Testing Document for the Music XML Converter

LE/EECS 2311 - Software Development Project

Prepared for: Vassilios Tzerpos

Prepared by: Group 16

Maisha Rahman 215083876
Nabaa Gaziy 215820095
Wenxuan Li 216374324
Ali Sheikhi 216777120
Lars Jaylen Palalon 217280041

February 28, 2021

LIST OF CONTENTS

1.0 Introduction

MusicXMl Converter is a program designed for musicians and music enthusiasts to convert guitar, bass, and drums tablatures text files into an .xml file to be used and played by other music applications such as MusScore. This document is provided for the user to test the quality of the system. Please do not hesitate to contact us if you have any questions or concerns at xmlconvertor@gmail.com.

1.1 The Requisites

To start the test execution the following matters need to be present:

- ❏ Eclipse and JDK must be installed
- ❏ Gradle is installed in Eclipse

Minimum system requirements:

- ❏ Must use Windows, Mac, or Linux operating systems

2.0 Objectives and Tasks

2.1 Objectives

This document is written for the user to understand how the functionality of the system was tested (refer to **Sections 4.1** and **4.2**). Also, the user can use the steps and guidelines in **Section 4.4** to test the Converter themselves. The purpose of this test (in **Section 4.4**) is to confirm that the system is ready for operational use.

2.2 Tasks

This document offers the user to:

- ❏ Convert multiple tablatures, to test the functionality of the program.
- ❏ Take note of the seed of the program.
- ❏ Report any problems that can occur.
- ❏ Take note of the experience the program offers.

3.0 Scope

3.1 General

MusicXML Converter is the software being tested. The graphical user interface (GUI) for the Converter is a window with options to *Load*, *Translate*, and *Save* translated text tablature. This document will guide you in testing the GUI of the Converter (in **Section 4.4**). Also, this document will talk about the test cases implemented in the Java project for the Converter (in **Section 4.1**) and how these tests are sufficient in showing that the internal functionality of this Java project is correct (in **Section 4.3**).

3.2 Tactics

The procedure to test the GUI of the Converter is:

- ❏ Check if it is possible to load text tablatures into its text window.
    - ❏ We will test using bass and guitar text tablatures.
- ❏ Then, check if it is possible to translate the tablature into xml format.
- ❏ Finally, check if the file can be saved and exported into the user's system.

4.0 Testing Strategy

User Groups

Musicians and music enthusiasts.

4.1 Unit Testing and Description of Test Cases

Various JUnit Test cases have been developed for the Music XML Converter. This is to test that the internal functions (ie. the methods) of the program are working properly.

All the test cases are in the StringInstrumentTest.java file in src/test/java directory. Here are the descriptions of some of its important test cases:

Test Case #	**Test Title**	**Test Summary (how it is derived)**	**Test Steps (the way it is implemented**	**Why is this Test Sufficient**
1	testxmlHeading4()	Tests if the header for the MusicXML output is printed properly	This test case compares the expected xml heading to the actual heading being printed	Will pass only if the actual heading is correct.
2	testPrintToXML()	Tests if the body for the MusicXML output is printed properly	This test case compares the expected xml body to the actual body being printed	Will pass only if the actual body is correct.
3	testEndHeading()	Tests if the ending for the MusicXML output is printed properly	This test case compares the expected xml ending to the actual ending being printed	Will pass only if the actual ending is correct.
4	testType4()	Tests if the getter for the type of	This test case compares the	Will pass only if the expected

		bass instrument (ie. 4-string bass or 5-string bass) is working properly	expected value of the type of the bass to the actual type of the bass	value of the type of bass is the integer 4.
5	testType5()	Tests if the getter for the type of bass instrument (ie. 4-string bass or 5-string bass) is working properly	This test case compares the expected value of the type of the bass to the actual type of the bass (which in this test case, the type is 5-string bass)	will pass only if the expected value of the type of bass is the integer 5
6	testString1()	Tests the getter for the string (in this case, it tests the getter for the first string).	This test case compares the expected first string of the tablature with the inputted string.	Test case will pass only if the expected string is equal to the inputted string
7	testString2()	Tests the getter of the second string	Compares the expected second string of the tablature with the input	Test will pass only if they expected match the input
8	testString3()	Tests the getter of the third string	Compares the expected third string of the tablature with the input	Test will pass only if they expected match the input
9	testString4()	Tests the getter of the fourth string	Compares the expected fourth string of the tablature with the input	Test will pass only if they expected match the input

