Da Capo

Testing Document

Version 1.1

Date: 04/13/2021

Gong-Fan (Billy) Bao - 213563150
Syed Quadri - 216204554
Elijah Nnorom - 214502504
Miguel Mesa Gonzalez - 216138323
Ayub Osman Ali - 217612847

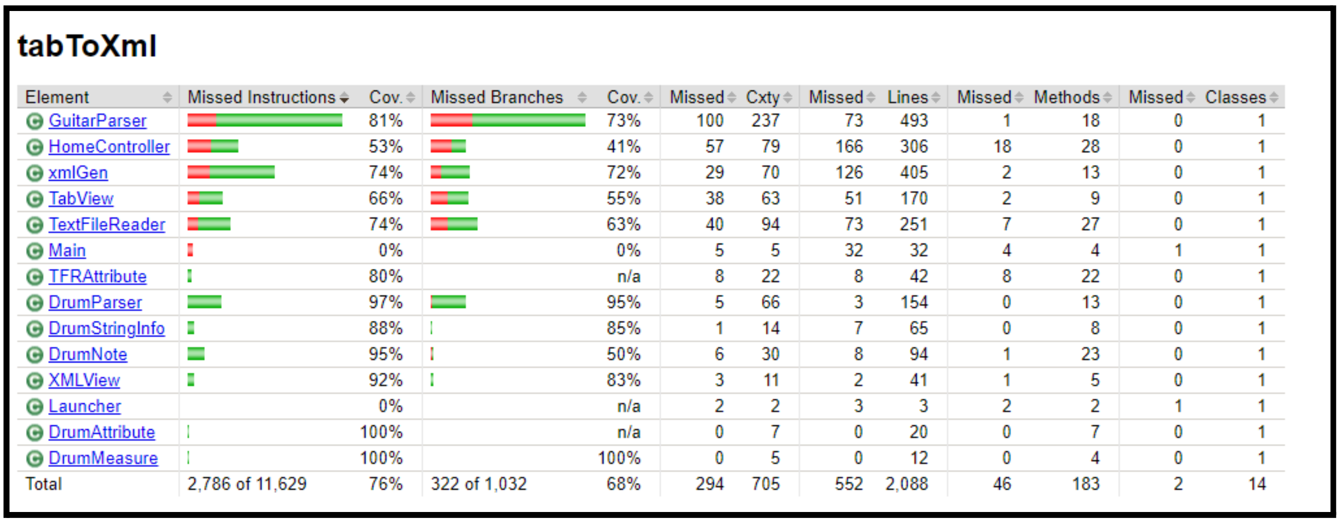
Table of Contents

1. Introduction

This testing document serves to provide the reader a detailed look at the various classes in the De Capo Tab to MusicXML Generator Application. The sections of this document can be broken down into six main sections, each of which cover different Sections of the information processed internally. This testing document also contains details on existing unit tests done by JUnit Testing as well Jacoco test coverage.

2. Quantitative Overview

The following is coverage of the tests conducted on the existing software.



