



**UNIT TESTING PROCESS  
VERSION 1.0**

**JULY 19, 2007**

---

**Information and Technology Management Branch**

**❖ IM / IT Standards & Guidelines ❖**

<b>1.</b>	<b>INTRODUCTION</b> .....	<b>3</b>
<b>2.</b>	<b>DOCUMENT PURPOSE</b> .....	<b>3</b>
<b>3.</b>	<b>UNIT TESTING PROCESS</b> .....	<b>3</b>
	3.1 UNITS.....	4
	3.2 UNIT TESTING APPROACHES.....	4
	3.2.1 <i>Structural Testing</i> .....	5
	3.2.2 <i>Unit-level Functional Testing</i> .....	5
	3.2.3 <i>Heuristic/Intuitive Testing</i> .....	6
	3.3 UNIT TESTING STANDARDS AND GUIDELINES .....	6
	3.4 UNIT TESTING ROLES AND RESPONSIBILITIES .....	7
	3.5 UNIT TESTING CHECKPOINTS IN THE MINISTRY ADE-SDLC.....	8
	3.5.1 <i>New Application development</i> .....	8
	3.5.2 <i>Maintenance of existing applications</i> .....	8
	3.5.3 <i>Infrastructure Migration Projects</i> .....	8
	3.5.4 <i>Emergency Fix</i> .....	8
	3.6 UNIT TESTING TOOLING OPTIONS .....	9
	3.6.1 <i>Open Source Tools</i> .....	9
	3.6.2 <i>Licensed Tools</i> .....	9

## 1. Introduction

**Units** are the smallest building blocks of software. **Unit testing** is the process of validating such small building blocks of a complex system much before testing of the subsystems or system as a whole. Unit testing focuses on the testing of the units of the software to ensure that those units are functioning the way they are supposed to do. In recent years, unit testing has become a much more structured process due to the availability of unit testing tools such as JUnit for Java, NUnit for .NET etc.

Some of the major benefits of a formalized Unit Testing process are that it :

- enables to test parts of a project without waiting for the other parts to be available.
- enables achieving parallelism in testing by being able to test and fix problems simultaneously by many developers.
- **enables detecting and removal of defects at a much less cost compared to other later stages of testing in the System Development Life Cycle (SDLC).**
- enables to take advantage of a number of formal testing techniques available for unit testing.
- **simplifies debugging by limiting debugging to a small unit in which to search for bugs.**
- enables testing of internal conditions (such as exception conditions) that are not easily reached by external inputs in the large system as a whole.
- enables achieving of a high level of structural coverage of the code.

## 2. Document Purpose

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

## 3. Unit Testing Process

By definition, a testable "**Unit**" is an entirely independent piece of code that can be tested autonomously, separate from any other code modules. The definition of a "unit" depends on the programming environment being used. For example, a testable unit in Java/J2EE environment would typically be a "java class".

The following sections describe the Ministry ADE recommendations on Unit Testing process.

### **3.1 Units**

The Ministry ADE process has identified some major components as **testable Units** of applications residing in various tiers of the infrastructure. This list is not a complete or comprehensive list and it is provided just as a guideline. Ministry ADE process recommends that all “testable units” should undergo unit testing before embarking on the System Testing of the application.

#### **Web Tier**

- Web Pages and their resources (urls etc.)
- Web Tier code such as PhP, Perl scripts etc.
- XML files
- XMI files

#### **Middle Tier**

- Java code such as Java Classes, Servlets etc.
- Forms Services (Form) component
- Reports Services (Report) component
- Web Service
- WSDL component
- Any other SOA component ??
- XML files
- XMI files

#### **Data Tier**

- Pl/Sql packages
- DDL Scripts
- XML files
- XMI files
- Oracle Discoverer components such as EUL, Discoverer Workbooks etc.

#### **Legacy**

Rdb components

Power House code components

FileMaker Pro components

### **3.2 Unit Testing approaches**

The Ministry ADE process recommends that the Unit testing process ideally should be a combination of three types of testing : “*Structural testing*”, “*Functional Testing*”, and “*Heuristic / intuitive testing*” approaches. The three approaches are described below.

### 3.2.1 Structural Testing

*Structural testing (also called White box Testing or Internal Testing)* is based on the structure of the code. The tester constructs a *test suite* (a collection of inputs and corresponding expected outputs) which demonstrates that all branches of the program's choice and loop constructs such as *if*, *while*, *switch* etc are executed during the testing. The test suite is said to *cover* the statements and branches of the program.

Structural test aims to make sure that all parts of the code have been executed during the testing process. The *test suite* must include enough input data sets to make sure that

- all methods have been called,
- both the “true” and “false” branches have been executed in “if” statements,
- every loop has been executed zero, one, and more times,
- all branches of every “switch” statement have been executed, and so on.

For every input data set, the expected output must be specified. Then the program is run with all the input data sets, and the actual outputs are compared to the expected outputs.

Structural test cannot demonstrate that the program works in all cases, but it is generally an efficient, effective and systematic way to discover errors in the program. In particular structural testing is a good way to find errors in programs with complex logic, and to find variables which are initialized with the wrong values.

Some of the *coverages* or metrics of the Structural testing are :

*Statement/Code Coverage*: Identifies test cases such that every line of code is executed in one test case or other.

*Branch Coverage*: Identifies test cases such that every branch of code is executed in one test case or other. 100% Branch Coverage automatically assures 100% Statement Coverage.

*Condition Coverage*: Identifies test cases such that condition in each predicate expression is evaluated in all possible ways.

*Condition-Decision Coverage*: Identify test cases such that each Boolean operand can independently affect the outcome of a decision.

