required qualities or properties of the system that address how well some behavioral or structural aspect of the system should be accomplished. Non-functional tests include: performance testing, quality testing, and interface testing, among others.

Performance Testing

Performance testing ensures that the item under test meets specified requirements for throughput, number of users, response times, maximum workloads, and other performance characteristics of the item.

Performance testing may include load, stress, and availability testing. Load testing determines the ability of the item under test to support the performance requirements while under increased use up to the stated bounds of its capability. Stress testing determines the ability of the item under test to deal with situations when loads exceed the bounds of the stated capability (e.g., whether the item locks up, fails, processes in a degraded fashion, or otherwise changes performance). Availability testing determines whether the item under test is available according to its stated availability requirements (e.g., 24x7 ability to log in to the application).

Load and stress typically have a direct impact on availability and performance testing and typically addresses each individually as well as in combination. For example, a performance test may test an item under load with transient stresses while looking simultaneously at the effect of load and stress on availability.

Each performance test should have a unique identifier and each should trace to the performance requirement (and its unique identifier) that describes the performance requirement to be tested.

Quality Testing

Quality validation includes the following subcategories:

• Correctness – Correctness is the extent to which specifications are satisfied and mission objectives are fulfilled.

• Efficiency – Efficiency is the relationship between the level of performance of the product and the amount of resources used, under stated conditions.

• Flexibility – Flexibility is the effort required to modify operational product.

• Integrity/Security – Integrity/Security is the extent to which access to the system or data by unauthorized personnel can be controlled.

• Interoperability – Interoperability is the effort needed to couple one system with another.

• Maintainability – Maintainability is the effort required to locate and correct an error during operation.

• Portability – Portability is the effort needed to transfer from one hardware or software environment to another.

• Reliability – Reliability is the extent to which the system performs with required precision and robustly responds to reliability challenges (e.g., through fail over or degraded operation).

• Reusability – Reusability is the extent to which the system and associated artifacts can be reused in another application.

• Testability – Testability is the effort needed to test to ensure software performs as intended.

• Usability – Usability is the effort required to learn, operate, prepare input for, and interpret output from the system.

Though many quality requirements for a system may be general to the system, some may be specific to particular function(s) or component(s) of the system. Any associated testing for these quality requirements must address the scope and limitations of the particular requirement. For example, the entire system may have a general security requirement that prohibits unauthorized users from accessing the system. Conversely, a portability requirement may be applicable only to a specific user interface of a system.

Specific quality requirements may also be mutually limiting or may compete with performance or functional requirements. For example, maintainability and security may conflict at times, since making a system more secure may also decrease its maintainability. Similarly, portability is often at odds with systems that require extremely high performance, since tuning a system for performance may make it less portable. Unless care is taken in test planning and design, these issues tend toward less objective criteria for testing quality requirements.

For each quality requirement that is satisfied through testing (as opposed to other forms of validation), the test should have a unique identifier, and each should trace to the quality requirement (and its unique identifier) to be tested.

Interface Testing

Interface testing determines the correctness of the defined interfaces for the system. These interfaces may be internal to the system or with interfaces with other systems. The interfaces may be between software and hardware components or software and software components. The interface typically has a specific format, message set, and protocol. Testing the interface according to the requirement will necessitate testing the interface features and any performance and error detection/recovery mechanisms for the interface.

Each interface test should have a unique identifier, and each should trace to the interface definition and requirement and its unique identifier.

Security Testing – Software Vulnerability Testing

Vulnerability testing is a method of evaluating the security of a computer system or network by simulating an attack from a malicious source. The process involves an active analysis of the system for any potential vulnerabilities that could result from poor or improper system configuration, both known and unknown hardware or software flaws, and operational weaknesses in process or technical countermeasures. This analysis is carried out from the position of a potential attacker and can involve active exploitation of security vulnerabilities. Any security issues that are found will be presented to the system owner, together with an assessment of their impact, and often with a proposal for mitigation or a technical solution. The intent of a penetration test is to determine the feasibility of an attack and the amount of business impact of a successful exploit, if discovered. It is a component of a full security audit.