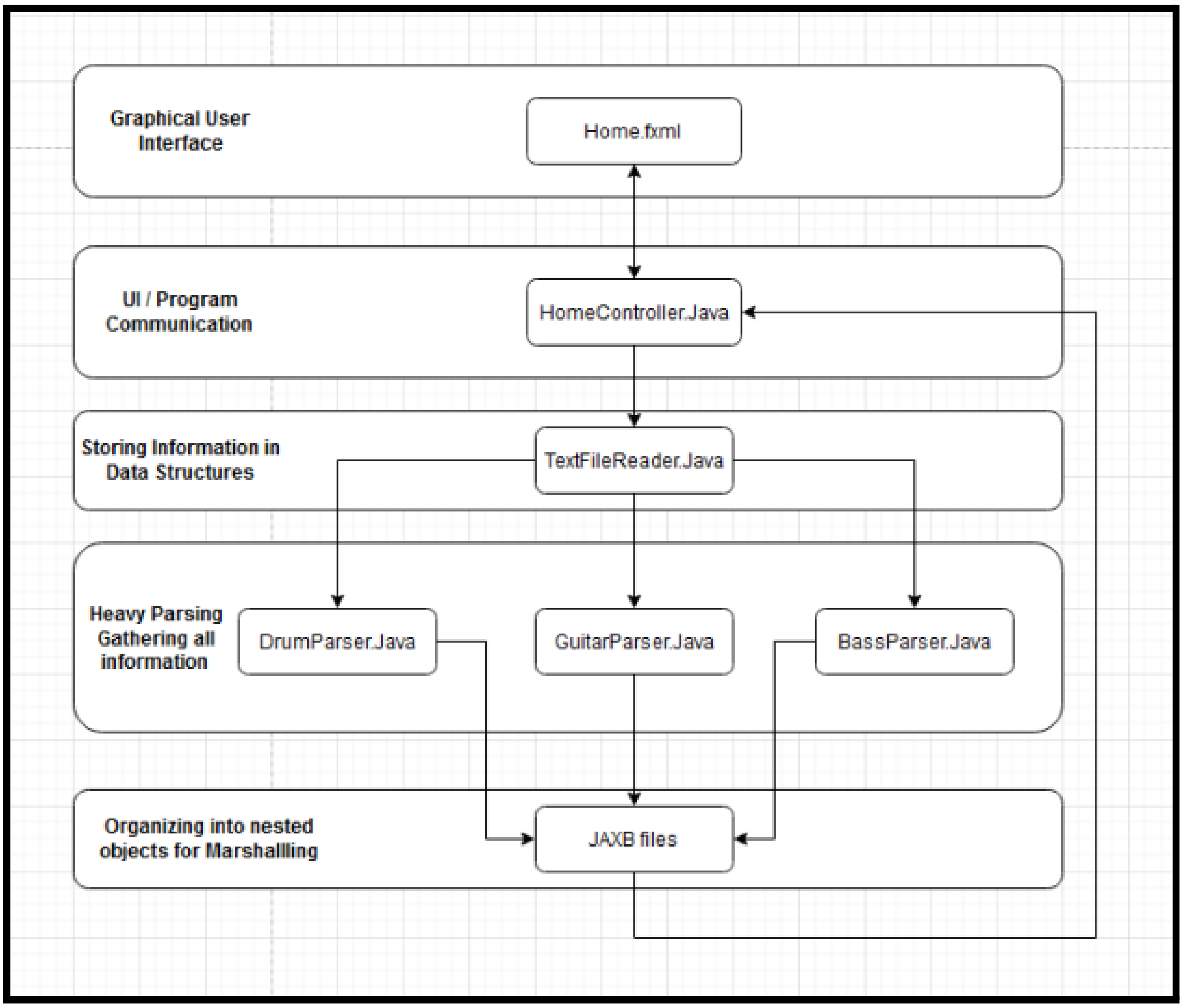
tabToXml

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GuitarParser		81%		73%	100	237	73	493	1	18	0	1
HomeController		53%		41%	57	79	166	306	18	28	0	1
xmlGen		74%		72%	29	70	126	405	2	13	0	1
TabView		66%		55%	38	63	51	170	2	9	0	1
TextFileReader		74%		63%	40	94	73	251	7	27	0	1
Main		0%		0%	5	5	32	32	4	4	1	1
TFRAttribute		80%		n/a	8	22	8	42	8	22	0	1
DrumParser		97%		95%	5	66	3	154	0	13	0	1
DrumStringInfo		88%		85%	1	14	7	65	0	8	0	1
DrumNote		95%		50%	6	30	8	94	1	23	0	1
XMLView		92%		83%	3	11	2	41	1	5	0	1
Launcher		0%		n/a	2	2	3	3	2	2	1	1
DrumAttribute		100%		n/a	0	7	0	20	0	7	0	1
DrumMeasure		100%		100%	0	5	0	12	0	4	0	1
Total	2,786 of 11,629	76%	322 of 1,032	68%	294	705	552	2,088	46	183	2	14

The report above generated by Jacoco runs a comprehensive search into the lines that have been covered by the JUnit test cases created to test the correctness of the execution of all the various areas of importance in the software. The tests above cover a vast majority of the testables areas of the software. Smaller supporting classes such as DrumMeasure, DrumAttribute, XMLView, DrumNote, DrumStringInfo, DrumParser, and TFRAttribute have a near 100% coverage while the larger classes with several supporting methods for extra functionality such as the GuitarParser, HomeController and xmlGen that contain certain constructors and methods which only execute at runtime.

