McGill

ECSE 321
Introduction to Software Engineering

Home Audio System
Deliverable 3
March 25, 2016

Kristina Pearkes 260 520 832
Aidan Piwowar 260 625 505
Kareem Halabi 260 616 162
Aurelie Pluche 260 622 575
Alexander Orzechowski 260 610 696

1. Unit Test Plan

Unit testing is used to ensure the functionality of methods and individual components within a program. The classes within the application which will be tested are the HAS Controller and Persistence classes. These class were chosen to be tested as they manipulate or store data of the system. Other classes within our application do not need to be tested as they can only be tested through other types of testing (integration and system) or are components of the model, which do not need to be tested as they were generated by Umple and do not carry out the manipulation of data. Dynamic verification and validation techniques will be used for testing.

All the methods within the controller will be tested (all listed below):

- Create Artist
- Create Album
- Add Song to Album
- Create Playlist
- Add Song to Playlist
- Create Room
- Create Room Group
- Add Room to Room Group
- Set Volume
- Set Mute
- Sort

The Persistence class consists of one method which stores the data in an xml file. This method is thoroughly tested for the inclusion of all data presented and when data has been modified.

Partition testing will be used in order to validate the execution of each of the methods. These methods will be tested by inputting a set of correct inputs as a first test then introducing one incorrect input (such as a missing name, missing album, etc) in subsequent tests. Only one incorrect input is introduced at a time in order to more easily identify the source of a problem if one arises. The quantification of the testing goal is to test 100% of these branches of execution as they should all be explored within unit tests. Therefore, each execution branch of these pivotal methods will be tested. This is to insure that the each of these essential methods, function as they should when certain errors are introduced.

Each test was written using a white-box approach and are executed whenever the controller has been modified or methods have been added (in which case new tests would be written). Furthermore, these tests will automatically run through the build. The white-box approach provides us with the information needed in order to reach the quantification goal. The different paths of execution are derived from the different loops within the method itself and thus the white-box approach is needed.

Each of these branches of execution will be tested using JUnit tests for both the desktop app and mobile app as they share the same controller. The difference when it comes to unit testing of the web application is that the PHPUnit tool is used. Because all of the features of the desktop app carry over to the web app, the same units exist and they will be tested in the same way.

2. Integration Strategy

Integration testing is to ensure the functionality between components. For the integration strategy, we will be using the Bottom-Up strategy.
The following is a list of additional tests that will be done in order.

- Persistence Layer tests: ensuring the data loads and saves to xml file
- Association Class tests:
    - Song must have an Album
    - Album must have an Artist
    - RoomGroup must have a Room
    - Playlist must have a Song
    - Playlist, Song and Album are Playable
- Play Class Functionality
    - Playing to a Room
        - Playing a Song
        - Playing an Album
        - Playing a Playlist
    - Playing to a RoomGroup
        - Playing a Song
        - Playing an Album
        - Playing a Playlist

Clarification: for the association tests, a simple JUnit/PHPUnit test using getters and setters (song.getAlbum(); album.getArtist(); playlist.hasSong(); etc) to verify the proper associations necessary will be used.

We have chosen the Bottom-Up strategy because it is efficient and less time consuming than other strategies. The bottom-up strategy utilizes many drivers with a few stubs. Most of the stub testing will be done in the Unit Tests, and using higher tier stubs as drivers for the integration testing. As an example, making sure that a song has an album, unit test for validating the album exists with correct properties and song exists with correct properties. Then, integration test if the album contains the song (album.hasSong();).

There aren't too many differences in testing between platforms. They will all be white box tests and will verify the same functionality of components listed above. The integration testing of the android app will mostly be done in the desktop testing (transferred through the export of the .jar file), then user testing to assure functionality between navigation of the view. The desktop app will contain testing with the view class as the web and mobile apps will be tested via user testing. For tool support, the desktop and android use JUnit and the web uses PHPUnit tools for testing.

The tests will cover approximately 70% of the executions paths. For example, we will be using album.getSongs();. It will return a list of songs in the album. Therefore we do not need to test album.hasSongs(); as this will of course will be true if album.getSongs() returns a list that contains 1 or more songs. Using album.getSongs() will cover most test associations between the Song and Album as we can also verify that a song was exists properly.

3. System Test Plan

After the application passes integration testing, we will focus on using a black-box approach to test the system. This will involve opening up the application and going through the different use-cases to see if we can break our app or not.

In order to test the system, we will use the following test situations:

Main Scenario:

1. The user enters music in their library successfully and then plays it in a given room or in multiple rooms.

Alternative Scenarios:

2. The user tries to play music from an empty library.
3. The user selects a room to play music in without first selecting music to play
4. The user leaves room name blank when creating a room.
5. The user leaves a music attribute (such as genre or name) blank when creating new music.
6. The user leaves the song field empty and then presses the create playlist button.
7. The user plays music without choosing a room to play it in.

