Distributed Polling System
Test Plan

Version 1.0

Table of Contents

1. Introduction

This document is the Software Verification and Validation Plan for Distributed Polling System (DPS). The project is conducted as a part of SCORE contest and distributed software development course for the year 2008-2009.

1.1 Purpose of this Document

This document describes methods and techniques that the DPS team is following to verify and validate the product being developed against customer's requirement. The document will help us to ensure that at each phase of product's development we follow our defined testing techniques and processes.

1.2 Scope of the document

This document describes the test plans and the test design techniques which are used to verify the DPS system.

1.3 Target Audience

The below is the list of intended audience for this document:

- DPS project team
- Customers and Supervisors
- SCORE reviewers

1.4 System Overview

Overall DPS interfacing architecture (Figure 1-1: System Interface Details) delineates the complexity of testing of the whole system.

Web-DPS application is the custom developed front-end web-based application responsible for fulfilling major functionalities of DPS (Figure 1-2: Layered Architecture Stack of web-DPS Application). Whereas SMS Gateway and Email server are the underlying applications responsible for fulfilling DPS product functionality i.e. integrated solution by sending email, SMS notification and receiving responses to/from the respective recipients.

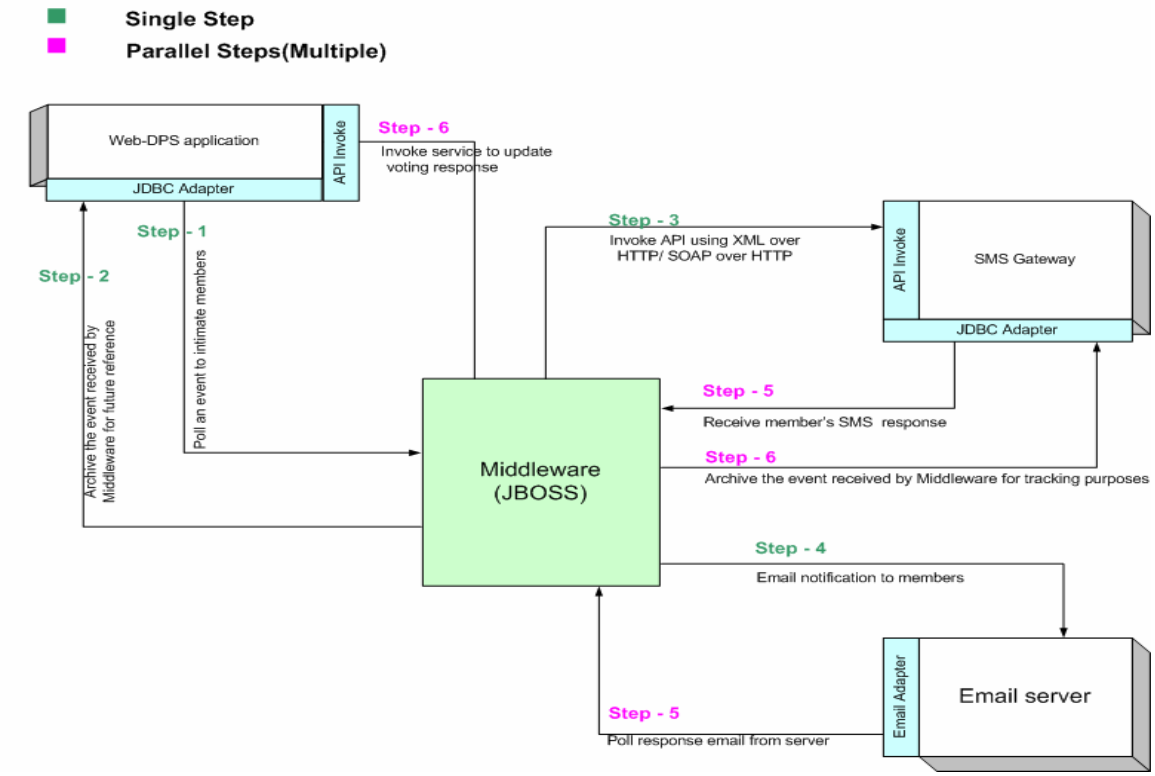

Figure 1-1: System Interface Details

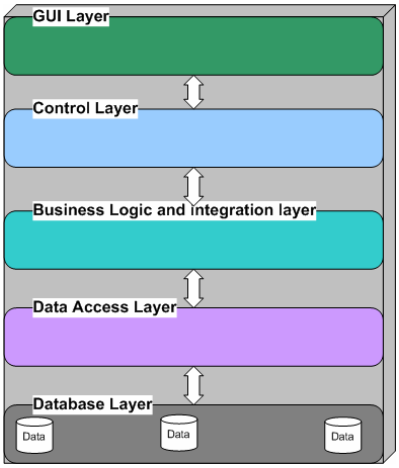

Figure 1-2: Layered Architecture Stack of web-DPS Application

1.5 High Level – Test plan considerations

Following are the aspects, in our considerations and analysis to proceed with test plan:

- Do we have any migration plan from legacy applications?
- Do we have stand-alone applications testing requirement?
- Do we have system's integration testing requirement?

Our analysis derives that it's a brand new development for DPS and no legacy data migration is required to our newly developed applications (like existing user accounts to web-DPS, existing email domains and accounts of Email) of DPS.
As web-DPS is a custom developed application and will be fulfilling major functionalities of DPS, we need to have stand-alone application testing requirement for web-DPS.
Email server and SMS Gateway are the subsidiary applications that are required to fulfill the product's behavior and functionality, we need to perform integration testing by invoking these two target applications from middleware layer (for request interface).
Now there are two ways of invoking these two target applications:

- Simulate from Jboss middleware layer as if actual data is flowing from web-DPS application (without any help from web-DPS application)
- Perform the actual data entry in the web-DPS application GUI and submit that data which in-turn will trigger the middleware interface to invoke target applications.

