**Table of Contents**

[Implementation Status Report 2](#_Toc277807151)

[User Manual for RUM System 3](#_Toc277807152)

[Testing for Graphical User Interface 4](#_Toc277807153)

[Tests for Database System 5](#_Toc277807154)

[JUnit Tests 6](#_Toc277807155)

[Team Contributions 9](#_Toc277807156)

# Implementation Status Report

This document will contain currently implemented features for our RUM system (requirements and design) as well as any bugs or limitations known.

**Requirements Satisfied/Partially Satisfied**:

* Currently we have 90% of all methods in place and either complete or partially complete. In terms of specific requirements we have implemented our check requirements method and it is almost fully functional. Our scheduling algorithm is 80% complete and is about 50% implemented, but we are still having bugs with certain sections of the functionality. Most other small requirements have been met by our implementation forgoing some basic methods.

**Design Implemented/Partially Implemented**:

* Graphical User Interface:
  + All screens included in our design are accounted for.
  + All buttons correctly link and change to the associated screen chosen.
  + All screens contain the tables and buttons needed to fully implement all requirements in our design (as mentioned above).
  + Due to partial implementation of certain things the GUI is now experiencing some minor issues that will be corrected. (Certain buttons do not link correctly)
* Persistence Database:
  + Created using ApacheDerby and done through NetBeans.
  + Contains all needed tables with appropriate attributes and primary keys included.
  + All code for the database can be found in the server folder of the submitted files.
* Protocol/Sockets:
  + All sockets have been written and are functional.
  + The client end of the protocol is 100% complete.
  + The server side of the protocol is about 50% complete.
* File Writing/Parsing
  + Both the file writer and parser have been written and tested.
  + Full functionality isn’t available due to lack of implementation of certain areas.
  + Changed design from CSV files to use XML for file formats.

For our project we have estimated that we have approximately implemented 60-65% of our project. We feel most of the code is written or almost complete, but there are many bugs and functionality issues that we need to fix. Also the connection between server and client through the protocol is not yet complete with hurts our implementation a lot.

# User Manual for RUM System

This will be a basic guide using our RUM system including: (1) installing the necessary components to your computer; (2) running the server for the application program; and (3) starting up the client program on your computer.

This system is a client/server application that checks requirements for a student’s graduation. It allows students to upload/add courses they have taken including their grades and if they are transfer credits. They can then check these classes against a list of requirements for whatever major they have selected. Then based off what requirements they have not met the system can generate a suggested schedule of classes they can take to graduate. The system also allows for user admins to log in, add/edit/remove majors and departments, add/edit/remove requirements for majors and upload/add course offerings for a school.

**Installing/Configuring/Starting Rum System on Server Side**

To use this system the user must have the following on their computer:

* Apache Derby – See http://db.apache.org/derby/derby\_downloads.html

**Running the Server Program.** User must run this before using the client.

1. Download the server .JAR executable file.
2. Double click the server .JAR executable file.

You can now proceed to running the client application.

**Installing/Configuring/Starting Rum System on Client Side**

To use this system the user must have the following on their computer:

* Java Runtime Environment – See <http://www.java.com/en/download/index.jsp>

**Running the Client.** For the client side there is no actual install on the computer.

Note: User must have the associated server running before using the client.

1. Download the client .JAR executable file.
2. Double click the client .JAR executable file.

You can now begin to use the RUM system.

# Testing for Graphical User Interface

**Student Run Through**

When we first start the application we correctly load our welcome screen. If we begin by clicking the Student option we change screens to the Student page. From there we have multiple options. If we click the add course button we navigate to the add course page. On that page we have a drop down menu to select a grade and a transfer check box. As of now both the ok and cancel button brings us back to the student page. If we click on the edit course button next from the student page we have the same screen as add screen page except the page says edit course page. The remove course button on the student page does not do anything but when it is implemented it will directly remove the highlighted course from the table on the student page. The upload course and download courses as of now do nothing but they will allow the user to download or upload courses from a specific file format once it is implemented. The check requirements and generate schedule also currently do anything.

