CSC408 Phase C

Team 15 – Nirvana

Integration Test Plan

Revision Sheet

Release No.	Date	Revision Description
Rev. 2	Dec 01, 2004	Test Plan (Integration)
Rev. 1	Nov 6, 2004	Test Plan (Integration)

TEST PLAN

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION 3
1.1 Purpose 3
1.2 Scope 3
1.3 System Overview 3
1.4 Project References 4
1.5 Acronyms and Abbreviations 4
1.6 Points of Contact 4
1.6.1 Information 4
1.6.2 Coordination 6
2.0 Test Definition 2-1
2.1 Input Test Data Control 2-2
2.2 Output Test Data Control 2-2
2.3 Data Recovery 2-2
2.4 Support Software 2-3
3.0 Test Execution 3-1
3.1 Test Schedule 3-1
3.2 Test Progression 3-1
3.3 Test Criteria 3-1
3.3.1 Tolerance 3-1
3.3.2 Samples 3-1
3.3.3 System Breaks 3-2
3.4 Test Control 3-2
4.0 CONNECTOR FEATURES 4-4
4.0.1 Nirvana Connector 4-4
4.0.2 OETeam08 Connector 4-4
4.0.3 Urn Connector 4-5
4.1 TEST TOOLS 4-5

1.0 GENERAL INFORMATION

1.0 GENERAL INFORMATION

1.1 Purpose

The purpose of this test plan is to verify the runtime correctness of OnmiEditor server component along with the Web Service interface exposed to be used by OmniEditor client. This document highlights the integration test plan conducted to verify the above along with non-functional requirement checklist for clients.

1.2 Scope

The scope of the integration test plan includes the following components:

- OmniEditor Web-Service Component
- Postgres Backend Database
- Web-Service client interface
- OmniEditor Connector API
- OmniEditor GUI Interface

1.3 System Overview

The Omni-Editor web-service component is a central repository of shared document resources. Client editors interact with this system to enable collaboration between users using heterogeneous text editors on heterogeneous platforms.

For the purpose of establishing compatibility with heterogeneous backends Nirvana provides a wrapper Connector API. This in essence provides a clean abstract connectivity to the frontend Nirvana GUI. Nirvana members implemented compatibility code for different CSC408 groups using the connector API for the purpose of evaluation.

Lastly Nirvana implements frontend Eclipse GUI composed of Eclipse plugin extensions. The GUI plugin interacts with Connector API to connect to a specified backend web-service.

This document presents the integration test plan that was employed for the system.

- Operational status

  Operational / Under development

1.4 Project References

The integration test plan is inspired from the following documents.

- *Phase A User Requirement Analysis*
- *Phase A Project Test Plan*
- *Phase C Component Testing*
- *Phase C Integration Testing*
- *Phase C Function Testing*

1.5 Acronyms and Abbreviations

- Nirvana – Project name for complete OmniEditor project.
- Integration Test Plan – Integration test plan for server component of OmniEditor.

1.6 Points of Contact

1.6.1 Information

Following are the points of organizational contact (POCs) that may be needed by the document user for informational and troubleshooting purposes.

Gilles Lising
Project Manager
647-287-9796 gvlvl@hotmail.com

Ashish Kaila
Test Team lead
416-247-4756 ashishkaila@hotmail.com

Sandy Lam
Quality Assurance Specialist – Project Nirvana
416-835-4845 sanlwy@hotmail.com

Dylan Durdle
Maintenance Lead
905-891-2136 99durdle@utsc.utoronto.ca

Mark Maglaque

Maintenance Assistant
416-979-8703 mark_rampage@hotmail.com

Yani Ioannou
Maintenance Assistant / IT Lead
yani.ioannou@gmail.com

1.6.2 Coordination

For the purpose of integration test plan, team Nirvana has a coordination plan between the project manager, QA and maintenance personals. The plan consists of reviews of outstanding functionality bugs and their addressing. Moreover details about specific support functions (e.g., installation coordination, security, etc.) will be addressed in the coordination meetings. Following is the tentative schedule of the meetings (excluding online group conversations):

November 11, 2004 4:00 – 6:00 pm SF-1101
November 18, 2004 4:00 – 6:00 pm SF-1101
November 25, 2004 4:00 – 6:00 pm SF-1101

2.0 TEST DEFINITION

2.0 TEST DEFINITION

For the development and testing purpose, Nirvana has adopted a test driven development approach. That is elementary functionality is first developed and tested and then dependencies follow the same fashion. This helps achieve two objectives:

- Solid ground work assertion – By testing elementary functions we ensure functionality is solid to grass root of code base. This helps achieve stability when add-on / complex functionality is developed on top of the elementary functionality.