Overall Sufficiency: The Test cases for Da Capo amount to a total of 80 test cases. They range from the larger overall test of functionality for each class as well as a comprehensive set of tests on both the guitar and drums parsers. The testing on the GUI has also covered all major actions the user may take. The only extensions left to make in terms of testing would be to test for all cases of error handling by passing incorrect procedures and information to the software in different areas.

3. Conducted Testing


Graphical User Interface
Home.fxml
UI / Program Communication
HomeController.Java
Storing Information in Data Structures
TextFileReader.Java
Heavy Parsing Gathering all information
DrumParser.Java
GuitarParser.Java
BassParser.Java
Organizing into nested objects for Marshallling
JAXB files

The First section of testing testing the TextFileReader.Java Class which contains all the initial gathering of information.

The Second section focuses on the GuitarParser Class which is currently the most developed class of the three instruments. As such, the testing is quite detailed and thorough.

The Third section focuses on the DrumParser Class. It makes use of the large constructor to complete the parsing process and checks out the various outputs.

The Fourth Section focuses on the XML generating class. It passes input from the parsers and checks to see if the out contains correct tags and objects.

The Fifth Section focuses on the GUI and HomeController. It contains methods that will function as the main line of connection between all the activities on the front end as well as the activities on the back end.

3.1. TextFileReader Testing

3.1.1 Testing Approach

This section provides an overview of the test cases for the Text File Reader. The File Reader is the first part of the software and all other parts of the software depend on it, if the File Reader crashes or provides an unexpected result the software as a whole will be affected. This section provides the test cases that have been written to thoroughly test the Text File Reader, an explanation of what each test case test has also been provided under the name of each test case.

3.1.2 TextFileReader Tests

Test ID	**001-001**
Test Name	test_TextFileReaderString()
Description	Checks to see whether the object for the TextFileReader.java is created successfully without crashing. This is important as it ensures that provided that a string representing the name/path of the file which contains a Guitar tablature, the file reader creates the object without crashing.
Sufficiency	Sufficient: This is sufficient because this test can always be modified to work with the path to a file that contains a different guitar tablature. Creating the object of the TextFileReader is a very important step during translation, this test is very important.
Additional Comments	This test is designed for the guitar tablature, a different test has been designed for the drum tablature.

Test ID	**001-002**
Test Name	test_TextFileReaderFile()
Description	Checks to see whether the object for the TextFileReader.java is created successfully without crashing. This is important as it ensures that provided that a string representing the name/path of the file which contains a drum tablature, the file reader creates the object without crashing.

Sufficiency	Sufficient: This is sufficient because this test can always be modified to work with the path to a file that contains a different drum tablature. Creating the object of the TextFileReader is a very important step during translation, this test is very important.
Additional Comments	This test is designed for the drum tablature, a different test has been designed for the guitar tablature.

Test ID	**001-003**
Test Name	test_CreateparsedTab()
Description	Checks to see if the tablature that has been provided to the TextFileReader is property saved as an arraylist and prepared for future stages of the translation process.
Sufficiency	Sufficient: This test is sufficient because having the tablature converted to the arraylist is crucial to the next stages in the translation process. This test adequately tests that the right arraylist is produced.
Additional Comments	This test was done on a file containing a guitar tablature, that was adequate to ensure the intended output was produced. It can be modified to work on a file containing a drum tablature as well.

Test ID	**001-004**
Test Name	testReader2()
Description	Checks to see whether the automatic instrument detection feature in the text file reader works for the drum.
Sufficiency	Sufficient: This test is sufficient enough to test the automatic instrument detection on a drum tablature. Further testing is required to ensure instrument detecting works for the drum.
Additional Comments	This test was designed to work on the drum tablature, another test will be designed to work with the guitar.

Test ID	**001-005**
Test Name	testReader3()
Description	Checks to see whether the automatic instrument detection feature in the text file reader works for the guitar.

Sufficiency	Sufficient: This test is sufficient enough to test the automatic instrument detection on a guitar tablature. Further testing is required to ensure instrument detecting works for the guitar.
Additional Comments	This test was designed to work on the guitar tablature, another test will be designed to work with the drum.

Test ID	001-006
Test Name	test_TuningLetterDetection()
Description	Check to confirm the tuning for a guitar tablature is correct.
Sufficiency	Sufficient: This test is simple but it serves its purpose adequately.
Additional Comments	This test was designed to work on the guitar tablature. It could be further expanded to work for drum tablatures but that has not been tested in this current System.

