Integration Strategy

Systems integration: the independently developed subsystems are put together to make up a complete system.

There are 3 different approaches that can be taken to test the following system:

- **Big Bang Approach:** all the subsystems are integrated at the same time
- **Bottom Up Approach**: testing is conducted from sub module to main module. If the main module is not developed a temporary program called "drivers" is used to simulate the main module.
- **Top Down Approach:** testing is conducted from main module to sub module. If the sub module is not developed a temporary program called "stub" is used for simulate the sub-module.

For our system, an incremental integration process where subsystems are integrated one at a time is the best approach. Therefore, we will not be using the Big Bang Approach. As well, since most of all of our systems depend directly on subsystems, we will be using the **Bottom Up Approach**. This integration strategy is the best suit for our system because it is especially useful if major flaws occur toward the bottom of the program. As well, the test conditions are easier to create and observing the tests is even easier.

The order in which we will be testing our classes

1. **Genre**
2. a) **Album** (Contains Genre)
   b) **Artist**
3. **Song** (Contains Album and Artist)
4. **Playlist** (Contains Song)
5. **Location** (Contains Song, Album and Playlist)
6. **HAS** (Contains the Location, Song, Album and Playlist)
7. a) **InvalidInputException**
   b) **PersistenceXStream**
8. **PersistenceHomeAudioSystem** (Contains PersistenceXStream)
9. **HomeAudioSystemController** (Contains HAS, InvalidInputException, PersistenceHomeAudioSystem)
10. **DateLabelFormatter**
11. **HomeAudioSystemPage** (Contains DateLabelFormatter)
12. **HomeAudioSytem** (Contains HomeAudioSystemPage and PersistenceHomeAudioSystem)

In addition to the unit tests, an example of a test that would be run in the integration strategy would be a persistence test. In other words, we would create a JUnit test in which we would create a song in the @Before method, test to see whether it is present in the XML file in the @Test method and later delete it in the @After method. This would be a perfect example of an integration test because it is verifying that the Song class and the Persistence classes are behaving in synchronicity. This type of test can of course be repeated for the Genre, Album, Artist, Playlist and Location classes. The test coverage of our system is essentially the effectiveness of system tests in testing the code of an entire system. Our testing goal is 85%. The tool that we will be using to support the testing will be JUnit, which is a testing framework for the Java programming language. As well, there are no discernable differences between the integration strategy of the desktop, web and mobile applications. Since all three types of applications follow near identical class diagrams and contain the same types dependencies between systems and their subsystems, the same Bottom Up Approach will be used for each type of application.