- By adopting this bottom up approach, we can better design our product in the domain of proposed architecture. Historically it has been found that as the complexity of code increases, so does the tendency to deviate from the initial intended architecture. Test driven development clearly addresses this issue.

For the purpose of integration test following categories of test cases were performed:

- Requirements Validation – Functionality exposed by the OmniEditor fulfill the requirements of the users.
- Functional Testing – Tests to validate functions perform as required.
- Performance/Volume/Stress Testing – Tests to ensure service performs well under stress and high volume of requests.
- Security Testing – Tests to ensure security against buffer overflows and SQL injections that could prove to be crucial security holes.
- Ease of Use – Recommendations and viewpoints of third party taken into consideration while designing service interfaces and input/output to/from functions.
- Operational Testing – JUnit tests / thin client developed to simulate and test operational capabilities from clients.
- Interface Testing – heterogeneous clients from on a variety of platforms used to test web-service interface.
- Component Testing – Using test driven development, the author of code will provide test driver for the component. These test cases will be made available to other groups that can perform black box testing on the code and report bugs.
- Unit Testing – Unit test are performed to verify a complete isolated system's functionality. Basically this is an interface based black box testing initially. If a failure is detected, Nirvana members perform drill down component testing

to isolate the bug source. After the fixes unit testing is again performed to ensure fix does not fails other positive tests.

- Integration Testing – After successful unit testing of independent modules, their conjunctive working is tested via integration tests. Integration testing ensures that complete cycle of requirement workflows, having dependencies on several independent modules.
- Function Testing – At the basic level, function testing is performed via automated JUnit test cases. This way on an atomic function level we can compare observed results to the expected results and report anomalies.

Integration testing stressed heavily on web-service interface definition to ensure data types used are interoperable across the platforms. More specifically we used both Java and .NET clients to test the interface functionality with native data types from both platforms.

Also we implemented integration test cases for different connector implementations to verify stability tests for GUI. Also these integration tests reveal the actual functionality of connectors implementing backend of different groups against Nirvana's frontend.

2.1 Input Test Data Control

Input data are controlled to test the system with a minimum number of data types and values. The function was tested with a range of bona fide data types and values that test for overload, saturation, and other "worst case" effects and was tested with bogus data and values that test for rejection of irregular input. Also heavy sized inputs and string inputs with sql injections were tested on the interfaces.

2.2 Output Test Data Control

For the purpose of test, the backend data store was on a high performance test server and the interface test cases were tested from a variety of front ends (java / .net) imagining moderate client configuration. The output data are analyzed to detect whether an output is produced, output is evaluated as a basis for continuation of the test sequence and against the required output to assess system performance.

2.3 Data Recovery

The Postgres backend RDBMS ensures data integrity and reliability. Moreover RDBMS is configured to optimize data retrieval and CRUD operations. Thus Nirvana does not reinvent the wheel, rather delegates the data management responsibility to Postgres. Data retrieval during the tests were done both by the SaveDocument feature of the server and at times using a SQL Client to verify both versions.

2.4 Support Software

- Eclipse 3.0
- Microsoft Visual Studio.NET 2003
- Microsoft Internet Explorer 6.0
- JUnit

3.0 TEST EXECUTION

3.0 TEST EXECUTION

3.1 Test Schedule

JUnit Test suite is executed on daily basis and after every maintenance task. This test suite contains all the test cases mentioned above and also other primitive functions that form the execution basis for others. After test suite execution an automated report is generated that is then circulated among the team members and published on the team's website accessible to client groups.

Test Type	Start Date	Frequency
Function Testing	November 2, 2004	3 days
Component Testing	November 7, 2004	5 days
Unit Testing	November 15, 2004	5 days
Integration Testing	November 17, 2004	7 days
Stress/ Performance Testing	November 17, 2004	5 days
GUI Testing	November 21, 2004	2 days

3.2 Test Progression

JUnit tests have internal dependencies within each test case that simulates activity progression and real time request cycles Include an explanation concerning the manner in which progression is made from one test to another so the cycle or activity for each test is completely performed. This workflow based testing with interactive clients forms a perfect basis to ensure high performance and system durability.