Penetration tests can be conducted in several ways. The most common difference is the amount of knowledge of the implementation details of the system being tested that are available to the testers. Black box testing assumes no prior knowledge of the infrastructure to be tested. The testers must first determine the location and extent of the systems before commencing their analysis. At the other end of the spectrum, white box testing provides the testers with complete knowledge of the infrastructure to be tested, often including network diagrams, source code, and IP addressing information. There are also several variations in between, often known as grey box tests. Penetration tests can also be described as "full disclosure" (white box), "partial disclosure" (grey box), or "blind" (black box) tests based on the amount of information provided to the testing party.

White Box Testing

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test case designers shall generate cases that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once

.

Branch Testing

Using the program flow graph for each function, we will be able to determine all of the branches that will need to tested and will be used to develop the corresponding test cases.

Black Box Testing

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We have decided to perform Equivalence Partitioning and Boundary Value Analysis testing on the application.

Equivalence Partitioning

In considering the inputs for equivalence testing, the following types could be used:

- Legal input values – Test values within boundaries of the specification equivalence classes. This shall be input data the program expects and is programmed to transform into usable values.
- Illegal input values – Test equivalence classes outside the boundaries of the specification. This shall be input data the software application system may be presented, but that will not produce any meaningful output.

The equivalence partitioning technique is a test case selection technique in which the test designer examines the input space defined for the unit under test and seeks to find sets of input that are, or should be, processed identically. The following table represents an example of equivalence classes, both valid and invalid.

Input/Output Event	Valid Equivalence Classes	Invalid Equivalence Classes
Input maximum number of allowed values	25 values	> 25 values
Input integers	Integers between –999 and 999	Integers > 999 Integers < -999 Non-integers (characters) Non-integers (decimal values)
Load external file	Comma delimited file with only one value per line File exists	No commas Multiple entries per line No file content File does not exist
Store external file	File exists	File does not exist

Function Validation Testing

Functional testing Integration will be tested based on the requirements. The behaviors of each function, as well as their respective algorithms, are contained in the Software Application System Specifications.

Function	Expected Behavior
Load	see Software Program Specification
Store	see Software Program Specification
Insert	see Software Program Specification
Delete	see Software Program Specification
Search	see Software Program Specification
Clear	see Software Program Specification
List in Ascending Order	see Software Program Specification
List in Descending Order	see Software Program Specification

Performance Testing

This test will be conducted to evaluate the fulfillment of a system with specified performance requirements. It will be done using black-box testing method. And this will be performed by:

- Storing the maximum data in the file and trying to insert, and observe how the application will perform when it is out of boundary.
- Deleting data and check if it follows the right sorting algorithm to sort the resulting data or output.
- Trying to store new data and check if it over writes the existing once.
- Trying to load the data while they are already loaded

Other Testing

Other testing includes testing of features or requirements that are deemed to require testing that are not covered in the types of tests described above, ie: Federal and State requirements, ADA compliance. All other tests should have a unique identifier and each test should trace to the definition, requirement, and its unique identifier.

Phases of Testing

Provide an overview of the test effort for the project by completing the Test Phase Chart provided in the template. Identify the planned test phases and provide the general objective, focus, test types, staffing, environment, entry criteria, suspension criteria, and exit criteria for each phase.