Similarly, for response interfaces email server and SMS gateway will act as source and web-DPS as target application.

1.6 Critical Success Factors

Following portrayed are some of the critical success factors that will influence our testing of the stand-alone application and the integrated product:

- Readiness of web-DPS, SMS Gateway and Email server applications by custom development and configurations
- Test suite availability for integrating all the application through middleware Jboss
- Completion of web-DPS custom development to trigger Jboss integration interface
- Resolution of connectivity establishment issue between middleware and the respective source-target applications
- Availability of test data
- Availability, support and collaboration of tester and developer to continue testing and bug fixing.

2. Testing Scope

The document entails both **stand alone application** and **integrated product** testing plan.
Following are the applications involved in testing scope of DPS:

- Web-DPS
- Email server
- SMS Gateway

We will be testing web-DPS custom built application in detail as it will be providing major functionalities from end-user perspective. Whereas Email server and SMS Gateway will be used as subsidiary application to facilitate the system functionality (to send/receive SMS to/from end user's mobile device and sending/receiving email to/from configured email recipients as attachment and so on).

Table 2-1: Features to be tested

S.No	Source System	Target System	Process Name	Interface Name
1.	Web-DPS	NA (means testing of stand-alone web-DPS functionality)	Managing members	Logging into web-page
			Managing members	Managing member's personal details
			Intimate poll details	Creating poll
			Intimate poll details	Opening and closing poll information
			Intimate poll details	Deletion of poll
			Intimate poll details	View poll results
			Verdict on Voting	Vote Calculation
			Verdict on Voting	Vote announcement on deadline
			Verdict on Voting	Vote announcement on hundred percent voting
			Verdict on Voting	Vote announcement on 51% voting on a single option
2.	Web-DPS	SMS Gateway	Intimate poll details	Poll intimation
			Intimate Voting Result	Result intimation
3.	Web-DPS	Email server	Intimate poll Details	Poll intimation
			Intimate Voting Result	Result intimation
4.	SMS Gateway	Web-DPS	Capture SMS Response	Capture SMS vote
5.	Email server	Web-DPS	Capture Email Response	Capture email vote

2.1 Web-DPS standalone application

DPS is a web-based application that is developed using the Struts MVC Model II architecture. The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user. Along with other benefits of this model, unit testing for each tier (presentation, data and business) is also easy to manage and to develop. Moreover, regression testing is also not a big deal to apply on a system for testing the rest of the application in case of any system update. In Figure 2-1: Struts- Request processing flow, DPS user request flow using this model is depicted.