All the test cases are in the DrumTest.java file in src/test/java directory. Here are the descriptions of the most important test cases:

Test Case #	Test Title	Test Summary (how it is derived)	Test Steps (the way it is implemented	Why is this Test Sufficient
1	testDrumXMLHeading 5()	Tests if the header for the MusicXML output is printed properly	This test case compares the expected xml heading to the actual heading being printed	Will pass only if the actual heading is correct.
2	testPrintDrumXML()	Tests if the body for the MusicXML output is printed properly	This test case compares the expected xml body to the actual body being printed	Will pass only if the actual body is correct.
3	testEndDrumHeading()	Tests if the ending for the MusicXML output is printed properly	This test case compares the expected xml ending to the actual ending being printed	Will pass only if the actual ending is correct.

All the test cases are in the MeasureTest.java file in src/test/java directory. Here are the descriptions of its most important test cases:

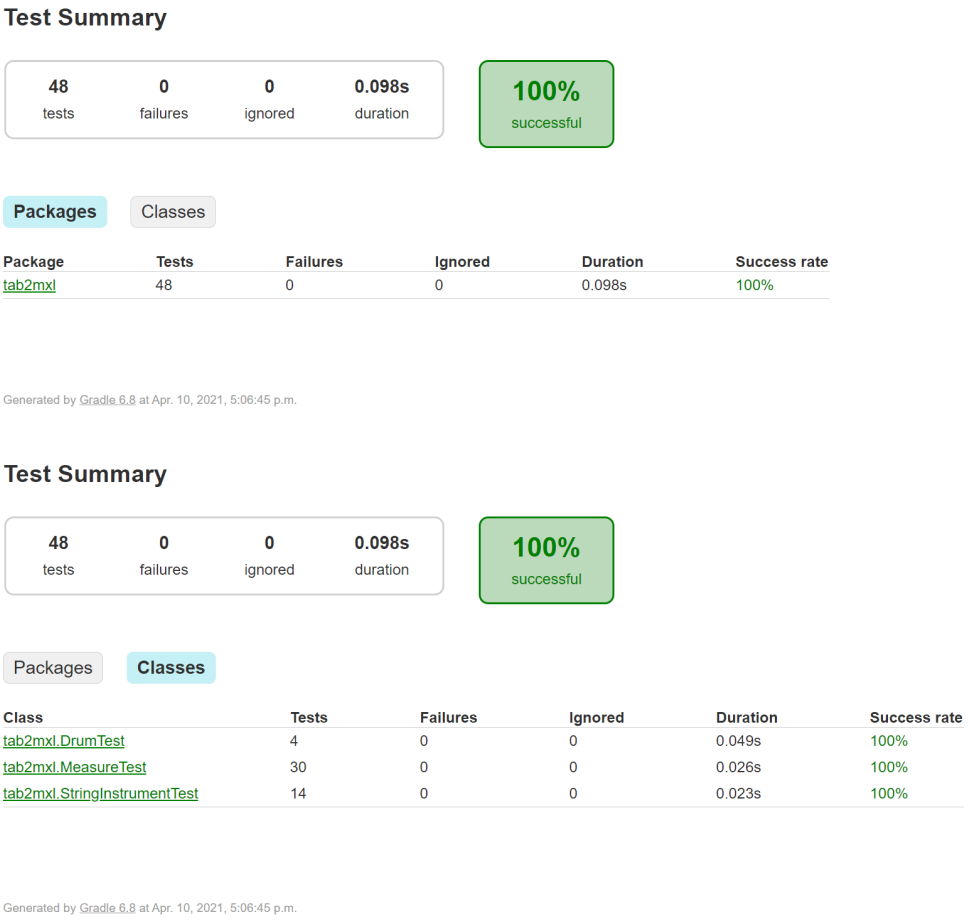
Test Case #	Test Title	Test Summary (how it is derived)	Test Steps (the way it is implemented	Why is this Test Sufficient
1	testLegato1()	Tests if it is able to detect if a note is a hammer-on	This test case compares the expected hammer-on notation to the actual notation being printed	Will pass only if the “hammer-on” is printed if the note contains a “h”.
2	testLegato2()	Tests if it is able to detect if a note is a pull-off	This test case compares the expected pull-off notation to the actual notation being printed	Will pass only if the “pull-off” is printed if the note contains a “p”.