<b>Ministry ADE process recommends that all major coverages described above should be used in unit testing.</b>
---

### 3.2.2 Unit-level Functional Testing

Unit-level *Functional testing (also called Black box Testing or External Testing)* focuses on the problem that the program is supposed to solve. Unit-level functional testing involves verifying that each unit is implemented according to specification. Thus the tester must have a fairly precise idea of the problem that the unit must solve.

The tester constructs a test data set (inputs and corresponding expected outputs) which includes ‘typical’ as well as ‘extreme’ input data. In particular, one must include inputs that are described as exceptional or erroneous in the problem description.

**Structural and functional test are complementary to each other. Ministry ADE process recommends that both Structural and unit-level functional testing should be performed for individual units before embarking upon the testing of the system as a whole.**

### 3.2.3 Heuristic/Intuitive Testing

In the *Heuristic testing (also known as Intuitive Testing) approach*, the tester separately reviews a program, categorizing and justifying problems based on a short set of heuristics. During *heuristic testing* the tester goes through the program numerous times carrying out a variety of tasks and inspecting how the program scores against a list of identified heuristics.

**Ministry ADE process recommends that just performing Heuristic/Intuitive Testing alone is not enough for Unit Testing. Heuristic testing should be performed in addition to and in conjunction with Structural and unit-level Functional testing.**

## 3.3 Unit Testing Standards and Guidelines

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

- Unit Testing must be always done in Ministry DEV environment. Ministry TEST environment may also be used for unit testing provided the unit(s) has been tested properly in the DEV environment before moving it to TEST environment. The TEST environment must not be used as a substitute/alternative for DEV environment.
- Always create a Unit Test plan (a set of test cases arranged in the sequence of chronological execution) for the units before starting the actual testing.
- Ideally the Unit Test Plan should be created before starting the programming of the unit(s).
- Always create a Unit Test data for the units before starting the actual testing.
- A test case must exist for every branch in the code.
- To minimize the number of test cases, combine test cases into one if they test the same feature.
- Unit testing is to be performed on the units in isolation and not in integration with other units.
- For unit testing of higher-level units develop and use *test drivers*, if the actual higher-level unit itself is not yet ready.

- For unit testing of lower-level units develop and use *test stubs*, if the actual lower-level unit itself is not yet ready.
- Test cases should cover the functional, input, output, and function interaction of the unit.
- Use already existing pre-established test cases wherever possible.
- Conduct the unit tests as specified in the test cases.
- Re-run the necessary tests after every change to the unit.
- If any of the actual results do not agree with the expected results, the developer fixes the code and re-runs the test.
- Unit Test Plan will need to be updated if it is determined that the Unit Test Plan is not correct or no longer up-to-date.
- Ensure the Unit(s) pass the unit test cases before they are moved from DEV to TEST environment and from TEST to UAT environment.
- Where possible, create scripts to automate the execution of the test case.
- Use live or representative data as much as possible in order to provide realistic test scenarios.

**Ministry ADE process requires that Unit testing is done on all the units of the 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 along with the request for promotion :**

- **Unit Test results (or log files) showing that the units have passed the unit tests**
- **Unit test plan (optional, but desirable)**
- **Unit Test Data (optional, but desirable)**

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

**Ministry ADE process also strongly recommends that good Unit testing be performed on all testable units before embarking on the system testing of the whole system.**

### ***3.4 Unit Testing Roles and Responsibilities***

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

The Ministry ADE process recommends that pre-established Unit 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 unit re-testing in future during application changes, infrastructure migrations etc without having to re-design unit test cases.

### **3.5 Unit Testing Checkpoints in the Ministry ADE-SDLC**

The Ministry ADE process recommends the unit 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 unit testing of all the units of the application is done as per the Unit Testing standards defined by the Ministry ADE process. Unit 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 unit test results and log files prior to promoting the application to UAT environment.

#### **3.5.2 Maintenance of existing applications**

During the implementation of post-production changes to the Applications, all the affected units must be subjected to re-testing in the DEV / TEST environments with the previously established unit test cases before promoting them to UAT environment.

#### **3.5.3 Infrastructure Migration Projects**

Unit Testing is an optional (but desirable) task during infrastructure migration projects such as 10g upgrade project. Unit testing should be done for the affected units during the migration projects. After applying fix(s) to the affected units, the affected units should be re-tested with the previously established unit test cases (if available) in the DEV / TEST environments.

#### **3.5.4 Emergency Fix**

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

## **3.6 Unit Testing Tooling Options**

### **3.6.1 Open Source Tools**

The tooling options for Unit Testing would depend upon the various roadmaps/technologies that the Ministry ADE process supports. No single tool is likely to provide Unit Testing solutions for all the roadmaps//technologies and all types of Units defined in section 4 of this document. The following URL on the internet provides a list of popular **open-source** Unit Testing tools for different types of units.

<http://www.testingfaqs.org/t-unit.html>

Developers and Testers may download and use these open-source tools for unit testing of Ministry applications subject to compliance with copyright, licensing and usage policies as outlined in the tool's web sites.

IMG believes that there are tools (such as Link Validator, WebTrends, Dreamweaver) already in place within the Ministry for testing of Web tier units. These tools are used mainly by Webmasters. These tools will continue to be used.

### **3.6.2 Licensed Tools**

Ministry IMG are in the process of acquiring licensed tools for automating the “end-to-end” regression testing process as a whole. If the tool that will be procured happens to support the Unit Testing process, it will be notified to all development teams and ITMB so as to enable them explore the option of using the licensed tool rather than the open-source tools for unit testing.