EECS2311: SOFTWARE DEVELOPMENT PROJECT

TAB2XML

Testing Document

March 6, 2022

PREPARED FOR

Vassilios Tzerpos
Lassonde School of Engineering

PREPARED BY

Hiba Jaleel - 215735020
Kuimou Yi - 216704819
Kamsi Idimogu - 216880288
Maaz Siddiqui - 216402927

Table of Contents

Introduction

The purpose of this document is to provide general information about the test cases that were generated and used to debug and improve the user-experience for our software, the TAB2MXL. These test cases helped ensure that the TAB2XML software will run successfully.

This application is a java-based program, implementing the graphical user-interface which generates a machine-friendly XML file, along with visual and sound information. The information in the XML file allowed us to create a music sheet generator and player that will be easy to understand by users.

Since the project is based on well-tested starter code, this document will focus on two stages of additional functionality: the music sheet generator and the music player.

Objective and Tasks

The objective of the testing procedures are to ensure that the users will have error-free experiences when they are using the software.

To reach this objective, testing procedures should complete the following tasks:

1. Create test cases which simulate user input and compare that to the correct desired output to ensure that the program will handle user input correctly.
2. Create test cases which target specific parts of the program which may cause errors, so that the program can be designed to handle those errors correctly.

Testing Strategy

General Approach

Our testing is based on the Junit framework and manual validation of results which are not machine friendly.

Our testing strategy include two different types of approach:

a) **The Overall Test Approach**
   This approach will consider the whole system as a “black box” where we only know the input and output. Test output will be generated automatically, but the output will be checked manually.

b) **Unit Test Approach**
This approach will break down different parts of that system into “units.” Only the possible inputs and outputs of that specific unit will be taken into consideration. The test output will be generated automatically and validated by the program automatically

Due to limitations and the costs of the program validation, the unit tests will not cover the methods that generate human-friendly information, such as the visual and sound components.

In order to conduct the program validation, we are using the Junit test, assertion function, to compare the system output with the desired output.

Overall Test Approach

Music Player Test

Testing method:

PlayerOutputTest.java will test the sound results generated with specific user input. A list of .txt files that contain user input will be sent into the system and run through the following test cases:

1. **Generate Music String**
    a. Our player is based on the JFugue library which requires a music String to play. This test will generate music strings automatically for group members to check the results manually, and ensure the desired results are met.
2. **Generate Sound (Multi-Thread)**
    a. The purpose of this test is to check whether the program will crash due to the music player feature. This test will generate sounds in parallel to ensure that the testing won’t be limited by super long test instances.
3. **Generate Sound (Single-Thread)**
    a. The purpose of this test is to check if the music player generates sound correctly. It will generate sounds one by one so that the programmer can compare the sound generated through the music player with the desired sound.

Each of the above test cases will be conducted using 11 .txt files as user input to ensure that it covers a variety of possible user inputs.

Visualizer test

Testing method:

VisualizerOutputTest.java will test the music sheet previewer and the visualized result generated using specific user input. A list of .txt files that contain various possible user inputs will be sent into the system and run through the following test cases:

1. **Generate PDF File**
    a. The purpose of this test is to check both the file export function and the visualized music sheet output. Each user input will be saved as an individual .pdf file into designated locations for the programmers to compare the outputs with the desired results.

Each of the above test cases will be conducted using 11 .txt files as user input to ensure that it covers a variety of possible user inputs.

Unit Test Approach

Player Test

Player Internal Test

The approach of the unit tests inside of the player.java class is to ensure that each part of the music string is generated correctly, helper methods are working as intended, and to ensure that the system does not crash with null output.

i. NoteConverterTest

1. The purpose of this test is to ensure that the player will convert the notes in the musicXML representation into a Jfugue representation successfully, and the convert method works with the null note type.