3	testGuitarNote1()	Tests if the correct guitar note is being printed.	This test case compares the expected guitar note to the actual guitar note being printed	Will pass only if the actual guitar note is correct.
4	testBassNote1()	Tests if the correct bass note is being printed.	This test case compares the expected bass note to the actual bass note being printed	Will pass only if the actual bass note is correct.
5	octaveCheck1()	Tests if the correct octave for a bass note is printed	This test case compares the expected octave to the actual octave being printed	Will pass only if the actual bass octave is correct.
6	octaveCheck2()	Tests if the correct octave for a guitar note is printed	This test case compares the expected octave to the actual octave being printed	Will pass only if the actual guitar octave is correct.
7	octaveCheck3()	Tests if the correct octave for a drum note is printed	This test case compares the expected octave to the actual octave being printed	Will pass only if the actual drum octave is correct.
8	testDrumNote1()	Tests if the correct drum note is being printed.	This test case compares the expected drum note to the actual drum note being printed	Will pass only if the actual drum note is correct.
9	testDrumInstrment1()	Tests if the correct drum instrument is being printed.	This test case compares the expected drum instrument to the actual drum instrument being printed	Will pass only if the actual drum instrument is correct.

Overall, these test cases are sufficient because they show if the internal functions (ie. the methods) of the program are working properly. By implementing these test cases, we know that the getters and setters for the strings and for the type of the bass or guitar are working properly.

4.2 Test Summary Report

To access the test summary report, click on this link:
file:///C:/Users/maish/git/demo2311/TAB2MXL/build/reports/tests/test/index.html



Test Summary

48 tests	0 failures	0 ignored	0.098s duration	100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
tab2mxl	48	0	0	0.098s	100%

Generated by Gradle 6.8 at Apr. 10, 2021, 5:06:45 p.m.

Test Summary

48 tests	0 failures	0 ignored	0.098s duration	100% successful

Packages | Classes

Class	Tests	Failures	Ignored	Duration	Success rate
tab2mxl.DrumTest	4	0	0	0.049s	100%
tab2mxl.MeasureTest	30	0	0	0.026s	100%
tab2mxl.StringInstrumentTest	14	0	0	0.023s	100%

Generated by Gradle 6.8 at Apr. 10, 2021, 5:06:45 p.m.

Figure 1. Test summary report of all test cases generated by Gradle for the TAB2MXL program.

4.3 Test Coverage Report and Why This Testing is Sufficient

To see the full test coverage report for the TAB2MXL program, click on this link:

file:///C:/Users/maish/git/demo2311/TAB2MXL/build/customJacocoReportDir/test/html/tab2mxl/StringInstrument.html

TAB2MXL > tab2mxl

Source Files Sessions

tab2mxl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
window		0%		0%	28	28	352	352	7	7	1	1
TablatureScanner		0%		0%	76	76	153	153	6	6	1	1
StringInstrument		63%		52%	72	114	105	406	16	31	0	1
Drum		66%		46%	71	105	113	351	6	14	0	1
Functioncallfile		0%		0%	33	33	136	136	15	15	1	1
Notes		67%		57%	63	111	82	201	1	9	0	1
loadingScreen		0%		0%	14	14	50	50	5	5	1	1
Editmeansure		0%		n/a	4	4	35	35	4	4	1	1
newtry		0%		n/a	3	3	46	46	3	3	1	1
Guitar		20%		0%	10	14	16	22	1	5	0	1
newwindow		0%		n/a	4	4	26	26	4	4	1	1
Bass		21%		0%	8	12	12	18	1	5	0	1
outputOrgTablature		0%		0%	5	5	23	23	3	3	1	1
Main		0%		n/a	2	2	29	29	2	2	1	1
Shortcut		0%		0%	13	13	14	14	4	4	1	1
Measure		83%		56%	20	35	13	74	0	6	0	1
Showtheerror		0%		0%	6	6	17	17	2	2	1	1
outputFile		0%		n/a	2	2	16	16	2	2	1	1
Editmeansure.new ActionListener() {...}		0%		n/a	2	2	7	7	2	2	1	1
errorMessage		0%		0%	4	4	11	11	2	2	1	1
CustomOutputStream		0%		n/a	2	2	7	7	2	2	1	1
overridepiant		0%		n/a	3	3	8	8	3	3	1	1
window.new ActionListener() {...}		0%		n/a	2	2	5	5	2	2	1	1
Editmeansure.new Runnable() {...}		0%		n/a	2	2	7	7	2	2	1	1