Test Phase	Unit	Integration	System	User Acceptance	Production
Objective	To test units of code that are considered complete	To ensure that aggregates of units perform accurately together	To ensure that the system performs according to documented requirements and the customer's expectations	To ensure that the completed system performs according to documented requirements and the customer's expectations	
Focus	Correctness of specific functionality of a unit and its input, outputs, and primary and fault handling	Correctness of the aggregate with regard to its associated requirements	Correctness of the system and that it conforms to stated requirements	Correctness of all functionality of the system	
Test Types/ Subtypes	Functional (low-level),	Functional, Data,	Functional, Data,	Functional, Data,	…

	Data, Performance, Integrity/Security, Interface	Performance, Reliability, Integrity/Security, Interface, Usability	Performance, Reliability, Integrity/Security, Interface, Usability	Performance, Reliability, Integrity/Security, Interface, Usability	
Staffing	Development Team (author of a unit should not be the tester)	Test Team, independent of the development team (member of the integration test team should not perform revisions to code he is testing)	Test Team, independent of the development team (member of the system test team should not perform revisions to code he is testing)	Customers or end users, can be lead by testers independent of the development team (member of the acceptance test team should not perform revisions to code he is testing)	…
Environment	Development	Should be performed in an environment segregated from development and controlled by a group independent of the development team	Should be performed in an environment segregated from development and controlled by a group independent of the development team	• Should be performed in an environment segregated from development and controlled by a group independent of the development team • Should be identical to the product-ion environment or as close as possible	...
Test Phase	**Unit**	**Integration**	**System**	**User Acceptance**	**Produ ction**

Entry Criteria	• Code complete • Software Test Plan approved • Unit test procedure and scenarios approved • Unit test data approved	• Code complete • Software Test Plan approved • Unit test successful • Integration test procedure and scenarios approved • Integration test data approved • Integration test exit criteria, including allowable errors and functional discrepancies, specified	• Code and documentation baselined • Software Test Plan approved • Integration test successful • System test procedure and scenarios approved • System test data approved • System test exit criteria, including allowable errors and functional discrepancies, specified • Unit and Integration test reports complete • Requirements Traceability Matrix complete • User documentation complete • Test Readiness Review minutes complete	• Code and documentation baselined • Software Test Plan approved • Integration test successful • Acceptance test procedure and scenarios approved • Acceptance test data approved • Acceptance test exit criteria, including allowable errors and functional discrepancies, specified • Unit, Integration, and System test reports complete	…
Test Phase	**Unit**	**Integration**	**System**	**User Acceptance**	**Production**

Suspension Criteria	• Environment Issues • Design rework • Development rework • Requirements changes	• Unit development not complete Unit test not complete, unsuccessful, or inadequate • Environment issues • Design rework • Development rework • Requirement changes • Defects encountered > X • Fault with major feature that prevents significant functionality from being tested	• Environment issues • Design rework • Development rework • Requirement changes • Defects encountered > X • Installation problems • Fault with major feature that prevents significant functionality from being tested	Environment issues • Design rework • Development rework • Requirement changes • Defects encountered > X • Installation problems • Fault with major feature that prevents significant functionality from being tested	…
Exit Criteria	• Test logs approved • Unit test scenarios passed • Unit is baselined	• Test logs approved Allowable errors and functional discrepancies • Allowable errors and functional discrepancies do not exceed thresholds	• Test logs approved Allowable errors and functional discrepancies do not exceed thresholds • System is baselined	• Test logs approved • Stakeholder acceptance	…

Testing Identified by Specific Project: Project Management Approach

The following subsections describe the types of testing which correlate to the various types of requirements for the system or system component. The specific

testing approaches will be defined per individual project during the Project Management Planning Phase and implemented in the Execution and Monitoring and Controlling Project Management Phases.

Test Schedule

Provides a reference to the location of the project test schedule information or specify the project test schedule information that establishes the sequence and coordination for the project's test activities. State the sequence and dependencies among all test activities and the relationship of key test activities to project milestones or events. Cover the duration of the Software Test Plan and include all major milestones of the project related to test activities. Include resources, prerequisites, and start/completion dates for each activity, deliverable, and milestone.

Test Monitoring

Describes monitoring activities and milestones that will be used to evaluate actual progress to plan. Execution of the Software Test Plan must be monitored to recognize deviations from plan. Testing progress and success depends on realistic, detailed Software Test Planning and frequent, interim milestones at which actual progress can be compared with the plan to identify deviations.

