VMWare Virtual Network Visualizer

Software Test Plan

Version 3.0

November 28, 2012

Team
–Intentionally Left Blank–

Pierce Trey, Everett Bloch,
Jonathan Lamb, Daniel Palmersheim

Prepared For
UI CS 383—Software Engineering I
Instructor: Paul Oman, Ph.D.
Fall 2012

Template acquired from www.epmo.scio.nc.gov/library/docs/testplan.doc

Document Revision History

Revision #	Revision Date	Description of Change	Author
1.0	11/3/2012	Initial Document	ILB
2.0	11/11/2012	Updated Document	ILB
3.0	11/28/2012	Final Document	ILB

Table of Contents

1. Introduction

This Software Test Plan is designed to ensure proper functionality of the VMWare Virtual Network Visualizer being developed by team Intentionally Left Blank. This product is a tool used for creating a visual representation of a virtual network for the purposes of better understanding the behavior of said network and also for other assorted research purposes. This test plan is meant to create a system of testing this product to ensure that the information that is delivered to the user both in visual form and in raw data form is accurate and to ensure that the program itself functions as it is supposed to. As features are added to the product, it is intended that plans for testing each of these be added to this document in an organized fashion.

1.1 Objectives

The objectives of this test plan are as follows:

a. To deliver a working program to the customer with as much specified functionality as is possible
b. To ensure that every portion of this software is tested both individually, and in conjunction with all other portions until the entire product is tested as a unit
c. To ensure through regression testing that any minor changes to the product do not produce major errors

1.2 Testing Strategy

The overarching strategy that ILB is taking in testing this product is one of systematically analyzing the product in order to produce a set of expected test input, expected test output and expected operation along the way. This will be achieved by testing from the ground-up, starting with the smallest components that make up the program before proceeding to the next highest level of architecture and finally to the entire program as a whole.

1.3 Reference Material

SRS - Software Requirement Specification - Intentionally Left Blank

SDD - Software Design Document - Intentionally Left Blank

1.4 Definitions and Acronyms

ILB - Intentionally Left Blank

GUI – Graphical User Interface

JDK – Java Developers Kit

JRE – Java Run-time Environment

JUNG – Java Universal Network/Graph library.

XML – Extensible Markup Language

SRS - Software Requirements Specification

SDD - Software Design Document

IP - Internet Protocol

Node - A vertex on a graph that corresponds to a machine in a network

Edge - A relationship between two nodes (connectivity)

Weight - The distance between two nodes (length of an edge)

Capacity - The bandwidth/speed of an edge

2. Test Items

2.1 Program Modules

- Logging (SRS 3.2.7, SDD 3.1)
- Input File (SRS 3.2.1)
- Input Processing (SRS 3.2.2, SDD 3.2)
- Graph Generation (SRS 3.2.3, SDD 3.2-3.4)
- Graph Visualizer (SRS 3.2.4, SDD 3.5)
- Graph Algorithms (SRS 3.2.5, SDD 3.5)
- Saving a JPG (SRS 3.2.6, SDD 3.5)

2.2 Job Control Procedures

The Entry Class contains the main functionality for initializing and organizing the run time sequence of the product. This is a very small class, so testing of this will be ensuring that each individual module is initiated and that the program exits correctly. Any errors will be logged via the Data Logger Class to identify potential module failures and the severity of each logged item.

2.3 User Procedures

To ensure that the User Manual identifies the features identified in the SRS, and described in the SDD. ILB team members will do a meticulous walk through step by step of the instructions to insure that following the steps verbatim ends in a legitimate, understandable, usable result.

A second test of the User Manual will be to have an untrained user not affiliated with the project walk through the instructions and see if all functionality is properly described. To ensure that when all steps are followed a legitimate, understandable, usable result follows.

2.4 Operator Procedures

Testing will be performed on all of the machines of the developers, which include varying machine specs and operating systems (OSX, Linux and Windows environments are all included in this machine set). Using all of these machines ensure that a change of platform does not affect the operation of the application.