Figure 2. The test coverage report for the entire TAB2MXL program.

TAB2MXL > tab2mxl > Measure

Sessions

Measure

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
durationCheck(double)		48%		37%	7	9	10	17	0	1
legatos(String)		86%		56%	11	16	1	26	0	1
getNoteMeasure(String, int, String)		92%		83%	1	4	1	10	0	1
getOctaveMeasure(String, int, String)		92%		83%	1	4	1	10	0	1
Measure(String)		100%		n/a	0	1	0	7	0	1
getDuration(int, int, int, int)		100%		n/a	0	1	0	4	0	1
Total	55 of 341	83%	25 of 58	56%	20	35	13	74	0	6

Created with JaCoCo 0.8.6.202009150832

Figure 3. The test coverage for the Measure class.

As shown in **Figure 2** and **Figure 3**, the testing for the Measure class is sufficient because the test cases cover 83% of the code in the Measure class. This means that most of the instructions and branches (like the code in if statements and while loops) have been tested and show that they return the correct values.

TAB2MXL > tab2mxl > StringInstrument

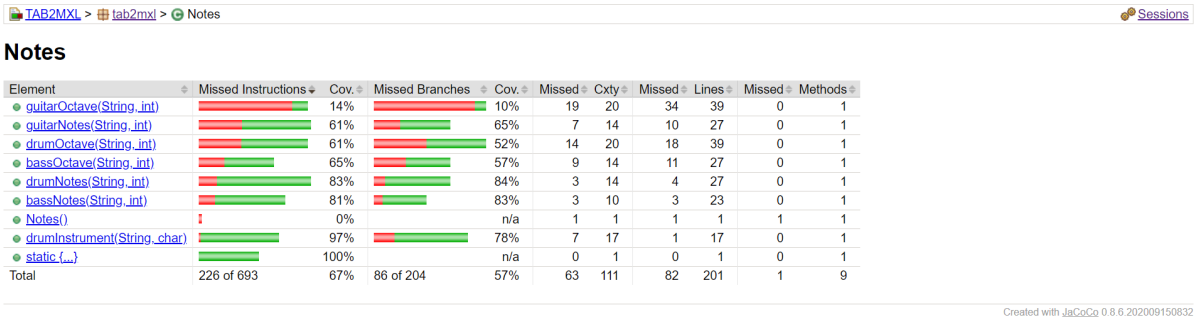
Sessions

StringInstrument

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
printToXML(String, String, String, String, String, String, String)		51%		48%	55	77	61	186	0	1
endHeading()		51%		50%	1	2	8	17	0	1
resetGlobal()		0%		n/a	1	1	8	8	1	1
getGuitar(String, String, String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
getGuitar(String, String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
getBass(String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
getBass(String, String, String, String)		0%		n/a	1	1	2	2	1	1
setStr1(String)		0%		n/a	1	1	2	2	1	1
setStr2(String)		0%		n/a	1	1	2	2	1	1
setStr3(String)		0%		n/a	1	1	2	2	1	1
setStr4(String)		0%		n/a	1	1	2	2	1	1
setStr5(String)		0%		n/a	1	1	2	2	1	1
setStr6(String)		0%		n/a	1	1	2	2	1	1
setStr7(String)		0%		n/a	1	1	2	2	1	1
setType(char)		0%		n/a	1	1	2	2	1	1
setTemp(int)		0%		n/a	1	1	2	2	1	1
getStr6()		0%		n/a	1	1	1	1	1	1
getStr7()		0%		n/a	1	1	1	1	1	1
xmlHeader(int)		100%		100%	0	7	0	125	0	1
StringInstrument()		100%		n/a	0	1	0	14	0	1
StringInstrument(String, String, String, String)		100%		n/a	0	1	0	11	0	1
StringInstrument(String, String, String, String, String, String, String)		100%		n/a	0	1	0	4	0	1
StringInstrument(String, String, String, String, String, String)		100%		n/a	0	1	0	4	0	1
StringInstrument(String, String, String, String, String)		100%		n/a	0	1	0	4	0	1

