



SYSTEM TESTING PROCESS

VERSION 1.0

AUGUST 10, 2007

Information and Technology Management Branch

❖ IM / IT Standards & Guidelines ❖

| | | |
|-----------|--|----------|
| 1. | INTRODUCTION | 3 |
| 2. | DOCUMENT PURPOSE | 3 |
| 3. | SYSTEM TESTING PROCESS | 3 |
| | 3.1 SUBSYSTEMS | 3 |
| | 3.2 SYSTEM TESTING APPROACHES | 4 |
| | 3.2.1 <i>Top-down approach</i> | 4 |
| | 3.2.2 <i>Bottom-up approach</i> | 4 |
| | 3.2.3 <i>Hybrid approach</i> | 4 |
| | 3.3 SYSTEM TESTING STANDARDS AND GUIDELINES | 5 |
| | 3.4 SYSTEM TESTING ROLES AND RESPONSIBILITIES | 6 |
| | 3.5 SYSTEM TESTING CHECKPOINTS IN THE MINISTRY ADE-SDLC..... | 6 |
| | 3.5.1 <i>New Application development</i> | 6 |
| | 3.5.2 <i>Maintenance of existing applications</i> | 7 |
| | 3.5.3 <i>Infrastructure Migration Projects</i> | 7 |
| | 3.5.4 <i>Emergency Fix</i> | 7 |
| | 3.6 SYSTEM TESTING TOOLING OPTIONS | 7 |
| | 3.6.1 <i>Open Source Tools</i> | 7 |
| | 3.6.2 <i>Licensed Tools</i> | 7 |

1. Introduction

System testing (also known as *Integration testing*) is a logical extension of unit testing. Two or more units that have already been tested are combined into a component (or a subsystem) and the interface between the units is tested. The goal of System testing is to ensure that all interacting units and subsystems in a system are interfacing correctly with one another to produce the desired results. System tests will also ensure that the introduction (or removal) of one or more units or subsystems into the system does not have an adverse affect on existing functionality of the system as a whole. A system test covers the testing of interface points between units and subsystems. System testing is performed once unit testing has been completed for all units contained in the subsystems being tested. System testing identifies problems that occur when units are combined. Since units were individually tested prior to combining (integrating) them, any errors discovered when combining units are likely related to the interface between units.

2. Document Purpose

The purpose of this document is to define and describe a process for System (Integration) Testing of Ministry's applications. This document also describes System testing roles and responsibilities, check points etc. The main intended audience for this document is the developers who need to know about System testing requirements of Ministry Applications. Also, Ministry staff who are involved in infrastructure upgrades such as Middle Tier upgrades would find this document useful.

3. System Testing Process

The Ministry ADE process recommends the following standards on System Testing process.

3.1 Subsystems

Strategy for integration of units into a subsystem and subsystems into a system is one of the most significant aspects of application development. There are established strategies for integration available such as "*all at once*", *top-down*, *bottom-up*, *critical pieces first*, *integration by functionality* etc. To be most effective, the system (integration) testing technique should fit well with the overall integration strategy. No single integration strategy would fit all the applications and projects. Ministry ADE recommends that project teams carefully evaluate the approaches and select suitable integration approach based on the project needs.

3.2 System Testing approaches

There are three common approaches available for system testing which are described below. In all these approaches, the concept of *Test Driver* and *Stub* is heavily used. A *test driver* is software (imitation of a higher-level unit) which executes software in order to test it, providing a framework for setting input parameters, executing the unit, and reading the output parameters. A *stub* is an imitation of a lower-level unit, used in place of the real unit to facilitate testing before the unit itself is ready.

3.2.1 Top-down approach

The top-down approach to integration testing requires the highest-level modules be tested and integrated first, with all called units replaced by *stubs*. Testing continues by replacing the stubs with the actual called units when ready, with lower level units being stubbed. This process is repeated until the lowest level units have been tested and integrated. In top down testing, individual units are tested by using them from the units which call them, but in isolation from the units called. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management since called units are to be replaced by stubs and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality. However, a top-down approach to the integration of units, where the units have already been tested in isolation, could be viable.

3.2.2 Bottom-up approach

The bottom-up approach requires the lowest-level units be tested and integrated first. The lowest level units are tested first, then used to facilitate the testing of higher level units. Other units are then tested, using previously tested called units. The process is repeated until the unit at the top of the hierarchy has been tested. These lowest-level units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for test drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.

3.2.3 Hybrid approach

This approach (also called umbrella approach) is a combination of best feature of top-down and bottom-up approaches. It requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The

weaknesses of this approach are that it can be less systematic than the other two approaches, leading to the need for more regression testing.

No single approach is likely to suite all the applications and projects. The Ministry ADE recommends that the project teams evaluate the pros and cons of each approach discussed above and adopt appropriate approach that suits the specific project.

