CS2043 Project - Test Documentation

Alec Sobeck
Kieran Lea

June 20, 2014

1. Introduction

1.1 Document Identifier

This document follows the naming convention established with the design and requirements phases – *[Phase] Document – Team 2.pdf.* This document makes extensive reference to the *Requirements Document – Team 2* for the functional requirements, non-functional requirements, and use cases defined in that document. Tests are also based off the sequence flow diagrams and other information established in *Design Document – Team 2.pdf*.

1.2 Scope

This document contains all the details of the system tests – Plan, Cases, and Report - merged into one document. All the information regarding the tests is therefore contained in this one document which is based on the *IEEE ISO/IEC/IEEE 29119-3 Test Documentation Standard.*

2. System Test Plan

1. Test Classes and Overall Test Conditions

Boundary class testing occurs for the following methods:

- BillboardList::get(int)
- BillboardList:set(int, int, String)
- BillboardList::remove(int)

Equivalence classes exist for the following methods:

- CharacterHelper::isValid(String) has the following equivalence classes:
  - Strings composed entirely of characters, digits, and valid special characters
  - The empty string
  - Control characters
- BillboardList::load(File) has the following equivalence classes:
  - Files that exist on disk
  - Files that don't exist on disk
  - Files that do not have a name ending in .xml (files with invalid names)
  - Files that do not contain a valid XML structure (files with invalid contents)
- BillboardList::save(File) has the following equivalence classes:
  - Files that exist on disk
  - Files that don't exist on disk
  - Files which do not have a name ending in .xml (files with invalid names)

Overall Test Conditions:

Two different types of testing will be performed to ensure that the GUI and all underlying code work:

1. The first is manual testing. This will be done to ensure the GUI functions properly and updates with the relevant information or error messages as expected. This is done manually because it's incredibly difficult to write unit tests for. The general test conditions for this will be that the software is open to the appropriate GUI, then the human user will follow one of the use cases and ensure it provides the expected output to the GUI.
2. The second is unit testing using JUnitTest. These tests will be conducted without regard for how the GUI is updated. The purpose of these tests is to ensure that individual methods like those responsible for file I/O or the methods of the BillboardList perform as expected given predetermined sample inputs.

Details for this Level of Test Plan:

<u>2.1 Test Items and Their Identifiers</u>

1. Non Functional Requirements

    1.1 File I/O completes in less than 10 seconds 95% of the time.

1.2 Adding a new element, removing an existing element, and modifying an element complete in less than 3 seconds 95% of the time.

    1.3 Search completes in 6 less than seconds 95% of the time

1.4 Billboard information may be searched for using either the title or location

2. Use Cases

    2.1. The user may create a new BillboardList within the program's memory

        2.1.0 Main-Case: Expected behaviour

2.2. The user may insert a new Billboard into the BillboardList and GUI
- 2.2.0 Main-Case: Expected behaviour
- 2.2.1 Alternative-Case: Billboard data not fully entered
- 2.2.2 Alternative-Case: Invalid Characters Entered
- 2.2.3 User Declines to Insert

2.3. The user may remove a billboard from the BillboardList and GUI
- 2.3.0 Main Case: Expected Behaviour
- 2.3.1 Alternative-Case: No rows to remove

2.4. The user may modify a billboard
- 2.4.0 Main-Case: Expected Behaviour
- 2.4.1 Alternative-Case: Invalid Characters entered

2.5. The user may able to search for billboards by location or title.
- 2.5.0 Main-Case: Expected Behaviour
- 2.5.1 Alternative-Case: Duplicates found
- 2.5.2 Alternative-Case: Information entered was not found

2.6. The user may save their billboards as an XML File, using the last save destination.
- 2.6.0: Main-Case: Expected Behaviour
- 2.6.1: Alternative-Case: No BillboardList in memory to save
- 2.6.2: Alternative-Case: File path not found
- 2.6.3: Alternative-Case: BillboardList never saved before

2.7. The user may save their billboard information as an XML file, to a location of their choice.
- 2.7.0: Main-Case: Expected Behaviour
- 2.7.1: Alternative-Case: User Declines To Save
- 2.7.2: Alternative-Case: Saving Fails

2.8. The user may open a billboard information XML file they have previously saved to disk.
- 2.8.0 Main-Case: Expected Behaviour
- 2.8.1 Alternative-Case: User declines to open a file
- 2.8.2 Alternative-Case: User selects an invalid file format
- 2.8.3 Alternative-Case: File Not Found

2.2 Test Traceability Matrix

This has been included separately as the *Team 2 - Traceability Matrix and Test Data.xls* file.

2.3 Features to be tested

2.3.1 Features to be tested manually
1. The GUI – specifically to ensure that it updates as expected in response to button clicks or other user input

2.3.2 Features to be tested using unit tests
1. XMLReader's readBillboards() method
2. XMLWriter's write() method
3. All methods of the BillboardList class
4. CharacterHelper's isValid(String) method

2.4 Features not to be tested

All features will be tested either manually or with unit tests, unless they are private methods within a class. Therefore, the only code not tested directly is the private methods of CharacterHelper.

2.5 Approach

Tests will be conducted using both a white box and black box approach:

- The unit tests will all be conducted using a whitebox approach to write a proper suite of unit tests, which also correspond to the use cases within the *Requirements Document – Team 2*, and any elaboration from *Design Document – Team 2.pdf*.
- Black box testing will be used to test that the GUI responds and updates as intended. A tester will be able to see only the GUI for the program and the XML files they read/write using the program while testing. Using just this information it will be up to the tester to ensure that the operations have done what they should've and produced a correct output and updated the GUI appropriately.

2.6 Item pass/fail criteria

Pass/fail criteria will depend on the test. Each test case in section 3 will define an expected result for the operation and a test will pass if it fulfils this expected result (with no adverse side effects).

3. Use Cases or Data Management Test Cases

This is included in the *Team 2 - Traceability Matrix and Test Data.xls* file.

Test case naming convention:
TC-[*Methodology_Identifier]-[Relevant_Identifier]*
Methodology_Identifier: 'U' for a unit test, 'M' for a manual test
Relevant_Identifer: corresponds to the requirement the test is based on. For example, 1.3.1 would be the first test related to requirement 1.3

4. Minor fixes during the implementation phase

General changes:

- Changes XMLWriter::write(...) so that it takes a File not a String for consistency
- Changes XMLReader::readBillboards(...) so that it takes a File not a String for consistency
- Billboard::remove() now throws an IndexOutOfBoundsException to handle one of the alternative cases
- An IllegalFileFormatException was added to handle the case where a file has an invalid format.

Changes to use cases:

- Use case 2: Alternative Case (i) is obsolete – there is no way to invoke this functionality
- Use case 2: Alternative Case (iii) is obsolete – there is no way to invoke this functionality
- Use case 5: Alternative Case (i) is obsolete – there is no functional difference between it and the sunny-day case
- Use case 5: Alternative Case (ii) is obsolete – there is no functional difference between it and the sunny-day case