Software Testing: Documentation and Agenda

MCS 593 Tutorial Management Web Application

**Version 1.0**

**Prepared by:  
Alec Meyer  
Robert Padilla**Anthony Prejean  
**James Kubena  
Britton Ware**

**CS 3398: The Back Row**

March 25, 2015

**Table of Contents**

1. **Preface………………………………………………………………………………………...3**
   1. **Purpose…………………………………………………………………………………….3**
   2. **Document Conventions…………………………………………………………………….3**
   3. **Audience…………………………………………………………………………………...3**
2. **Macro Test Plan and Objectives…………………………………………………………….3**
   1. **Goals………………………………………………………………………………………3**
   2. **Schedule…………………………………………………………………………………..4**
3. **Development Testing..……………………………………………………………………….5**
   1. **Unit Testing………………………………………………………………………………..5**
4. **Object Class and Method Testing………….…………………………………………..5**
   1. **Component Testing………………………………………….……………………………..6**
5. **Testing Policies………………………………………….……………………………..6**
6. **Interface Testing………..……………………………….……………………………..6**
7. **Parameters…..……..……………………………….……………………………..6**
8. **Shared Memory...…..……………………………….……………………………..6**
9. **Procedural…..……..…………………………….….……………………………..6**
10. **Message Passing………………………………….….……………………………..6**
    1. **System Testing……………………………………………………………………………..7**

**1. Interface Testing…………….…………………………………………………………..7**

**2. Use Case Testing…………….…………………………………………………………..7**

**1. Use Case 1: Logging in……..………………………………………………………..7**

**2. Use Case 2: Scheduling a session…...………………………………………………..8**

**3. Use Case 3: Cancelling a session……………………………………………………..8**

1. **Release Testing………………...……………………………………………………………..9**
   1. **Requirements Testing……...…………...………………………………………………..9**

**1. Non-Functional Requirements Testing………….………………………………..9**

**2. Functional Requirements Testing………………………………………………..9**

* 1. **Performance Testing……...……………………………………………………………...9**

**5. User Testing……...……………………..…...………………………………………………10**

**1. Alpha Testing...……………..………..…...………………………………………………10**

1. **Beta Testing. ...……………..………..…...………………………………………………10**
2. **Acceptance Testing.. ………..…………....………………………………………………10**

**1. Preface**

**1.1 Purpose**

The purpose of this document is establish testing procedures that must be passed in order to implement the Tutoring Lab software introduced in previous documentation. The tests will demonstrate to the developer and end users that the software meets the outlined requirements in the Software Requirements and Specification document, and discover any situations in which the behaviors of the software does not conform to specified requirements.

## 1.2 Document Conventions

This document uses a dot-number outline format to organize the information presented by the group. This information contained in this document has been organized categorically. The table of contents may be used as a reference to the sections of information.

## Audience

This testing document is intended for end users and developers of the Tutorial Management Web Application.

2. **Macro Test Plan and Objectives**

**2.1 Goals**

The goal of the tests that are specified in this document are to ensure the software is fit for the purpose of managing the Computer Science tutorial services offered in MCS 593. The results will demonstrate to the developer and end users through defect and validation tests that the software meets the outlined requirements in the Software Requirements and Specification document, and specify any situations in which the behavior of the software does not conform to specified requirements.

* 1. **Schedule**

**Stage 1: Development Testing**

**Unit Testing: Object Class and Methods**

* + - 1. Create automated tests for partitioned classes/methods.
      2. Test each class and method with normal and abnormal conditions.

**Component Testing: Interfaces**

1. Develop Testing Policies for Interfaces
2. Run Interface tests for parameters, shared memory, procedural and message passing

**System Testing: Component Integration and Usage**

1. Integrate component subsystems and test the interactions and interfaces for parameters, shared memory, procedural and message passing by the subsystems.
2. Test the integrated system against the specified use cases in the Use Case Documentation.

**Stage 2: Release Testing**

**Requirements Testing**

1. Create usage scenarios for each requirement in the Software Requirements and Specification Document.

2. Test integrated release version of the software for each specified requirement in the Software Requirements and Specification Document.

**Performance Testing**

3. Run performance stress tests on system until failure to determine limits, dependability, reliability, and capability.

**Stage 3: User Testing**

**Alpha and Beta Testing**

1. **Alpha** testing done with end users and tested at development site. End users should supply inputs, scenarios, and advice for testing system.
2. **Beta** release of software will be released to end users for experiment for tested in MCS 593 Tutoring Lab environment.

**Acceptance Testing**

1. Develop acceptance criteria for acceptance with end users, plan tests and create tests based on this criteria.
2. Run acceptance tests and document results for negotiation of acceptance with end users.

**3. Development Testing**

* 1. **Unit Testing**
     1. **Object Class and Method Testing**

The following test cases will be used in coordination with diagrams 1.6.1-1.8.1 in the Software Architecture Document. Classes and methods should be partitioned into groups of similar inputs and outputs:

Partition 1:

stuCancelAppt(int time)

tutCancelAppt(int time)

stuMakeAppt(int time)

tutMakeAppt(int time)

Partition 2:

tutLogin(str netID)

stuLogin(str netID)

checkLogin(str netID)

Example inputs with expected outputs for creation of automated testing code for verification of individual unit functionality:

Output should return true:

Valid input cases for time(using military):

Partition 1:

900

1500

1630

2100

Examples of valid ID’s:

Partition 2:

Hu6866

Go1234

ID5813

Ra1123

Output should return false:

Example invalid input cases for time(using military):

Partition 1:

765(not open)

913(invalid increment)

2400(closed)

Astring(invalid format)

Examples of Invalid ID’s:

Partition 2:

111111

Aaaaaa

1234no(invalid formats)

Az0000(invalid number)

Output should behave according to specifications:

Partition 1:

Boundary input cases:

800(opening time)

2200(closing time)

2145(insufficient tutoring time)

Partition 2:

Any specified ID’s that should have access that do not conform to normal format.