3.2. Guitar Parser Testing

3.2.1 Testing Approach

The following test cases are for confirming that the Guitar Parser class performs as intended. They cover the generation of the Duration, Type, Note, Fret Number, Fret String, Chord, Hammer-on and Pull-offs, Bend and Release, Slides, and Grace arrays. Currently, any note durations that are larger than the next largest note value (whole, dotted half, half, dotted quarter, quarter, dotted eighth, eighth) will be defaulted to that note duration. Any note with duration less than an eighth will be considered a 16th note.

Overall Sufficiency: Moderately Sufficient, covers the majority of the cases where the tablature data is already present. However, these cases do not cover when the user adjusts the meta-data per measure.

3.2.2 Duration Tests

Test ID	002-001
Test Name	testDuration1()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists entirely of non-chord eighth notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-002
Test Name	testDuration2()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists entirely of non-chord quarter notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-003
Test Name	testDuration3()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists entirely of non-chord half notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-004
Test Name	testDuration4()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists entirely of a non-chord whole note, with default number of dashes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-005
Test Name	testDuration5()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists of a mixture of half, quarter, eighth, and 16th non-chord whole notes.

Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-006**
Test Name	testDuration6()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists of chordal whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-007**
Test Name	testDuration7()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists of a mixture of chordal half, quarter, eighth, and 16th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-008**
Test Name	testDuration8()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists of a mixture of irregular/dotted notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose.

3.2.2 Type Tests

Test ID	**002-009**
Test Name	testType1()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists entirely of non-chord eighth notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType2()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists entirely of non-chord quarter notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType3()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists entirely of non-chord half notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType4()

Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists entirely of a non-chord whole note.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType5()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists of a mixture of half, quarter, eighth, and 16th non-chord whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType6()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists of chordal whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	**002-009**
Test Name	testType7()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists of a mixture of chordal half, quarter, eighth, and 16th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-009
Test Name	testType8()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Type array values, when given a Guitar tab that is a measure long and consists of a mixture of irregular/dotted notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	002-009
Test Name	testWholeNote()
Description	Checks to see whether the parseToRhythm method in GuitarParser.java returns the correct Duration array values, when given a Guitar tab that is a measure long and consists of a whole note, but with less dashes than expected.
Sufficiency	Not Sufficient: This test only covers one case of irregular number of dashes.

3.2.3 Notes Tests

Test ID	002-009
Test Name	testNotes1()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct values, when given a Guitar tab that is a measure long and consists entirely of non-chord 8th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

Test ID	002-009
Test Name	testNotes2()

Description	Checks to see whether the traslateParsed method in GuitarParser.java returns the correct values, when given a Guitar tab that is a measure long and consists entirely of chordal whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

3.2.4 Fret Number Tests

Test ID	**002-009**
Test Name	testFretNumber1()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct values, when given a Guitar tab that is a measure long and consists entirely of non-chord 8th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

3.2.5 Fret String Tests

Test ID	
Test Name	testFrestString1()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct FretString array values, when given a Guitar tab that is a measure long and consists entirely of non-chord 8th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional	

Comments	

Test ID	
Test Name	testFrestString2()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct FretString array values, when given a Guitar tab that is a measure long and consists entirely of a chord of whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

3.2.6 Chord Tests

Test ID	
Test Name	testChord1()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct Chord array values, when given a Guitar tab that is a measure long and consists entirely of non-chord 8th notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

Test Name	testChord2()
Description	Checks to see whether the translateParsed method in GuitarParser.java returns the correct Chord array values, when given a Guitar tab that is a measure long and consists entirely of a chord of whole notes.
Sufficiency	Highly Sufficient: This test is sufficient for its purpose, as future tests cover other scenarios.
Additional Comments	

3.2.7 Hammer-on and Pull-off Tests

Test ID	
Test Name	testHammeron1()
Description	Tests that data needed by the application to identify and process hammer-ons in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers hammer-on between two frets of values between 0 and 12. In regards to hammer-on this test is sufficient.
Additional Comments	Sequential hammer-on and pull-off will be tested in another test case that has been designed for this purpose.

Test ID	
Test Name	testPulloff1()
Description	Tests that data needed by the application to identify and process pull-offs in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers pull-offs between two frets of values between 0 and 12. In regards to pull-offs this test is sufficient.
Additional Comments	Sequential hammer-on and pull-off will be tested in another test case that has been designed for this purpose.