3.3 System Testing Standards and Guidelines

Ministry ADE process recommends the following guidelines for System Testing of Ministry Applications :

- System Testing must be always done in Ministry TEST environment and in no other internal or external environments.
- Always create a System Test plan (a set of test cases arranged in the sequence of chronological execution) for the subsystems and systems before starting the actual testing.
- Ideally the System Test Plan should be created before starting the integration of the unit(s) into subsystems and subsystems into systems.
- Always create a System Test data before starting the actual testing.
- A test case must exist for testing every interface between the units and subsystems.
- To minimize the number of test cases, combine test cases into one if they test the same interface point.
- Test cases should cover the functional, input, output, and function interaction between the units.
- Use already existing test cases wherever possible.
- Conduct the system tests as specified in the test cases.
- Re-run the necessary tests after every change to the unit(s), integrated subsystems and the system as a whole.
- If any of the actual results do not agree with the expected results, the developer fixes the code and re-runs the test.
- System Test Plan will need to be updated if it is determined that the Test Plan is not correct or no longer up-to-date.
- Ensure the Unit(s) pass the unit test cases before they are integrated into subsystems/system and before they are moved from DEV to TEST environment.
- Where possible, create scripts to automate the execution of the test case.
- Divide the interface points into logical groupings, with each group corresponding to at least one test case.
- Arrange test cases in the order that minimizes the effort required for test setup and that keeps related functions together.
- Design test data that will ensure that all conditions and qualities of data edits are covered.

- Use live or representative data as much as possible in order to provide realistic test scenarios.

Ministry ADE process requires that System testing is done on all the subsystems and the whole system (application) before the request for promotion of the whole application to UAT is considered. To this effect, the development teams are required to submit the following to the Ministry configuration management team or Middle Tier team along with the request for promotion :

- **System Test results (or log files) showing that the system test was successful**
- **System test plan (optional, but desirable)**
- **System Test Data (optional, but desirable)**

Ministry ADE process recommends preserving and storing of all System Test cases for future re-running purposes.

3.4 System Testing Roles and Responsibilities

The Ministry ADE process recommends that the whole process of System Testing (which could ideally include System Test Plan creation, System Test cases design, System Test execution, System Test results review and action, promoting the whole system from DEV to TEST etc) is the responsibility of the Development Teams.

The Ministry ADE process recommends that pre-established System Test cases are preserved and saved in a readable format and at an accessible location, so that the test cases could be re-used for doing system re-testing in future during application changes, infrastructure migrations etc without having to re-design the test cases.

3.5 System Testing Checkpoints in the Ministry ADE-SDLC

The Ministry ADE process recommends the System testing be performed for all Ministry applications (regardless of category, roadmap and technology) at the following checkpoints in the Application's life cycle :

3.5.1 New Application development

All new applications must ensure that full System testing of the application is done as per the System Testing standards defined by the Ministry ADE process. System test results and log files (if any) must be submitted along with the request for promotion of the application from TEST to UAT environment. Ministry staff at their discretion may do a random auditing of the system test results and log files and/or randomly re-run the system test cases (if available) prior to promoting the application to TEST environment.

3.5.2 Maintenance of existing applications

During the implementation of post-production changes to the Applications, all the affected subsystems and also the whole system must be subjected to re-testing in the TEST environment with the previously established System test cases before promoting the system from DEV to TEST environment. Modification to the System test cases, if required, may be done before using them.

3.5.3 Infrastructure Migration Projects

System Testing is mandatory for infrastructure upgrade projects such as 10g migration project. All applications impacted by the migration project must be subjected to full System testing in the TEST environment with system test cases that were previously established.

3.5.4 Emergency Fix

System Testing is mandatory during Emergency Fix. Emergency fixes must not be applied directly to the Production environment. They must be applied first in the Emergency Fix environment (EFX). System testing of impacted subsystems and the full system testing must be conducted in the EFX environment with the previously established system test cases before promotion of the system from FIX to Production. Ministry ADE recommends that applying the fix and system re-testing should be done as a post-mortem exercise progressively in all other environments : DEV → TEST → UAT → PRD.

3.6 System Testing Tooling Options

3.6.1 Open Source Tools

For System Testing, no reliable full-fledged tools could be found on the internet. There are however a number of open source tools available for Graphical User Interface (GUI) testing. The developers and testers may use any good open source system testing tool that they know or come across to perform the System Testing.

3.6.2 Licensed Tools

Ministry IMG is in the process of acquiring licensed tools for automating the “end-to-end” regression testing process as a whole. Most end-to-end testing tools are believed to provide good System testing capabilities. Upon procurement and installation and availability of the end-to-end testing tools, the developers and testers will be notified of the same.

