4/6/2013

# Supermarket Automation Software

Test Suite Design document

Siddharth Rakesh 11CS30036

Sachin Kumar 11CS30043

# Table of Contents

# Introduction

This document is a high-level overview defining testing strategy for the Supermarket Automation software. Its objective is to communicate project-wide quality standards and procedures. It portrays a snapshot of the project as of the end of the planning phase. This document will address the different standards that will apply to the unit, integration and system testing of the specified application. Testing criteria under the white box, black box, and system-testing paradigm will be utilized. This paradigm will include, but is not limited to, the testing criteria, methods, and test cases of the overall design. Throughout the testing process the test documentation specifications described in the IEEE Standard 829-1983 for Software Test Documentation will be applied.

# Test Objective

The objective of this test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, a broad range of tests will be exercised to achieve the goal. There will be following functions that can be performed by this application: Sales transaction, Inventory management, Employee management, viewing statistics. The user interface to utilize these functions is designed to be user-friendly and provide easy access to all the functions.

# Process Overview

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases shall be derived using the current Program Specification.

2. Identify which particular test(s) will be used to test each module.

3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.

4. Identify the expected results for each test.

5. Document the test case configuration, test data, and expected results.

6. Perform the test(s).

7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the Unit/System Test Report (STR).

8. Successful unit testing is required before the unit is eligible for component integration/system testing.

9. Unsuccessful testing requires a Bug Report Form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.

10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

## Testing Process

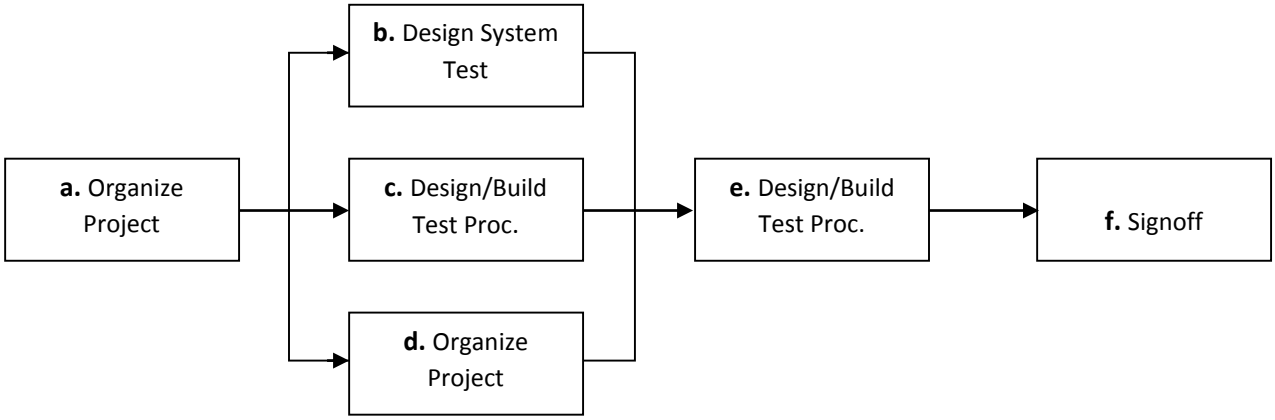


**Figure 1: Test Process Flow**

The diagram above outlines the Test Process approach that will be followed.

a. **Organize Project** involves creating a System Test Plan, Schedule & Test Approach, and assigning responsibilities.

b. **Design/Build System Test** involves identifying Test Cycles, Test Cases, Entrance & Exit Criteria, Expected Results, etc. Test Cases and the Data required are identified. The Test conditions are derived from the Software Requirement Specifications.

c. **Design/Build Test Procedures** includes setting up procedures such as Error Management systems and Status reporting.

d. **Build Test Environment** includes requesting/building hardware, software and data set-ups.

e. **Execute System Tests –** The tests identified in the Design/Build Test Procedures will be executed. All results will be documented and Bug Report Forms filled out and given to the Development Team as necessary.

f. **Signoff** - Signoff happens when all pre-defined exit criteria have been achieved.