Test ID	
Test Name	testHammeronAndPulloff()
Description	Tests that data needed by the application to identify and process sequential hammer-ons and pull-offs in the provided tablature has been extracted and stored.
Sufficiency	Sufficient: This test covers sequential hammer-on and pull-off operations involving frets of values between 0 and 12. This test builds on other tests designed to check hammer-on and pull-off individually.
Additional Comments	This test ensures that hammer-ons and pull-offs are tested comprehensively.

3.2.8 Bend and Release Tests

Test ID	
Test Name	testBend1()
Description	This tests ensures that data needed by the application to identify and process bend in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers bend involving fret numbers of values between 0 and 9. Expansion of this test to cover fret values between 10 and 12 will make this test more sufficient.
Additional Comments	Bend always provides an output for every legal fret value that is accepted by the application, however, values between 10 and 12 are inconsistent or could be incorrect. The expected result is provided for fret values between 0 and 9. This inconsistency was identified from testing with this test case.

Test ID	
Test Name	testRelease1()
Description	This tests ensures that data needed by the application to identify and process release in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers release involving fret numbers of values between 0 and 9. Expansion of this test to cover fret values between 10 and 12 will make this test more sufficient.
Additional Comments	Release always provides an output for every legal fret value that is accepted by the application, however, values between 10 and 12 are inconsistent or could be incorrect. The expected result is provided for fret values between 0 and 9. This inconsistency was identified from testing with this test case.

3.2.9 Slides Tests

Test ID	
Test Name	testAscedingSlide()
Description	This tests ensures that data needed by the application to identify and process ascending slides in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers ascending slides involving fret numbers of

	values between 0 and 12. This test is sufficient for its purpose, further testing is needed for sequential slides.
Additional Comments	Sequential ascending and descending slides will be tested in another test case that has been designed for this purpose.

Test ID	
Test Name	testDescendingSlide()
Description	This tests ensures that data needed by the application to identify and process descending slides in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers descending slides involving fret numbers of values between 0 and 12. This test is sufficient for its purpose, further testing is needed for sequential slides.
Additional Comments	Sequential ascending and descending slides will be tested in another test case that has been designed for this purpose.

Test ID	
Test Name	testSequentialDandASlide()
Description	This tests ensures that data needed by the application to identify and process sequential ascending and descending slides in the provided tablature has been extracted and stored.
Sufficiency	Sufficient: This test covers sequential ascending and descending slides involving frets of values between 0 and 12. This is an expansion of a previous tests that checked either ascending or descending slide.
Additional Comments	This test ensures that ascending and descending slides have been tested comprehensively.

Test ID	
Test Name	testLegatoSlide()
Description	This tests ensures that data needed by the application to identify and process legato slides in the provided tablature has been extracted and stored.
Sufficiency	Sufficient: This test covers legato slides involving frets of value between 0 and 12. This test is self sufficient as it tests all scenarios involving legato slides in a guitar tablature.
Additional	This test ensures legato slides have been tested comprehensively.

Comments	

3.2.10 Grace Tests

Test ID	
Test Name	testGrace1()
Description	This tests ensures that data needed by the application to identify and process grace tags in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers grace tag generation for a simple regular tablature, involving only eighth notes and a chord, with no grace indicators. This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	
Test Name	testGrace2()
Description	This tests ensures that data needed by the application to identify and process grace tags in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers cases where grace indicators are present, but should not generate grace tags in the musicXML. This test is sufficient for its purpose, as future tests cover other scenarios.

Test ID	
Test Name	testGrace3()
Description	This tests ensures that data needed by the application to identify and process grace tags in the provided tablature has been extracted and stored.
Sufficiency	Moderately Sufficient: This test covers cases where grace indicators are present and should generate grace tags, including single and sequential grace notes. This test is sufficient for its purpose.

Test ID	
Test Name	testGraceAndDuration()

Description	This tests ensures that data needed by the application to identify and process duration values in the provided tablature has been extracted and stored, while grace tags are present.
Sufficiency	Moderately Sufficient: This test covers cases where grace indicators are present and should generate grace tags, and checks if they affect the Duration array. This test is sufficient for its purpose.

Test ID	
Test Name	testGraceAndType()
Description	This tests ensures that data needed by the application to identify and process type values in the provided tablature has been extracted and stored, while grace tags are present.
Sufficiency	Moderately Sufficient: This test covers cases where grace indicators are present and should generate grace tags, and checks if they affect the Type array. This test is sufficient for its purpose.