Commonly monitored test metrics include percent of system tested, test cases executed/passed/failed/not executed, number and severity of problems reported, number of problems closed/resolved/retested, and cost of test effort (estimated versus actual).

Examination and evaluation of these metrics should include trend analysis. Trend analysis of metrics provides the ability to make comparisons between the project's metrics from other test phases or cycles and between metrics from other projects. Trend analysis can provide substantial information about the quality of the product or product segment.

Test Reporting

Describe reports that will be used during testing. Each test phase should have one or more reports to document test execution and results. The reports will:

- Identify the items tested
- Summarize the evaluation of the planned test items (expected versus actual, including the impact of variances)

- Indicate the version/revision level of the software tested
- Indicate the environment in which the testing took place
- Contain references to the Software Test Plan, test scenario, test procedure, test log, and problem reports, if they exist
- Specify metrics that were monitored during the testing effort, including any trend analysis compiled
- Contain a comprehensive test evaluation summary, including conclusions regarding the quality and stability of the product.

Examples of Deliverables

- Program function specifications
- Program source code
- Test plan document - this document should address testing objectives, criteria, standards, schedule and assignments, and testing tools.
    - Unit Testing Plan
    - Integration Plan
    - System Testing Plan
- Test Design Document
    - Unit white-box test design – covers white testing criteria, methods and test cases
    - Unit black-box test design – covers black-box testing criteria, methods and test cases
    - System test design – covers system test criteria, methods, and test cases, scripts.
- Test report document
    - Unit white-box test report – covers unit white box test results, problems, summary and analysis
    - Unit black-box test report – covers unit black box test results, problems, summary and analysis
    - System Test report – covers system test results, problems, summary and analysis

Unit Testing Phase

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. Unit testing tests a single module or a related group of modules with the intention of finding errors. The unit test cases shall be designed to test the validity of the programs correctness.

- Test negative exceptions as well as positive conditions.

- Test for boundary conditions, i.e., high, low, range values or alpha in numeric fields.
- Test all possible paths of flow, including unchanged code.
- Verify screen navigation.
- Check transaction security is at appropriate levels.
- Check screen or report layouts are to specification.
- Check response time is at an acceptable level, considering run is in the test environment. Check new ECL or ECL changes.
- Verify that the program reaches successful completion.
- Verify program documentation has been modified or added.
- If applicable test internal controls.

Use test scripts or a plan which lists conditions and expected results for individual modules.
Ideally, use test case designs which maximize the number of errors by a finite number of test cases. This is not required to be formal documentation.
Errors which are detected may be formally or informally documented by the programmer for correction and retesting.

Module Testing Phase

Testing two or more modules for functionality and recording results. Example: There are two primary modules that will need to be integrated: the Application Layer front end and the Application Layer back end. The two components, once integrated, will form the complete module or Application System test.

Integration Test Phase

In this phase of software testing individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

The primary purpose of integration testing is to prove that representative data will flow all the way through the system, producing valid output and will handle invalid data conditions. Integration testing also verifies that the data flow between this system and other interfacing systems is correct.

- Ensure that modules have been unit tested.
- Develop a formal plan of how each set of programs/modules will be integrated.
- Define a time line when each set of programs will be tested.
- Create a test script to list areas to be tested at high level functions, i.e., 'order entry', 'update A/R'.
- Create a sequence of specific tests including expected results within each high-level grouping.

- An integration test log of errors should be maintained by the tester for releases, large and medium development projects.
- Production and test data should be used. If production data is not available, the test data should be representative of the production environment.
- A complete review of test results by the programming staff is extremely important at this phase; users may be involved in test data entry and review of results.
- Timing runs should be made using input volumes similar to or greater than those anticipated in actual operations to ensure peak loads can be handled.
- Retest job-to-job controls.
- Verify storage allocations are adequate for expected growth.
- Performance tests should be run to be sure run times/response times are acceptable.
- Program changes should be reflected in the program, user, and system documentation.
- The documentation should be tested against what is actually delivered to verify that it is adequate to enable users and computer operations to run the system.