Figure 4. The test coverage for the StringInstrument class.

Based on the test coverage metrics for the StringInstrument class shown in **Figure 4**, the testing for the StringInstrument class is sufficient. This is because the test cases show that the heading and ending part of the MusicXML output is correct. Although, half of the code in printToXML method is covered by the test cases, it should be sufficient because the main instructions and branches (i.e. the if statements and while loops that are used to build the string containing the body of the MusicXML output) have been tested by the test cases and show that they return the correct values.



TAB2MXL > tab2mxl > Notes — Sessions

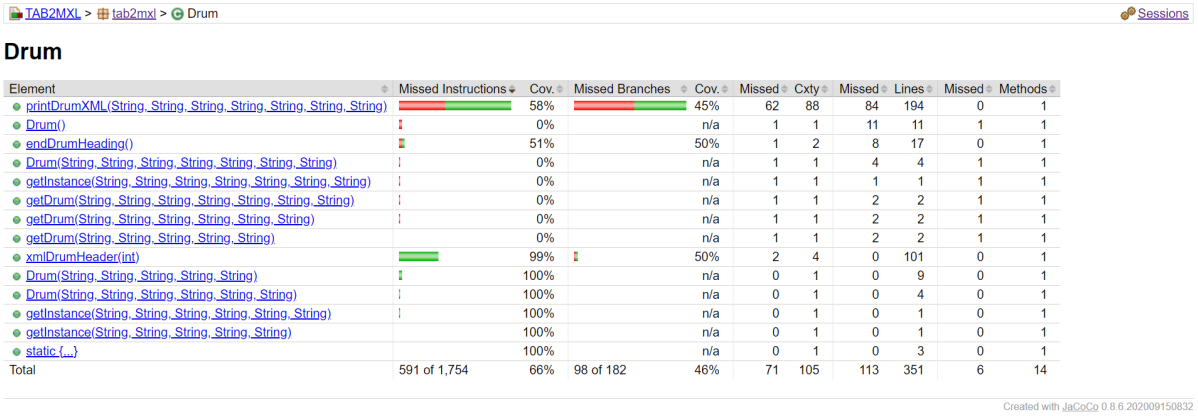
Notes

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
guitarOctave(String, int)		14%		10%	19	20	34	39	0	1
guitarNotes(String, int)		61%		65%	7	14	10	27	0	1
drumOctave(String, int)		61%		52%	14	20	18	39	0	1
bassOctave(String, int)		65%		57%	9	14	11	27	0	1
drumNotes(String, int)		83%		84%	3	14	4	27	0	1
bassNotes(String, int)		81%		83%	3	10	3	23	0	1
Notes()		0%		n/a	1	1	1	1	1	1
drumInstrument(String, char)		97%		78%	7	17	1	17	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	226 of 693	67%	86 of 204	57%	63	111	82	201	1	9

Created with JaCoCo 0.8.6.202009150832

Figure 5. Test coverage for the Notes class.

Based on the test coverage metrics for the Notes class in **Figure 5**, the test coverage for the Notes class is sufficient. Although some of the methods in Notes class have low coverage, the testing is still sufficient because the most important instructions and branches have been tested and they return the correct values. For instance, bassNotes and guitarNotes methods have been tested and they return the correct bass notes and guitar notes.