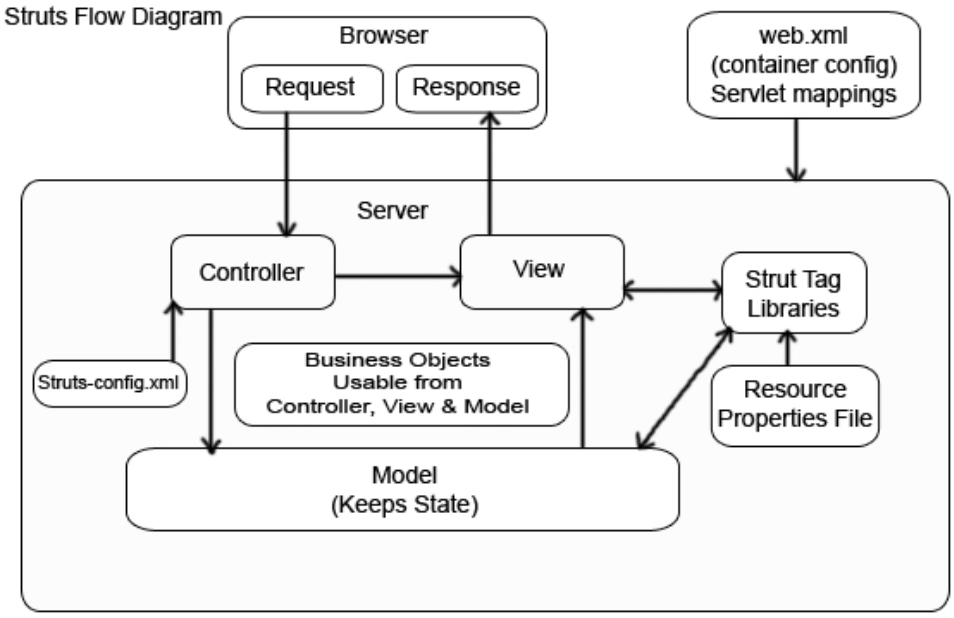

Figure 2-1: Struts- Request processing flow

DPS is a web based application designed and developed using the MVC pattern. Thus it consists mainly three tiers.

- Web tier
- Business tier
- Database tier

2.1.1 Web tier testing

In a typical enterprise application, many areas require testing. Starting from the simplest component classes, the developers or specialized test developers need to program unit tests to ensure that the application's smallest units behave correctly.

2.1.2 Database and business tier testing

During the development phase of DPS, we have used the Test-Driven Development approach. So before going for the actual implementation we have first designed and developed the unit test cases and then we have written the code (actual implementation) that passes the designed unit test cases.

3. Test Entry/Exit criteria

This section specifies the entry and exit criteria for DPS testing.
It specifies general pre-requisites for testing of a particular interface in DPS. From integrated product perspective, Figure 1-1: System Interface Details depicts the required interfaces and from web-DPS stand alone application perspective, it's the communication interface among each layer as depicted at Figure 1-2: Layered Architecture Stack of web-DPS Application.
Our project deliverable DPS Design Description Document specifies the use cases at high-level that portrays the entry and exit criteria for interface testing requirement for system's integration (from business process perspective) as below.
For Example:
Use Case-1

Goal	Intimating members with poll details through SMS and email
Actors	SMS Gateway, Email server, DPS Application
Triggering Event	As soon as an event is triggered in the DPS_EVENT_TABLE located at DPS application database
Entry Criteria	Once a poll is created at DPS application
Exit Criteria	Successful sending of SMS, EMAIL and status update at DPS_ARCHIVE_TABLE

Process Flow	As soon as an event is inserted in DPS_EVENT_TABLE, a JDBC listener that is always polling this table picks up the event and send it to integration ESB (enterprise service bus) where a service is subscribed to that event.
Success Conditions	When all the steps mentioned in the Business service flow diagram are successfully executed.
Failure Conditions	Please refer to DPS_Error_Handling Document

4. Test Pass/Fail Criteria

This section specifies the pass/fail criteria definition that will denote whether a test case has passed or failed the testing qualification, once it is executed.

