NOT USER FRIENDLY

TEST PLAN OUTLINE

**1  TEST PLAN IDENTIFIER NUF-TP2.0**

**2  REFERENCES**

* + ../Deliverables/LOCEstimation.xlsx
  + ../Deliverables/HighLevelFunctionalDia.pdf
  + ../Deliverables/TaskProcessMap.pdf
  + ../Deliverables/Software Requirements Specification.docx
  + ../Deliverables/UserDiagram.pdf

**3  INTRODUCTION**

This document will outline the testing process for the Not User Friendly project in Dr. Doyle’s CSC 440 class. Not every class and method will be tested, but the key components must have tests implemented for further development.. Any changes in the test plan will be documented in a separate revision so no ideas are lost. Functionality tests will be ran. Unit testing will be implemented as well.

**4  TEST ITEMS (FUNCTIONS)**

* Connector
  + Interacts with:

1. Google cloud
2. Cache
3. Parser/Writer
4. Player
5. Composer

* Parser/Writer
  + Interacts with:

1. Connector
2. Components

* Player
  + Interacts with:

1. Components
2. Connector

* Composer
  + Interacts with:

1. Components
2. Connector

**5 SOFTWARE RISK ISSUES**

The security risks of this project are low, but there is a potential for data loss. There is, most likely, no need to worry about data integrity with the communication from the Google cloud, but we do have to make sure files transmitted from the parser to the connector are retained and not able to be sniffed. Though there is a slight potential for this, it is not an issue. There just needs to be a mechanism in place that checks that the file has not been corrupt or any data loss from the connector to the viewer or composer. This will be done through my testing.

**6 FEATURES TO BE TESTED**

* Opening a file
* Saving a file
* Sending a file
* Receiving a file

**7  FEATURES NOT TO BE TESTED**

* UI
  + The interface will be tested for functionality, but will not require unit tests to tell us that a specific button does not work

**8  APPROACH (STRATEGY)**

Since this is not a huge project and there is not a lot of time to do it, testing will have to be done during the development phase. We are using an agile strategy, so it will be useful to have testing implemented, as soon as, the class has been written. The tester and developers will be working together to accomplish a final project, as apposed to a developer completely finishing and ‘handing’ their work to the tester. This creates an environment where the tester is against the developer. Once there are enough classes complete for a connection between actors, tests will be immediately put in place to assure that connection is working.

**9 ITEM PASS/FAIL CRITERIA**

* Unit Test Level
  + Connector
* Must be able to

1. retrieve a file from the cloud
2. save a file to the cloud
3. receive a file from the writer
4. send a file to the parser
5. cache a file
6. empty cache
7. search cache for file

* How this will be accomplished
  + The test will create a connection to the cloud initially as well as authorizing a user. ( There will be not tests of Google Api; therefore, the authorization process will not be tested. )
  + To test saving a file to the cloud, a test SMIL file will be sent to the cloud and an assertion will check that the cloud contains that file.
  + To test retrieving a file from the cloud, that same file will be saved to the cloud. Then a request to retrieve that file will be sent to the Cloud. An assertion will check that the file retrieved is the same as the file sent.
  + As for sending a receiving a file to/from the parser they will be simple getter and setter tests.
  + Testing the cache will simple run the empty Cache method and assert the count of the cache is 0.
  + To test the functionality of the cache, we will do the same thing that was done with the cloud except call the cache functions.
  + Parser/Writer
    - Must be able to

1. receive a file from the component
2. send a file request to the component
3. write user input into SMIL file
4. parse SMIL file

* How this will be accomplished
  + Again there will be a test SMIL file that will be used to assert the functionality of the Parser.
  + To test the sending and receiving files from the connector, we will use the connector to send a message. And assert the Parser received the message and visa versa for sending to the connector.
  + To test that the parser iterates through the SMIL file correctly and parses it, we will run the method to parse the file and assert all data was obtained in the proper way and format.
  + Composer
    - Must be able to

1. send user input to parser/writer
2. receive user input

* How this will be accomplished
  + The composer has not been completely finished and a test plan for this specific function has not been decided upon.
  + Player
    - Must be able to

1. send user input to parser/writer
2. receive user input

* How this will be accomplished
  + Using the sample SMIL file, the player will be tested to assert when the play button is pressed the video starts.
  + It will assert that the pause button stops correctly and the current time will equal the correct value.
  + The stop button will be tested as well, to assert the video is not playing.
* Functionality Test Level
  + Once Unit tests are complete
  + Test the program as a user
  + Document how the program works
* Performance Test Level
  + Once Functionality tests are complete
  + Create scripts to test how the program performs

1. when multiple users compose a file
2. when multiple users try to access a file
3. how the cache performs

**10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS**

Most of the unit tests will not have requirements of another test to pass for it to run. There may be a test or two that do require a connection to pass a test for it to run. Then, that test would be halted on previous test failure. These tests will be design so that will not have to happen though.

**11 TEST DELIVERABLES**

* Test plan document
  + NUF-TP1.0
  + NUF-TP2.0
* Test cases
  + testing/src/csc440/nuf/complay/TimerTest.java
  + .
  + .
  + .
* Functionality test documents
  + Errors logged
  + Notes of user interface
* Performance test documents
  + Scripts that ran the tests
  + Diagrams of results
  + files that resulted from scripts

**12  REMAINING TEST TASKS**

All

**13  ENVIRONMENTAL NEEDS**

* + JVM
  + Android SDK
  + Google API Cloud
  + JUnit installed
  + Scripts will be written by tester for performance
  + Possible ruby installed for scripts to output graphs
  + Wifi or 3G ??
  + System restricted while tests are being ran
* users would have an impact on test performance

**14  STAFFING AND TRAINING NEEDS**

None

**15  RESPONSIBILITIES**

* Alex
  + Create all tests
  + Write and modify Testing plan
  + Run all tests
  + Communicate with developers on errors
  + Communicate with project manager on deliverables
* Jake
  + Modify any code that does not pass tests
* Brad
  + Discuss performance and deliverables

**16 PLANNING RISKS AND CONTINGENCIES**

* Late delivery of the software