TAB2MXL > tab2mxl > Drum — Sessions

Drum

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
printDrumXML(String, String, String, String, String, String, String)		58%		45%	62	88	84	194	0	1
Drum()		0%		n/a	1	1	11	11	1	1
endDrumHeading()		51%		50%	1	2	8	17	0	1
Drum(String, String, String, String, String, String, String)		0%		n/a	1	1	4	4	1	1
getInstance(String, String, String, String, String, String, String)		0%		n/a	1	1	1	1	1	1
getDrum(String, String, String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
getDrum(String, String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
getDrum(String, String, String, String, String)		0%		n/a	1	1	2	2	1	1
xmlDrumHeader(int)		99%		50%	2	4	0	101	0	1
Drum(String, String, String, String, String)		100%		n/a	0	1	0	9	0	1
Drum(String, String, String, String, String, String)		100%		n/a	0	1	0	4	0	1
getInstance(String, String, String, String, String, String)		100%		n/a	0	1	0	1	0	1
getInstance(String, String, String, String, String)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	3	0	1
Total	591 of 1,754	66%	98 of 182	46%	71	105	113	351	6	14

Created with JaCoCo 0.8.6.202009150832

Figure 6. The test coverage for the Drum class.

The test coverage for Drum class is sufficient because the tests cover the most important aspects of the Drum class. The tests cover the printDrumXML method (which builds the body of the MusicXML output for the drum tablature), the xmlDrumHeader method (which builds the header part of the MusicXML output), and endDrumHeader method (which builds the ending part of the MusicXML output). Although about 50% of the printDrumXML and endDrumHeading methods have

been covered, the testing is still sufficient because the most important instructions and branches have been tested for.

4.4 User Acceptance Testing

Here, the purpose of the acceptance test is to confirm that the system is ready for operational use. During the acceptance testing, end-users (customers) of the system compare the system to its initial requirements.

4.4.1 Scenario 1: Create a MusicXML file from Guitar Tablature

Scenario Description

Someone wishes to run the Music XML Converter to convert a guitar tablature into a MusicXML file.

The following scripts will cover this scenario:

1. Access the Music XML Converter
2. Upload a guitar tab
3. Translate the tab using the converter
4. Save the converted file as an xml file.

Script Steps

Step #	Test Action	Expected Results	Pass/Fail
1	Access the Music XML Converter	The Music XML Converter window successfully opened.	
2	Click *Open*	The File Explorer window is opened. Can choose a file and click Open to load it into the Converter window.	
3	Upload a guitar tablature	Guitar tablature is successfully loaded into the Converter window.	
4	Press *Translate*	The guitar tablature is successfully translated, and is shown in the window.	
5	Click *Save As*, and save the converted file as an xml file	The converted file is successfully saved on the user's system.	

Why this should be sufficient testing for this scenario:

The above steps are sufficient because it involves/uses all the important features of the GUI. For instance, the *Convert*, *translate*, and *Save As* functions have been used and tested. If these features work (and all the steps above have passed), the system proves to the user its compatibility with converting guitar .txt tabalatures to an .xml file.

4.4.2 Scenario 2: Create a MusicXML File From a Bass Guitar Tablature

Scenario Description

User trying to convert a bass guitar tablature from .txt file to an .xml file.

The following scripts will cover this scenario:

1. Access the Music XML Converter
2. Upload a bass guitar tablature
3. Translate the tab using the converter
4. Save the converted file as an xml file.

Script Steps

Step #	Test Action	Expected Results	Pass/Fail
1	Access the Music XML Converter	The Music XML Converter window successfully opened.	
2	Click *Open*	The File Explorer window is opened. Can choose a file and click Open to load it into the Converter window.	
3	Upload a bass tablature	Bass tablature is successfully loaded into the Converter window.	
4	Press *Translate*	The bass tablature is successfully translated, and displayed on the window	
5	Click *Save As*, and save the converted file as an xml file	The converted file is successfully saved on the user's system.	

Why this should be sufficient testing for this scenario:

The above steps are sufficient because it involves/uses all the important features of the GUI. For instance, the *Convert*, *translate*, and *Save As* functions have been used and tested. If these features

work (and all the steps above have passed), the system proves to the user that its compatible with converting bass .txt tablatures to an .xml file.