Rationale

The reason why we choose the above test situations is because these are most of the paths that users can take through our application. Because our use cases describe expected system behaviour, they are good features to test the system for.

The next phase of testing would be Release Testing, where checks would be done to make sure the application meets the software requirements. Before starting Release Testing we want to make sure that no user behaviour will break the application. Thus, it makes sense to look to use cases as a guide to test the system so that we can be confident that it is of high quality and ready for the next phase of testing.

Our system testing goal is to use equivalence partitioning to achieve about 75% test coverage. This goal means that we have tested most paths a user can take through the application. Because there are only a few things you can do with HAS, we think we can cover most user behaviour throughout the system testing process.

Test Case for Situation 1

Title: Test playing different songs to different rooms and then updating one of the currently playing songs with a different one.

1. In your file directory, go to C:\XAMPP\htdocs and copy the eclipse project there.
2. Open XAMPP and start the apache server.
3. Navigate to the page localhost/HAS in your browser.
   -A Home page should appear. The page should show that no music is in the library
4. Click the “Add Music” button.
   -A new page titled Add Music should appear.

5. In the Add Artist Field, type “Adele”
6. Click the “Add Artist” button.
   -The page should refresh and “Adele” should appear in the drop down menu next to Select Artist
7. In the add Album field, type “21”
8. Click the “Add Album” button.
   -The page should refresh and “21” should be listed in the drop down menu.
9. In the add Song field, type “Rolling in the Deep”
10. In the album field, click on “21”
11. Click the “Add Song” button.
    -The page should refresh and “Rolling in the Deep” should appear in the select song drop down menu.
12. Repeat steps 9, 10, and 11 but enter the following song names: “Rumor Has It”, “Hello”, “Someone Like You”, “Set Fire to the Rain”, and “When We Were Young”
13. Navigate to the home page by pressing the “Home” button
    -You should now see all of the songs you just added.
14. Click the “Select Room(s)” button
15. Type “Family Room” in the Name of Room field.
16. Type “50” in the Volume field box
17. Click the “Add Room” button, then repeat step 15 and 16 but instead type “Basement” as the room name.
    -Both rooms should appear in the select room drop down menu.
18. Click the “Home button”
    -The Home page should appear, still with all 5 songs listed under My Music.
19. Click on “Set Fire to the Rain” then click the “Play” button.
20. You should be asked where you want to play that song. Click “Family Room” and then click “Play”.
21. Now click on “Home” and select “Rumor Has It.” You should again be asked where you want to play the song. Choose “Family Room” and click “Play”
22. Repeat step 21 but select “Someone Like You” as the song and “Basement” as the room.
23. Press the “Home” button. Under where it says Now Playing you should see “Rumor Has It” with “Family Room” as its location followed by “Someone Like You” with Basement as its location. Additionally, under where it says My Music you should see all 5 songs listed. Test Passed!

Test Case for Situation 2

Title: User tries to play music from an empty library.

1. In your file directory, go to C:\XAMPP\htdocs and copy the eclipse project there.
2. Open XAMPP and start the apache server.
3. Navigate to the page localhost/HAS in your browser. -A Home page should appear. The page should show that no music is in the library
4. “Click the “Select Room(s)” button. - A new page should appear.
5. Type “Family Room” as the name of the room.
6. Type “75” as the volume.
7. Click the “Add Room” button - The page should refresh and “Family Room” should be listed under the Room-to-Add drop down menu.

8. Click the “Home” button - The home page should appear, still with an empty music library.
9. Click the “Play” button. - An error that reads “You must select music to play before you can play music” should appear and you should still be on the home page. Test Passed!

Updated Work Plan

- Deliverable 1 Completed on February 22nd (High effort)
    - Add an album (2.1)
    - Add an artist (2.3)
    - Add a song to album (2.2)
    - Associate album to artist (3.1)
    - Functional and Non-Functional requirements
    - Domain Model
    - Use Case Diagram
    - Use Cases
    - Requirements-level sequence diagram for “Add Album” use case
    - Prototype of application on all three platforms for “Add Album” use case
- Deliverable 2 Completed on March 7th (Medium effort):
    - Add a room (1.2)
    - Description of architecture of proposed solution including a block diagram
    - Description of detailed design of proposed solution including class diagram
    - Updated umple model, see below
- Deliverable 3 by March 25th (High effort):
    - Create playlist(2.4)
    - Function in multiple rooms (1.3)
    - Organize music by album or artist (3.2)
    - Choose a volume level for a room (1.4)
    - Be able to mute a room(1.5)
    - Select an item to play (1.1)
    - Description of unit testing
    - Description of component testing
    - Description of system testing
    - Description of performance/stress testing
- Deliverable 4 by April 1st (Low effort)
    - Description of release pipeline.
- Deliverable 5 by April 11/12/14 (Medium effort)
    - Presentation about project
- Deliverable 6 by April 15 (Medium-High effort)
    - Source code of full implementation on each supported platform

Updated Umple Model