A test case will be considered passed if it meets the following criteria:

- The test case meets the output requirement (once executed) as defined by the requirement and design
- All required steps should be executed for the particular test case
- No abnormal behavior pops up during execution of the test case

A test case will be considered failed if it meets any of the following criteria:

- All required steps are not executed during test case execution
- Does not meet output requirements
- Any side effect or abnormal behavior pops up during execution

For failed test cases there will be logging mechanism to log the failed test case as defect in software. For defect also there will be different level of severity definition as portrayed in below table:

Table 5-1: Severity Level

Severity Level	Description
Most Critical	It specifies an urgent call needs to be taken on the issue, high level management intervention required further steps can not be proceeded
Major	Same as above, but here a work around is available and fix could be provided
Medium	A defect that needs to be fixed as soon as possible prior to deployment
Minor	A defect and needs to be looked into
Enhancement	A suggestion for improvement that could be fixed any time

\

5. Verification and Validation Process

Verification and Validation activities and tasks are defined throughout the development life cycle of DPS.

5.1 Features to be tested

The tests cover the functional requirements of DPS as specified by the requirements definition, i.e. it is checked that the required functionality for executing all use cases is provided by the application.

5.2 Features not to be tested

Non-functional requirements such as scalability and few securities (like HTTPS, digital signature) are not verified. DPS uses features of the underlying frameworks (Struts, Tomcat, JBoss) for fulfilling these requirements. Jboss contains the security features built in so there is no need to develop and test these activities implicitly.

5.3 Testing Approach

High level approach for DPS testing activities are as follows:

1. Finalization of test scenarios, preparing test cases, test data and script for automation (if required)
2. Components developed in each member's laptop, needs to be tested by individual developer as per the output requirement.
3. Once the component is tested, it needs to be dropped in the integration testing environment for

composition and make applications, middleware ready for end-to-end testing. (Figure 5-1:)

4. For integration testing, we will have to ensure that the interfaces required to trigger middleware interfaces, are ready and tested during stand-alone application testing.
5. During simulation from middleware, to test target applications, availability of proper test data to synchronize target application configuration (like manually configured email accounts: middleware@middleware.com)