# Testing Strategy

The following outlines the types of testing that will be done for unit, integration, and system testing. It includes what will be tested, the specific use cases that determine how the testing is done. The template that will be used for designing use cases is shown in Figure 2.

## Unit Testing

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. The unit test cases shall be designed to test the validity of the programs correctness. Unit testing includes testing two classes of the controller module, 6 classes of the data entities and the graphical user interface. Some of these functions are:

- addEmployee() EmployeeDatabase class
- editEmployee() EmployeeDatabase class
- changePassword() EmployeeDatabase class
- Login() EmployeeDatabase class
- getEmployee() EmployeeDatabase class
- addProduct() InventoryDatabase class
- updateInventory() InventoryDatabase class
- addBill() InventoryDatabase class
- getOverallProfit() InventoryDatabase class
- getOverallStats() InventoryDatabase class

## 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 cases are generated 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.

### 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 Error guessing and Boundary Value Analysis testing on our application.

## Integration Testing

### Incremental Testing

There are two primary modules that were needed to be integrated: the Graphic User Interface module and the controller module which connects to the database server (back-end). The two components, once integrated form the complete Supermarket Automation Software Application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

**Module 1 - Graphic User Interface (GUI) Module**

This module provides a simple GUI where the user can perform the different actions (functions). This module was tested separate from the backend to check if each interface (e.g. search button) is functioning properly, and in general, to test if the mouse-event actions were working properly. The testing was performed by writing a stub for each element in the interface.

**Module 2 – Controller Module (Client module)**

The Controllers provide functions to send requests to the servers and then obtain relevant data from the database to return to the GUI. This module was tested separate from the GUI by printing out the results to the Console. In testing this module we followed the incremental testing method i.e. testing one function first and then keep adding additional function and test it again until all the required functions are tested.

When the GUI is combined with the backend module, we will have a complete Supermarket Automation Software. To achieve complete integration of these two modules, we test each element in the GUI by replacing the stubs with the appropriate function from the back end. The results were displayed within the GUI instead of through the Console. In testing the combined modules, we followed the incremental testing

method. Each stub will be replaced one at a time and tested. This was done until all stubs have been replaced by the appropriate functions from the backend.

The following interfaces were tested:

- SalesClerkFrame
- SupermarketStaffFrame
- ManagerFrame
- Error messages and information messages (JOptionpanes)
- Statistics dialog
- Change password dialog

## System Testing

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system or some major part of it. Generally, system testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. But in our case we focused only on function validation and performance. And in both cases we used the black-box method of testing.

### Function Validation Testing

The integrated "SAS" was tested based on the requirements to ensure that we built the right application. In doing this test, we tried to find the errors in the inputs and outputs, that is, we tested each function to ensure that it properly implements the parsing procedures, and that the results are expected. The behavior of each function are contained in the Software Requirement Specification.

| **Function** | **Expected Behavior** |
|---|---|
| Login | see Software Requirement Specification |
| New Sales transaction | see Software Requirement Specification |
| View inventory | see Software Requirement Specification |
| Add inventory | see Software Requirement Specification |
| Update prices | see Software Requirement Specification |
| Manage employees | see Software Requirement Specification |

| | |
|---|---|
| View statistics | see Software Requirement Specification |

In addition, we tested:

- The interfaces to ensure they are functioning as desired (i.e. check if each interface is behaving as expected, specifically verifying the appropriate action is associated with each mouse_click event).
- The interaction between the GUI and the backend controller classes. In this case the data will be inserted and check if they are sent to the server properly and the expected results are obtained.

## Performance testing

This test was conducted to evaluate the fulfillment of a system with specified performance requirements. It was done using black-box testing method. Following things were tested:

- Adding large number of products to the database to see how much time it takes to retrieve them from the server.
- Logging in large number of employees from various terminals to test the maximum stress which the server can handle without substantial loss in performance.
- Calculating the sales statistics for a very large sales history to test the performance of chart generation.

## Features not to be tested

The feature of drawing charts related to sales and profit statistics is done using jfreechart library which has been thorough tested and multiply used by many people. Hence it need not be tested separately.