3. Features To Be Tested

- Logging
- Opening Input Files
- Input Processing
- Graph Generation
- Graph Visualization
- Graph Algorithms
- Saving to a JPG
- Node Information Display

4. Features Not To Be Tested

- JUNG Library: Already tested by its own developers and others that have used it
- Java Swing/Awt Graphics Libraries: Heavily tested by the Java community, standard functionality

5. Test Cases

This section describes the approach, test procedure, and pass/fail criteria for each test case. Every component test case is described in Component Testing, the merging of components is described in Integration Testing, testing the functionality of updated modules will be described in Regression Testing, and final approval of product tests will be described in Acceptance Testing.

5.1 Component Testing

5.1.1 Logger Test

Testing the logger will consist of repetitive and concurrent calls on all of its methods in order to stress test the synchronicity. The logger is static and will not change in terms of resources or its reference in memory. No concern is given to the garbage collector as the logger is only dereferenced upon program exit.

Pass/Fail Criteria

This test passes if the session log window contains the proper sequence of initialization logging info, of file processing info and any errors that occurred during the logging process. This information should be appended to the end of the log file, as well as saved in the current session log window.

This test fails if the data displayed in the log is either corrupt, out of sequence or incorrect.

5.1.2 Input Processing Test

We will be generating randomly connected node XML files with a separate utility that vary in both connectivity and in total number of nodes to test the theory of operation. The main goal of the input processing is to prohibit any two nodes from having two edges to each other. By testing various input files and comparing the log file with the theory of operation we will be able to confirm this process is correct.

Pass/Fail Criteria

This test passes if a valid input file is opened without throwing any errors to the logger, and if the data stored in the graph data structure matches that in the file, and if there are no nodes with two links between them.

This test fails if there are any errors in opening a valid file, or if there are no errors produced in opening an invalid file. This test also fails if the data stored in the graph data structure does not align with that in the input file.

5.1.3 Graph Visualization Test

Visualization test will occur simultaneously with the input processing test. Similarly, the test will consist of the visualization of randomly connected graphs and edges.

Pass/Fail Criteria

This test passes if the graph generated matches the given input of nodes and edges.

This test fails if the graph generated has nodes and edges that are improperly connected.

5.1.4 Graph Algorithms Test

Graph algorithm tests will be conducted in an ad hoc fashion, testing both valid and invalid node IDs and seeing if they correspond to the proper paths shown on the graph. In the case of invalid IDs, a message should be displayed identifying the error.

Pass/Fail Criteria

This test passes if the paths calculated with the algorithms correspond to the actual shortest respective unweighted/weighted path, and if any input errors are detected.

This test fails if a path is calculated that is in fact longer than an alternate shortest respective unweighted/weighted path, or if the dialogue window allows for invalid input without showing an error message.

5.1.5 Saving as a JPG Test

To test Saving JPG, multiple graphs will be saved to various locations to see if the saving process is accurate.

Pass/Fail Criteria

This test passes if the JPG file output is saved in the proper location in the system's directory structure, and if the JPG file matches the current size and configuration of the graph on the visualizer screen, including the presence/absence of labels, skew/rotation/zoom of the graph, and highlighting, placement of nodes and edges.

This test fails if the JPG file output is incorrect in that it does not match the current visualizer graph in any of the ways specified in the pass criteria. If the file is corrupt, saved under the wrong name, or in the wrong directory.

5.1.6 Node Information Test

Tests on the node info query feature will be conducted in an ad hoc fashion, testing valid and invalid node IDs and verifying the validity of the information via the visualizer. Invalid entries will result in a message displaying the error.

Pass/Fail Criteria

This test passes if the information displayed in the node window corresponds to the information specified in the input XML file. The information should also match what is displayed on the visualizer if this has already been tested and verified to be correct. The ID of the specified node and its IP must be correct, as well as all the IDs and IPs of neighboring nodes and the information about the edges connecting them. All neighbors must be present.

This test fails if any of the information in the node information window is incorrect. If any of the values are off, this test fails, and if any neighbors are missing for the specified node, the test fails.

5.2 Integration Testing

Integration testing will be conducted sequentially, combining each component in order of instantiation. Throughout all integration testing the Logger will be present to inform the tester of any errors.

