NOT USER FRIENDLY

TEST PLAN OUTLINE

**1  TEST PLAN IDENTIFIER NUF-TP1.0**

**2  REFERENCES**

List all documents that support this test plan. Refer to the actual version/release number of the document as stored in the configuration management system. Do not duplicate the text from other documents as this will reduce the viability of this document and increase the maintenance effort. Documents that can be referenced include:

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

**3  INTRODUCTION**

This document will outline the testing process for the Not User Friendly project in Dr. Doyle’s CSC 440 class. This plan is used to keep track of the whole testing process that was gone through until the final copy. The plan will cover all classes implemented in this project. Not every method will be tested, but, since there are not many classes in the entire project, each one holds high importance, therefore, it must be tested for errors. Any changes in the test plan will be documented in a separate revision, so no ideas are lost. Functionality tests will be run. Unit testing will be implemented next during development. Lastly, performance testing will be done.

**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. Once all classes have been written functionality

**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. log in a user to google cloud
6. cache a file
7. empty cache
8. search cache for file
   * Parser/Writer
     + Must be able to
9. receive a file from the component
10. send a file request to the component
11. write user input into SMIL file
12. parse SMIL file
    * Composer
      + Must be able to
13. send user input to parser/writer
14. receive user input
    * Player
      + Must be able to
15. send user input to parser/writer
16. receive user input

* 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
* Test cases
  + .
  + .
  + .
  + .
  + .
* 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
* Chris
  + Modify any code that does not pass tests
* Brad
  + Discuss performance and deliverables

**16 SCHEDULE**

Week Date

1. Feb 2-8
   1. Complete first draft of Test Plan
   2. Start writing test cases for Parser
2. Feb 9-15
   1. Finish test cases for Parser/Writer
3. Feb 16-22
   1. Start and Finish Connector tests
4. Feb 23-29
   1. Start Player/ Composer tests
5. Mar 1-7
   1. Finish Player/Composer tests
6. Mar 8-14
   1. Complete Functionality tests
   2. Document Functionality test
7. Mar 15-21
   1. Create scripts to complete performance test
8. Mar 22-28
   1. Run Performance tests
   2. Document all performance tests
9. Mar 29-Apr 4
   1. Create Diagrams of all tests
10. Apr 5-11
    1. Complete all deliverables

**17 PLANNING RISKS AND CONTINGENCIES**

* Late delivery of the software