**Department Admin Run Through**

If we hit back from the student page we go back to the welcome screen. From here we can click on department administrator. This brings us to the log in page where we must enter a username and password and then click log in. As of now we can simply log in regardless of the input but in the future we will check the user name and password. The back button brings us back to the welcome screen. If we click on log in we are brought to the Department Administrator Page. The add major button pops up a dialogue box where we can enter a major. As of now ok and cancel both bring you back to the department administrator page. Edit major button pops up a dialogue box as well where you can enter the major you wish to change. Remove major button does nothing as of now but it should automatically remove a major from the table on highlighted major on the screen. The edit requirements button brings up a new page called requirements. From here we can add remove or edit. Clicking back brings us back to the department administrator page. From here if we click back again we can go back to the welcome screen.

**Registrar Admin Run Through**

We now will choose are last option from the welcome screen which is registrar administrator. We will go through the same login screen which will bring us to the registrar admin page. From here we have three options. The first option is clicking view edit courses button. This brings us to the edit department page. Once again we have several options. Add course brings us to the add course page. In this page there is a bunch of information that can be typed in such as name, number, department, and description. As of now we don’t actually save anything so ok and cancel do the same thing (bring us back to edit department page). The remove course button will delete a course from the table in the future. Currently the edit course button is the same as the add course button with just a different title. Upload courses will allow the user to import a file in the future by clicking the browse button.

# Tests for Database System

**Test 1:** Add Major to Database

This tests whether or not data for majors can be added to the database.

Precondition: User must be logged in as admin(either) and major has valid information in all required fields.

1. Major information is read into system.
2. JDBC inserts the data into the database table through the JPA.

Requirements successfully associated with major.

Outcome: Test was successful. Data inserted into correct table, objects associated with correct major.

**Test 2:** Add Course to Database

This tests whether or not data for courses can be added to the database.

Precondition: User must be logged in as a Registrar Admin and course has valid information in all required fields.

1. Course information is read into system.
2. JDBC inserts the data into the database table through the JPA.

Outcome: Test was successful. Data inserted into correct table, objects associated with correct course.

**Test 3:** Basic Socket Communication

Passes commands and object over communication channel between client and server

Precondition: Connection between client and server established

1. Client send message
2. Server receive message
3. Server generate response
4. Server send response
5. Client receive/handle response

Outcome: Test was successful. Messages and objects are passed between client and server. However, some of functions available in the protocol are incomplete.

# JUnit Tests

JUnit tests may be found in the following files:

* CourseGroupTest.java
* CourseTest.java
* MajorTest.java
* RequirementTest.java
* SystemManagerTest.java
* UserTest.java

A small sample of tests are listed here, but many more complete (and incomplete) tests may be found in the JUnit files.

**Test 1:** Generate Schedule

This tests generates a schedule,

Precondition: User must have a major and major must have requirements, also, courses must exist in system.

1. Get remaining courses for all requirements
2. Assemble courses in a tree

Outcome: Test was partially successful. Bugs exist in overall algorithm, but inner methods return correct value.

**Test 2:** Writing file

This tests whether or not data for courses can be added to the database.

Precondition: Data must be present to Write to file

1. Compile information
2. Write information to string, which will be sent over protocol

Outcome: Test was not successful. Data inserted into correct table, objects associated with correct course.

**Test 3:** Parsing file

Generates info from uploaded Files

Precondition: File is in valid xml format

1. Reads info in from xml file
2. Generates objects based on file data, object types based on file information

Outcome: Test was successful. Three types of files tested generated correct objects.

# Team Contributions

William Matrix Peckham

* Continued to work on GUI.
* Created Action Listeners

Chris Scarola

* Wrote file writing implementation.
* Updated implementation status report and other documentations.
* Helped with developing scheduling algorithm and filled in/added small methods.

Tom Biegner

* Wrote currently implemented protocol and sockets.
* Wrote check requirements method and scheduling algorithm/code.
* Wrote and conducted many J-Unit tests.

John Paul Pennisi

* Continued to work on GUI and did form validation.
* Has been writing the ethics homework in collaboration with the team.

Homework 8: Code 2

John Paul Pennisi

William Matrix Peckham

Chris Scarola

Thomas Beigner