4.4.3 Scenario 2: Use .xml Tablature on Music Application

Scenario Description

The user imports the resulting .xml file into any music tablature, for this scenario MuseScore will be used, however any music application should work successfully. Refer to the link below if you would like to install MuseScore:
https://musescore.org/en

The following scripts will cover this scenario:

1. Upload the tablature to the MusicXML Convertor.
2. Click on *Translate*
3. Save .xml file on the computer
4. Open the .xml file with MuseScore 3
5. Play tablature

Script Steps

Step #	Test Action	Expected Results	Pass/Fail
1	Upload the tablature to the MusicXML Convertor	Tablature is successfully loaded into the Converter window.	
2	Click on *Translate*	The tablature is successfully translated, and displayed on the window.	
3	Save .xml file on the computer	Tablature is saved on the computer as an .xml file.	
4	Right click on .xml file → *open with→ MuseScore 3*	The tablature will be displayed on the window.	
5	Play tablature by clicking on ▸ on the top panel	Sound of the tablature will be played.	

Why this should be sufficient testing for this scenario:

The above steps are sufficient because it checks if the exported .xml file is acceptable and can be supported by many music applications, like MuseScore. Once all the steps have passed successfully, the program is proved to be a reliable software to be used to convert .txt tablature files into .xml tablature files.

5.0 Environment Requirements

Eclipse should be installed and updated to the latest version, in order to use the provided program. Refer to the link below to install Eclipse on your system:
https://www.eclipse.org/downloads/

6.0 Control Procedures

6.1 Problem Reporting

Users should take notes of any problems encountered during the execution of the test cases or anytime while running the system and report them via the following email: xmlconvertor@gmail.com

6.2 Change Requests

Since the Converter is still in its prototype phase, there will be updates that will enhance the application. The main changes will affect:

- ➔ The ability for the Converter to translate drum tablatures
  - ◆ Expect the system to be able to translate drum tablatures into MusicXML format.
  - ◆ Expect the system to be able to save this file into the user's system.
- ➔ The ability for the Converter to detect chords from the tablature

If you request a change, fill out the Change Requests form:
https://docs.google.com/document/d/1n32r0o9hdt-Yc2tBxx8HHvusaq5RH6ANOthyb05ZSQ8/edit?usp=sharing

7.0 Features To Be Tested

The following features of the Music XML Converter that are to be tested are:

- ❏ The user interface of the Converter (i.e. the GUI)
  - ❏ The ability to select and load a text file into the window
  - ❏ The ability to translate text tablature into MusicXML format and show the result in to window
    - ❏ The only types of music tablatures to be tested are bass and guitar.
  - ❏ The ability to to save the translation

8.0 Features Not to Be Tested

The following features of the Music XML Converter that are not to be tested are:

- ❏ The ability to translate drum text tablature into MusicXML format

9.0 Resources

Name	Responsibility
Ali Sheikhi	Backend developer and project manager
Maisha Rahman	Backend developer, editor and quality assurance
Nabaa Gaziy	Backend developer and editor
Wenxuan “Shawn” Li	Frontend developer
Lars Jaylen Palalon	Full-stack developer

10.0 Dependency

Significant constraints on testing:

- MusicXml Convertor only accepts bass and guitar tablatures.
- The program only accepts tablatures in .txt file format.
- The program is not able to detect chords yet.

11.0 Approval

Person(s) who must approve this plan:

1. Manager: Professor Vassilios Tzerpos
2. Customer

12.0 References

[1] “Sample Test Plan Document (Test Plan Example with Details of Each Field),” *Software Testing Help*, 18-Feb-2021. [Online]. Available: https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/. [Accessed: 28-Feb-2021].

[2] O. T. L. OpenThink Labs Follow, “Software Development : Change Request Template,” *SlideShare*, 24-Feb-2015. [Online]. Available: https://www.slideshare.net/openthinklabs/software-development-change-request-template. [Accessed: 28-Feb-2021].

[3] “Best test plan templates and examples: manual and automation,” *EasyQA*, 03-Jul-2019. [Online]. Available: https://geteasyqa.com/qa/best-test-plan-template/. [Accessed: 28-Feb-2021].