5.2.1 Parsing and Graph Generation

First, the XML parser is integrated with the graph generator to see if the XML file can be correctly parsed and the parsed data can be used to generate the graph data.

Pass/Fail Criteria

The XML parsing fails only upon missing and/or invalid data. If the parsing passes then the node information is analyzed for back-linking issues. This process double-checks the XML data for logical errors, otherwise it's operations are fail-safe.

5.2.2 Graph Generation and Visualizer

Second, the graph visualizer will be integrated with the XML parser and graph generator. This will test that the generated graph data can be validly visualized via the visualizer.

This sets up the foundation of the program.

Pass/Fail Criteria

This test passes if the graph is generated upon a successful pre-processing return. It is assumed that JUNG will function as expected. This process should exit if any errors were encountered during the node data pre-processing.

This test fails if the graph is not generated given valid input, or if it does not exit upon a fatal error.

5.2.3 Graph Foundation and Algorithms

From here we can integrate the algorithms into the foundation, to test that the algorithms can correctly identify paths in an integrated environment.

Pass/Fail Criteria

This test passes if the algorithms function correctly given any form of input, including invalid parameters.

This test fails if the algorithms produce data that does not match what is seen in the visual graph.

5.2.4 Graph Foundation and Node Data Display

Next we can integrate the Node Info into the foundation, to test that node data can be correctly extracted.

Pass/Fail Criteria

This test passes if the information about a specified node in the sidebar matches to what is displayed in the visual graph.

This test fails if the information displayed about a given node is different than that which is displayed visually in the graph.

5.2.5 Graph Foundation and JPG Generation

Lastly we can integrate the Save as JPG feature into the foundation, to see if the visualizer is saved correctly.

Pass/Fail Criteria

This test passes if the JPG image created is saved to the proper location and if the image matches to what is displayed in the visual graph in the program.

This test fails if the JPG image generated is either saved to an improper location, is corrupt, or if it contains a version of the graph that does not match the configuration of the graph at time of image creation.

5.3 Regression Testing

Procedure for regression testing is as follows:

1. Perform component tests on modified components.

2. Run through integration test to verify that modified components do not impact overall functionality.

5.4 Acceptance Testing

Acceptance testing will consist of sitting down with our customer to show them each feature of the product, to show that all features are working together correctly. Upon completion the customer will either accept or reject the product. Acceptance will consist of signing a document indicating their approval of the final product. Rejection will result in identifying features to add/fix, and modification of features appropriately. Before another acceptance test, a regression test will be run to confirm functionality of the product.

6. Environmental Requirements

(Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.)

6.1 Hardware

Any hardware capable of running JRE 7+ should be able to run this program given enough (approx. 100 Mb) of ram. Refer to SDD 2.0 Deployment Diagram and http://www.java.com for system requirements.

6.2 Software

Any computer running JRE 7+ should be able to run these testing procedures.

6.3 Tools

A XML document generator tool is used to generate an XML document with a specified number of nodes and paths to aid in the testing of XML input and parsing, and regression tests.

6.4 Risks and Assumptions

All these tests assume the proper location of libraries and files that this software depends on. These tests also assume that enough system resources are given the program to successfully operate and complete all tests unto passing or failure.

7. Change Management Procedures

Team members may add to or modify any portion of this document that pertains to their respective tasks. It is not alright for a team member to change a portion of this plan that is meant to test another team member’s code without consulting them first.

8. Traceability Matrix

Tests (5.x) Features	1.1	1.2	1.3	1.4	1.5	1.6	2.1	2.2	2.3	2.4	2.5	3
Logging	✓											✓
Opening Input File		✓					✓					✓
Input Processing		✓					✓					✓
Graph Generation							✓	✓	✓	✓	✓	✓
Graph Visualization			✓					✓	✓	✓	✓	✓
Graph Algorithms				x					✓			✓
Saving to a JPG					✓						✓	✓
Node Info Display						✓				✓		✓

✓ = Test Passed x = Test Failed ✓ ∪ x = tests that apply to given feature

This Page Intentionally Left Blank