3.3. Drum Parser Testing

3.3.1 Testing Approach

The following test cases are for confirming that the Drum Parser class performs as intended. Although there are not many, they cover the complete process of parsing from start to end with passing of the tablature as Strings, the correct preprocessing of information and a minor check on the final output of Measure containing Notes that will be used to generate the output XML.

3.3.2 Drum Parser Tests

Test ID	
Test Name	test_NumberOfMeasures()
Description	This test checks whether the parser is able to recognize the number of measures correctly for certain input provided
Sufficiency	Sufficient. The test is able to recognize both single and double barline interruptions in the tablature to detect the correct number
Additional Comments	Two instances of this test are provided. Both tests were successful

Test ID	
Test Name	test_NumberOfStrings()
Description	Checks to see if the drum parser object that is being passed a tablature as a group of strings is parsing and storing the information correctly in an arraylist of measures that hold an arraylist of several notes. In this case, the length of the arraylist containing measures was used as an indicator of this property. This will be improved and broken down into sections in the future.
Sufficiency	Sufficient: Tablatures of varying numbers of lines are given. Output remains correct across all other tests of similar nature
Additional Comments	This particular test passes a Tab of 4 strings

Test ID	
Test Name	test_NumberOfStrings2()
Description	Tests whether the parser is able to recognize the number of lines passed to it correctly.
Sufficiency	Sufficient: Tablatures of varying numbers of lines are given. Output remains correct across all other tests of similar nature
Additional Comments	This particular test passes a Tab of 2 strings

Test ID	
Test Name	test_NumberOfStrings6()
Description	Tests whether the parser is able to recognize the number of lines passed to it correctly.
Sufficiency	Sufficient: Tablatures of varying numbers of lines are given. Output remains correct across all other tests of similar nature
Additional Comments	This particular test passes a Tab of 6 strings

Test ID	
Test Name	test_NumberOfStrings8()
Description	Tests whether the parser is able to recognize the number of lines passed to it correctly.
Sufficiency	Sufficient: Tablatures of varying numbers of lines are given. Output remains correct across all other tests of similar nature
Additional Comments	This particular test passes a Tab of 8 strings

Test ID	
Test Name	test _TransposeMethod()
Description	Checks to see if the drum parser object that is created stores the information passed as an arraylist of strings is correctly transposed before beginning the parsing process. This test ensures that the pre processing is done correctly which makes the other tests and processes more reliable. This will be updated in the future with several variations of tablature passed to it.

Sufficiency	Sufficient: The entirety of the method was covered in the test. Output remains consistent across other tests of varying input.
Additional Comments	Testers may be inclined to include error handling in the event that the software ends up passing on erratic input to transpose

Test ID	
Test Name	test_TextFileReaderObject
Description	This test ensures that the DrumParser object is correctly storing an attribute of a TextFileReader upon instantiation.
Sufficiency	Sufficient: The test is quite small and does not require extensive testing.
Additional Comments	The test may be further developed to ensure all necessary attributes in the textfilereader object are instantiated and filled in correctly

Test ID	
Test Name	test_getMeasuresParsed()
Description	Tests if the process of extracting measures from a text input and into separate measures and lines in correctly executed for a given drum tablature
Sufficiency	Moderately Sufficient: While multiple tabs have been given, there are still ways to corrupt input data so that the measures are not correctly passed. They lie unchecked.
Additional Comments	Future testers may want to look into expanding the sufficiency with cases and new input.

Test ID	
Test Name	test_VoiceSelectorMethod()
Description	Tests if the process of selecting the instrument voice is chosen correctly. The helper method is passed initials that correspond to certain values each relating to a voice that is returned.
Sufficiency	Sufficient: The method under testing is a short execution with several instrument and voice types.
Additional Comments	Can be expanded to include new instruments as the developers feel the need to add more instruments to the functionality

Test ID	
Test Name	test_SetDrumStringInfo_Length()
Description	Tests if the parser is correctly recognizing the amount of string instruments it needs to create for each music piece. The test is an extension of the prior tests for correct line count.
Sufficiency	Sufficient: Given differing sets, the test returns back the correct amounts
Additional Comments	Nothing of significance

Test ID	
Test Name	test_getInstrumentInitials_Values_Valid
Description	Test if the values of instruments detected by the parser are matching with a given truth. The values must match up exactly to pass.
Sufficiency	Moderately Sufficient: The method tests different tab inputs and satisfies each. It also perform correctly in the event of invalid instruments passed or compared.
Additional Comments	

3.4. XMLGen Testing

3.4.1 Testing Approach

The test cases described in this section are designed to ensure that the class that generates the xml document performs as intended. They cover distinct xml documents that are generated for the different instruments accepted by our application which are; Classical Guitar, Bass Guitar and Drum.