2. Test method

a) The player.getNoteDuration method will get null and different note types as input and the Junit will compare the system output with the desired output

ii. instrumentConverterTest()

1. The purpose of this test is to ensure that the player will convert the instrument name in the musicXML representation into a Jfugue

representation successfully, and the convert method works with the null note type.

2. Test method

    a) The player.getInstrument method will get different types of instrument id as input and Junit will compare the system output with the desired output.

iii. NoteDetailUnpitched()

1. The purpose of this test is to ensure that the player will generate the correct music String for a single note. This test will focus on notes with unpitched attributes.

2. Test method

    a) The player.getNoteDetails method will get different types of note with unpitched attributes as input and Junit will compare the system output with the desired output.

iv. NoteDetailpitched()

1. The purpose of this test is to ensure that the player will generate the correct music String for a single note. This test will focus on notes with pitched attributes.

    a) The player.getNoteDetails method will get different types of notes with pitched attributes as input and the Junit will compare the system output with the desired output.

v. NoteDetailInstrument()

1. The purpose of this test is to ensure that the player will generate the correct music String for a single note. This test will focus on notes with instrument attributes.

    a) The player.getNoteDetails method will get different types of notes with instrument attributes as input and the Junit will compare the system output with the desired output.

Visualizer Test

ImageResourceHandler Test

ImageResouceHandler is a part of the program that will read image asserts from the disk and send them to the visualizer. The imageResourceHandler will be tested with the following test cases.

i. ResourceLoadTest:

1. The purpose of this test case is that imageResourceHandler will load a json file which contains the path of image asserts. This test is focused on checking if the imageResourceHandler has loaded the json file successfully and stored the path of image assert into the designated data structure.

   Test method: We will check the size of the internal map after loading.

ii. ResourceURLNonNullTest

1. The purpose of this test case is to check whether the path under the json file has been read successfully after imageResourceHandler loads the json file.

   Test method: We will check if the value in the map is not null.

iii. resourceNonNullTest

1. The purpose of this test case is that after ImageResouceHandler loads the path of assert, imageResourceHandler also responds by loading image data from the assert path. We will check if the loaded file is not null too.

   Test method: We will check the returned image from getImage method is not null.

iv. resourceNullTest

1. The purpose of this test case is to ensure that the resourceHandler is establishing one-to-one mapping on the assert and it's id.

Visualizer internal test

The approach of unit testing inside of the Visualizer class is to ensure that testable coordinates are generated correctly, helper methods work as intended, and make sure the system does not crash with null input.

i. initPDFtest()

1. The purpose of this test case is to ensure that the visualizer will open the desired pdf file successfully so that the visualizer can draw in it.

   Test method: call initPDF() method in the visualizer.

ii. getRelativeTest()

1. The purpose of this test case is to ensure that this method provides correct relative distance between current drawable elements and the baseline position. This method get step and oct than return the relative position compare with “central C”

   Test method: List all possible steps and oct, compare the resulting relative with desired relative.

iii. getMeasureLengthTest()

1. The purpose of this test case is to ensure that this method provides the correct length of staff line.

   Test method: List all possible steps and oct, and compare the resulting relative with the desired relative.

iv. MeasureAttributesTest()

1. The purpose of this test case is to ensure that desired attributes (time, clef and staff lines) have been applied in the current measure setting.

   Test method: Assert equal between the internal measure and the external measure.

v. drawNullElementstest

1. The purpose of this test case is to ensure that nullable elements in the musicXML file will not cause the program to crash.

   Test method: pass elements with null parts into the draw calls.

vi. switchLine/Page test

1. The purpose of this test case is to ensure that after the switchline/page, elements will be drawn in the desired locations.

   Test method: compare the coordinate after switchline/page with desired coordinate.

Code Coverage Rate Report

As shown in the image below, our test cases cover over 89% of methods in the software visualizer and over 85% of methods in the software music player. This is sufficient for programmers to identify problems in the program and provide a solid and stable experience to the end-user.