System Acceptance Test Phase

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system. System testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. System test focuses on functions and performance, reliability, instillation, behavior during special conditions, and stress testing.

User Acceptance Test Phase

The primary purpose of user acceptance testing is to determine if the system performs to business specification. Development of the acceptance criteria and test plan and the actual performance of user acceptance testing is the responsibility of the user.

The test plan includes tests of all system functions. Each system function should include specific test cases with expected input data and test results. The user project manager/coordinator maintains the log to keep track of test cases, error resolution and retesting. The format for documenting test cases and the error log is up to the user project manager. Samples from other systems user testing may prove helpful.

In some instances, such as production problem resolution or releases, user testing may have been performed during integration testing. All user acceptance testing is done in a test environment. If there are several libraries in test, user testing should be reserved for one of the testing environments.

- Verify valid data flows through the system correctly and invalid data is handled appropriately.
- Check all edits, validation rules, default values, error messages and warning messages.
- Attest system security is at appropriate levels, e.g., restricted add/update authority for tables/databases.
- Validate system controls such as error file handling or duplicate entry processing.
- Make sure response times for processing are acceptable.

Ensure disaster recovery procedures are put in place.
If applicable to the system, it is also suggested that you:

- Verify screen navigation is correct and easy to follow.
- Check screen and/or report layouts to assure all required data exists and is mapped correctly.
- Assure system can handle high volumes.
- Where add/updates are performed, assure correct values are placed on the databases/tables.
- Verify data transmission to and from system are successful.
- SQA verifies that all open product defects, regardless of fixed defects, documented, deferred, or otherwise addressed.
- SQA verifies that regression testing on all product defects and the entire product has been completed.
- SQA verifies that all defects “For Verify” have been regressed.

The software is frozen when the product passes its final milestone. If any code changes are made after the final milestone, the features fixed must be re-tested. SQA and Development Team closely monitor fixes that go into the final build to minimize risk. After the final milestone criteria have been met, the product enters the Test for Production phase.

- Business stakeholder acceptance signoff approval for production implementation.

Test for Production Phase

The test-for-production (TFP) environment must mirror the production environment except during the brief period when you are testing application migrations.

The TFP environment provides the platform for testing the effect and impact of an imminent change to production environment. By performing these changes on the TFP environment the risk of performing the changes directly to the production environment is reduced.

Production Implementation Phase

Production implementation follows the Project Development Team Implementation Plan. Project stakeholders are informed of the pending implementation. Stakeholder teams are prepared for production implementation, roles and responsibilities. Implementation is conducted. Stakeholder Go-NoGo decision point is conducted. Implementation proceeds with post implementation tasks, monitoring and controlling or the fall back plan is implemented.

Exemptions from this Standard:

There will be no exemptions to this standard.

Refresh Schedule:

All standards and referenced documentation identified in this standard will be subject to review and possible revision annually or upon request by the DPW Information Technology Standards Team.

References:

This document contains best practices, suggestions, and ideas that were gathered from experience or study of the following sources.

- IEEE Standard for Software Test Documentation (ANSI/IEEE Standard 829-1983)

- Defense Finance and Accounting Service – Software Test Plan
- **www. CarnegieQuality.com**
- **http://www2.dir.state.tx.us/management/projectdelivery/projectframework/Pages/Framework.aspx**

Standard Revision Log:

Change Date	Version	Change Description	Author and Organization
04/29/2010	1.0	Initial creation	Thomas King
12/28/2010	1.1	Reviewed Content – No Changes	Thomas King
02/25/2014	1.2	Reviewed Content – Changed Director name to Kevin Gray, content is current	Michael Light