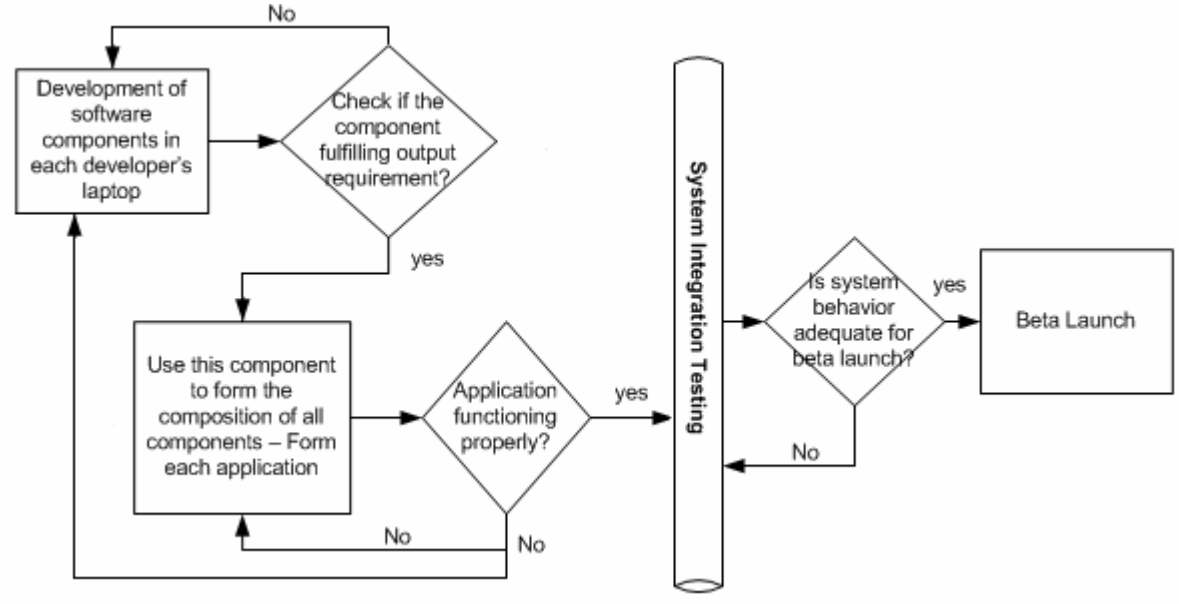

Figure 5-1: Test approach

We will follow to adopt the following testing approaches for each development components:

1. Sanity testing
2. Integration testing and then
3. bug fixing (as depicted Figure 5-1: Test approach)

Sanity testing ensures that the component or the composed component is can be used for integration as a next step.

Integration testing ensures fulfilling functionality from end-to-end perspective.

Bug fixing arises at each step whenever there is found any bug at component development or during composition of components or during integration testing. We believe that as soon as any bug is found in early phases of a DPS development-life-cycle, it will reduce the effort and malfunctioning of the software.

5.3.1 Requirements verification and validation

Requirements testing ensure that the requirements have been interpreted correctly, are reasonable, and are recorded properly. In software industry different approaches are being used for requirement verification and validation like: prototyping, tracing approaches, user manual writing, reviews and inspections. We used the Review approach for requirement verification and validation. In regards of this approach we read and analyze the requirements from the SCORE document, then conduct internal team meetings, discuss the problems and agreed on actions to address these problems. Meanwhile, we also arranged meetings with internal customer i.e. project supervisors to validate the requirements. As an external customer was not involved on site during development span but he was involved through electronic means of communication. For requirement validation we clarify requirements from the external customer through emails and he gave us feed back with some changes and finally after the changes were made it was verified and validated by internal customer. Finally, we were able to put requirement analysis to next level of development phase.

6. Test Environment

6.1 Software

6.1.1 Base technologies

Due to versatile nature of the architecture, scalability and for loose coupling design, a lot of software components need to interact with each other.

Following is the list of software components to be used for DPS system implementation.

Table 6-1: Software Component List

Sr no	Software Component	Product Suite	Version
1.	Jboss ESB	jbossesb-server	4.4
2.	Apache Ant	Apache	1.7.1
3.	JDBC Listener	jbossesb-server	4.4
4.	Email Listener	jbossesb-server	4.4
5.	MySQL DB	MySql	5.0
6.	Email Server	Winmail	4.6
7.	Eclipse platform	Development IDE	3.3.1.1
8.	Struts	Apache Struts, a flexible control layer based for web programming	1.3

6.1.2 Web browsers (client side)

DPS will be implemented so that it is fully functional in the following stable versions of the browsers:

- Mozilla Firefox 1.5 or higher
- Internet Explorer 6 or higher

6.1.3 Mobile Handset

- Nokia N95 8GB
- Sony Ericsson z550i

6.1.4 Email Client

- Eudora 7.1

7. Responsibilities

7.1 Developers

- To get the error reports from testers
- To fix the missing functionalities in the application
- To do initial testing of repaired application

7.2 User representative

- To check if the application is fulfilling his/her needs
- Report any error/bug encountered to tester

8. Test Deliverable

Following section will depict the deliverables from testing of the product.

8.1 Test Specifications

Test specification defines the test cases used for the software testing.

Table 8-1: Test specification

Sr. No	Test Specification File name	Description	Deliverable Phase
1	DPS test specification	Specification defines test cases for the features to be tested identified in the Test Plan	Implementation Phase

9. Risks and contingencies

In the testing, no real, important data should be used, as in testing phase, a bug can occur, which could lead to the data loss.