3.4.2 XMLGen Tests

Test ID	
Test Name	testXMLGenConstructor()
Description	This test ensures that the Constructor in the xmlGen accepts both a Guitar and Drum tab when the object is being initialized. This test ensures that the DrumParser object is correctly storing an attribute of a TextFileReader upon instantiation.
Sufficiency	Sufficient: This is a short test but very important as initialization determines the type of XML document that is generated. Overall, this test is effective and can be used whenever a correct Guitar or Drum tab is provided as an input to the xmlGen constructor.
Additional Comments	This test case can easily be utilized for the tablature of a different instrument.

Test ID	
Test Name	test_CorrectNestObjectCreationDrums()
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	test_CorrectNestObjectCreationGuitar()
Description	Create a musicxml output for a given guitar file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	testGuitarXMLGen()
Description	This test is designed to confirm the correct generation of XML tags that are specific to Guitar tablatures. This includes tags for; Hammer-on, Pull-off, Slide, Bend, Release and Harmonics. This test also checks for other tags that are common to all the XML documents produced for Guitar tablatures including; part-name, measure and chord tags.
Sufficiency	Sufficient: This test is sufficient because it tests for both common tags as well as tags that depend on user input in the guitar tablature.
Additional Comments	This test can be expanded or repurposed, the tags used as expected value should be tailored to the value from the tablature provided to it.

Test ID	

Test Name	testDrumXMLGen()
Description	This test is designed to confirm the correct generation of XML tags that are specific to Drum tablatures. This includes tags that are common to all the XML documents and tags specific to the drum tablatures.
Sufficiency	Sufficient: The test is sufficient because it tests for both common tags as well as tags that depend on user input in the drum tablature.
Additional Comments	The test may be expanded to check for more tags, it can also be tailored to test for more specific tags that depend on the input in the drum tablature.

3.5. Graphical User Interface Testing

3.5.1 Testing Approach

These tests are to ensure that the UI acts accordingly to user input. It tests for highlighted text input errors, user clicks and verifies the correct state of all elements.

3.6.2 GUI Tests

Test ID	
Test Name	containsTextArea
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	CheckIfPastedTabisValid
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	editBtnDisabled

Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	convertBtnDisabled
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	editBtn_DisabledAfter_userInput
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	

Test Name	convertBtn_DisabledAfter_userInput
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	formartBtn_enabled
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	convertBtn_DisabledAfter_userInput
Description	Create a musicxml output for a given drum file and check to see that the major high level tags are correctly being instantiated.
Sufficiency	Sufficient: This test covers the high level tags while the test below cover the low level tags, together they span a wide coverage
Additional Comments	It is possible to further test the mid level tags as well

Test ID	
Test Name	xmlSceneisLoaded

Description	Checks to see if after the user selects the convert button the music XML scene is loaded.
Sufficiency	Sufficient: this tests ensures up until the convert button is pressed then
Additional Comments	

Test ID	
Test Name	Savebtn_Text_is_saveXML_after_sceneswap
Description	Ensures that while in the xml scene the button is labeled accordingly.
Sufficiency	Sufficient: as it also makes sure that the file type saved is of .txt for tab and .musixXML for the output.
Additional Comments	

Test ID	
Test Name	Savebtn_Text_is_saveTAB_after_sceneswap
Description	Ensures that the text of the save button is appropriate to the scene.
Sufficiency	Sufficient: as it also makes sure that the file type saved is of .txt for tab and .musixXML for the output.
Additional Comments	

Test ID	
Test Name	clicking_save_changes_closes_dialog
Description	Inputs tab, complies it, selects edit and then views and changes a range of measures, after which it hits saves changes. It makes sures that the dialog does close.

Sufficiency	The sufficiency of this test is that it covers a high percentage of features, and confirms up to the changes closing every UI element works as expected.
Additional Comments	