***Will add more/Any suggestions welcome***

* 1. **Component Testing**

The following test cases will be used in coordination with diagrams 1.6.1-1.8.1 in the Software Architecture Document.

**3.2.1 Testing Policies**

1. Each input case checked in 3.1 must be retested in now with integration of other components to ensure the verify output is correct.

2. Each function in the diagrams should be tested to verify the function completes its task for each specification.

3. User interfaces should be tested individually and in conjunction to verify correct functionality and error recovery in both cases while integrated.

4. Each function that requires input should be tested with valid and invalid data.

5. Each function that manipulates the tutoring queue should be tested to verify the correct result in conjunction with other queue manipulation functions.

6. Each function with a pointer must be tested with a null pointer.

7. Appointments must be created, and cancelled by both the student and tutor in different orders and sequences from one account and more to ensure no issues accessing shared memory.

8. Functions must be tested by increasing load to determine boundary conditions and stress limit. Corresponding boundaries should also be tested.

6. ***More can be added/Any suggestions welcome.***

**3.2.2 Interface Testing**

Each of the following should be tested in accordance with 3.2.1 Testing Policies:

**3.2.2.1 Parameters**

**3.2.2.2 Shared Memory**

**3.2.2.3 Procedural**

**3.2.2.4 Message Passing**

* 1. **System Testing**
     1. **Interface Testing**

Interface testing for the integrated system should be done to ensure correctness and functionality of each interface subsystem in accordance with the testing policies listed in 3.2.1. Interfaces to be tested are listed above in 3.2.2 for each subsystem that has been integrated.

* + 1. **Use Case Testing**

* + - 1. **Use Case 1 – Logging into the system**

Login functionality should be tested in both the student and tutor interfaces. It is recommended to create an automated program to select and run the tests:

-Attempting to login with 50 valid logins.

-Attempting to login with 50 invalid logins.

-Attempting to login with 50 invalid and valid logins simultaneously (stress) (using threads for example).

-Attempt to login with 50 valid logins that are not in a CS class, and 50 valid logins that are in a CS class.

-Any nonconformist results that vary from result A or B specified for Use Case 1 in the Software Requirements and Specification should be documented and addressed.

* + - 1. **Use Case 2 – Scheduling a tutoring session**

The tutor scheduling system should be tested on the student and tutor interfaces. It is recommended to create an automated program to select and run the tests:

-Attempting to schedule a student session using the maximum number of different valid logins until the tutoring lab is full to ensure behavior of upper boundary.

-Attempting to schedule exactly 1 hour of tutoring using 50 different logins.

-Attempting to schedule exactly 2 hours of tutoring using 50 different logins.

-Attempting to schedule exactly 3 hours of tutoring using 50 different logins.

-Attempting to schedule exactly more than 3 hours of tutoring using 50 different logins.

**Use Case 3 – Cancelling a tutoring session**

The tutor cancelling system should be tested on the student and tutor interfaces. It is recommended to create an automated program to select and run the tests. Number of appointments should be checked for student and tutoring lab:

-Attempt to schedule and cancel 1 appointment and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule and cancel 2 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule and cancel 3 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 1 and cancel more than 1 appointment and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 2 and cancel 1 appointment and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 2 and cancel 2 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 2 and cancel 3 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 3 and cancel 1 appointment and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 3 and cancel 2 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

-Attempt to schedule 3 and cancel 3 appointments and verify correct schedule results and appointments for 50 different logins, and 10 different timeframes per login.

**4. Release Testing**

**4.1 Requirements Testing**

**4.1.1 Non Functional Requirements Testing**

These requirements also are tested in the tests outlined in 3.3.2. Further testing for consistency is outlined in this section.

-**Reliability** of the system can be further tested by an extension of the tests run in 3.3.2. This can be tested using an automated program and threads to steadily increase the load placed on the application to verify correctness and timing to avoid mis-scheduling due to data race.

-**Robustness** of the system can be further tested by attempting to login and allowing to timeout with 50 different logins, and attempting to overschedule previously scheduled sessions with 50 different logins to verify application correctly handles error.

-**Performance** testing is further specified in 4.2, but also tested in the outlined tests in 3.3.2.

**4.1.2 Functional Requirements Testing**

Functional requirements can be tested using the tests that have been outlined in section **3.3.2 Use Cases**, as they are identical to the functional requirements.

**4.2 Performance Testing**

The release system should be tested for performance by creating high load of students attempting to login and schedule and/or cancel an appointment. This can be easily implemented by creating a thread based program and steadily increasing the number of threads attempting to login and make changes. Student eligibility, appointment count, login verification, lab count, and schedule should all be checked for accuracy. Performance results as load is increased should be documented proper use.

**5. User Testing**

* 1. **Alpha Testing**

Alpha testing will be done with end users and tested at development site. End users should supply inputs, scenarios, and advice for testing system.

* 1. **Beta Testing**

Beta release of the application for testing in the MCS 593 Tutoring lab will be released in the Texas State Datacenter as a client-server system, accessible via web interface, once Alpha testing has completed.

**5.3 Acceptance Testing**

Acceptance criteria details will be discussed with end users and department chair, Hong Shi.