3.3 Test Criteria

3.3.1 Tolerance

Maximum ranges of data types from various platforms used to test system tolerance. In case of network packet loss, tests were re-run to ensure system remains tolerant irrespective of network glitches.

3.3.2 Samples

Boundary conditions for data types included minimum and maximum values and special values (e.g. Double.NaN). Objects have boundary conditions as null values and also various states they can possibly acquire.

3.3.3 System Breaks

System was tested against interrupts, halts, and other system breaks which may occur because of non-test conditions and high network activity. Recovery mechanism was tested thoroughly via stress simulations. Also relative and absolute failure statistics were recorded via these simulations to determine relative uptime.

3.4 Test Control

Tests conducted were controlled by manual, semiautomatic, or automatic means. Randomized inputs were also derived from simulations for automated test purposes. Moreover an ASP.NET web application was used for performing interactive service tests to test asynchronous functionality.

4.0 CONNECTOR FEATURES

4.0 CONNECTOR FEATURES

As described before, Nirvana implements connector API for heterogeneous backends. This section compares features of different connectors, with respect to Nirvana connector.

4.0.1 Nirvana Connector

Function	Implemented
Connected	Yes
Close	Yes
ValidateConnection	Yes
CreateDocument	Yes
CloseDocument	Yes
LineLock	Yes
LineRelease	Yes
RetrieveDocument	Yes
ListDocuments	Yes
ListLineLocks	Yes
AddLine	Yes
DeleteLine	Yes
ReplaceLine	Yes
Sync	Yes
IsEditable	Yes
GetMinPollingInterval	Yes

4.0.2 OETeam08 Connector

Function	Implemented
Connected	Yes
Close	Yes
ValidateConnection	Yes
CreateDocument	Yes
CloseDocument	Yes
LineLock	Yes
LineRelease	Yes
RetrieveDocument	Yes
ListDocuments	No
ListLineLocks	Yes
AddLine	Yes
DeleteLine	Yes
ReplaceLine	Yes
Sync	Yes

IsEditable	Yes
GetMinPollingInterval	Yes

4.0.3 Urn Connector

Function	Implemented
Connected	Yes
Close	Yes
ValidateConnection	No
CreateDocument	No
CloseDocument	No
LineLock	No
LineRelease	No
RetrieveDocument	No
ListDocuments	No
ListLineLocks	No
AddLine	No
DeleteLine	No
ReplaceLine	No
Sync	No
IsEditable	Yes
GetMinPollingInterval	Yes

As can be seen clearly there is a number of feature shortcomings in non-native connectors. For all non-implementations we throw a NonImplementedException to indicate to GUI that the feature hasn't been implemented. For a list of complete test results and tests performed please refer to test document.

4.1 TEST TOOLS

For the purpose of automated testing, following test tools were employed:

- JUnit Test Cases – The module omni.client.core.connectors.testing.junit in the source branch was created to hold common JUnit test cases for all the connectors implemented. Since connectors implement connector API, we can perform test cases for all the connectors benefiting from the abstraction. Similarly OmniWebService.server.junit package contains test cases for the Nirvana backend webservice.

- Manual GUI Tests – GUI was tested manually by Nirvana members. Common wizard functionality was tested (note actual implementation is in connectors, hence we are only concerned with look and feel with some GUI presentation functionality in these tests), so was editor feature based testing.

- Test Summary Report – Automated test result summary was prepared by JUnit test cases and summarized in the OmniWebService.reports.JUnit namespace.

- Bug Tracking System – Nirvana depended on Bugzilla as bug reporting system. All the internal and external (by other groups) bugs are logged here and assigned by the Test lead to the author of the code. Apart from that team meetings include discussion on RFC and bug priorities and outstanding bugs.

4.2 BUG SIGNUP FORM

Bug # _____
Date Reported _____
Current Priority _____
Current Status _____

If Fix Intended, Fixer's Name _____

Outstanding Yes / No

If Outstanding State Reason _____

Components Involved Backend / Connector / GUI / Other _____

Test Type Black / White / Function / Component
Unit / Integration

Test Case Source JUnit Class _____
Manual Details _____
Other _____

Detail Test Summary
_____
_____
_____
_____
_____

Signature
_____