

# COBA/SITC Course Scheduler

SE14F

Nov 16, 2014

By Team Zaboo (Team 3)

Raj Patel,

Jessica Lin,

Landon Gray

<u>Table of Contents:</u>	<u>Page</u>
<b>I. DESIGN</b>	<b>4</b>
<b>A. Introduction</b>	<b>4</b>
1. Problem Statement	4
2. Mission	4
3. Design Goals	4
4. Estimated Time of Completion	4
<b>B. Architecture</b>	<b>5</b>
1. Input/Output	5
2. Platform/Language	5
3. Data	5
4. Sample Schema	6
<b>C. Code</b>	<b>7</b>
1. Introduction	7
2. Frameworks	7
3. Google Calendar API	7
4. User Case Diagram	7
5. ERD Diagram	8
6. UML Diagram	9
7. Views	9
8. API (Application Programming Interface)	9
<b>D. Operation</b>	<b>10</b>
1. User Types	10
2. Scenarios	10
<b>E. Miscellania</b>	<b>10</b>
<b>II. SPECIFICATIONS</b>	<b>11</b>
<b>A. Overview</b>	<b>11</b>
<b>B. Scenarios</b>	<b>11</b>
<b>C. Extra Features</b>	<b>13</b>
<b>D. Views</b>	<b>13</b>
<b>E. View Samples</b>	<b>13</b>
1. Log In View	13
2. CRN, Time, and Day Submission View	14

3. Calendar Conflict View.....	14
<b>F. Data Flow Diagram.....</b>	<b>15</b>
<b>III. VERIFICATIONS.....</b>	<b>16</b>
<b>A. Black Box Testing.....</b>	<b>16</b>
1. Feature: The user should be able to log into the system.....	16
2. Feature: The inputs CRN,requested time, days to database.....	16
3. Feature: Finding Conflicts.....	17
4. Feature: Conflict Possibilities.....	17
5. Step Definition.....	20
6. Results of Implemented Cucumber.....	23
<b>B. Module Testing.....</b>	<b>25</b>
1. Sign-in Testing.....	25
2. CRN-sending Testing.....	25
3. Color-coded Testing.....	25
<b>C. Integration Testing.....</b>	<b>25</b>
1. Log-in Testing.....	25
2. Data-input-validation Testing.....	25
3. Conflict-search Testing.....	25
4. Conflict-calendar-display Testing.....	26
<b>D. System Testing.....</b>	<b>26</b>
<b>E. Acceptance Testing.....</b>	<b>26</b>
<b>F. Performance Testing.....</b>	<b>27</b>
<b>G. GOMS Testing.....</b>	<b>27</b>
1. Keystroke Level Model.....	27
2. Goals.....	27
3. Results of GOMS Tests (Users' Average).....	28
<b>H. Status Reports.....</b>	<b>28</b>
<b>IV. GLOSSARY.....</b>	<b>29</b>
<b>V. BIBLIOGRAPHY.....</b>	<b>31</b>

# I. DESIGN

## A. Introduction

### 1. Problem Statement:

After students are enrolled for the next semester, the process to move sections is difficult.

### 2. Mission:

Provide a web based application that searches for the most efficient time to hold a rescheduled course in a student schedule.

### 3. Design Goals:

- Help identify possible faults in our understanding about the overall system
- Help assist developers to identify areas where we can design for change.
- Provide the client with an overall outline how the web app will be laid out
- Help identify and point out areas where security methods need to be implemented
- Express how this web app will meet the needs and goals of our client

### 4. Estimated Time of Completion:

The magnitude of this project should take 80 hours to complete. Cost will be \$1200.

## B. Architecture

### 1. Input/Output:

Users will be able to enter in a CRN and a potential time change. The results will consist of the total number of conflicts and which classes are involved in them. The results with details will show which student conflicts are important by student classification and the frequency a course is offered. This will be represented in calendar form, day of the week by time. Each entry will be highlighted a color of severity based off of the aforementioned measure of importance. There will also be a similar list of room availability in case a class needs to relocate as well.

### 2. Platform/Language:

For this project we will be using a system that runs on the Internet. We have chosen to use PHP as our main language for our web application. We will be running our web app on a server with a LAMP stack.

### 3. Data:

One of the largest portions of this project is the database. In this database will be the list of classes by CRN along with each classes' student roster, time of day, day of the week, and room number. There will also be each student's schedule, transcript, and classification. This information is highly sensitive so security must be provided and will be enforced through an application level authentication. The database will be interfacing with banner to get information from along with the front end of our project to print out this information.

We will be using a standard relational database called mysql. This design choice is due to the fact that we are using a LAMP/WAMP stack. It is used in many companies and open source projects and because of that it has been widely tested and is somewhat secure. In the future if developers decide to go with a different database system it will have minimal effect on our design. The only modules that will be affected are our api and the database config file. Our view will not be affected because they only call the api.

There will be five entities with several attributes. The first table will have the details of 4771 students at ACU. We will also need a course table. The third is the student schedule table, typically ranging from 3-6 courses. We need this entity in order to implement a many to many relationship between the students and sections table. The last table is a transcript, which provides all the information of the classes the student has taken.

## 4. Sample Schema

```
CREATE TABLE section(  
  crn int primary key,  
  section_num int,  
  start_date date,  
  end_date date,  
  days varchar(5),  
  start_time time,  
  end_time time,  
  department varchar(20),  
  room_num int,  
  enroll_max int,  
  enroll_now int,  
  instructor varchar(30),  
  sts_code varchar(5),  
  sts_description varchar(10),  
  course_id int  
);  
  
CREATE TABLE schedule(  
  student_id int,  
  section_id int,  
  primary key(student_id, section_id)  
);  
  
CREATE TABLE student(  
  banner int primary key,  
  pdim int,  
  email varchar(40),  
  f_name varchar(18),  
  l_name varchar(18),  
  classification varchar(10),  
  major varchar(20),  
  expected_graduation int  
);  
  
CREATE TABLE transcript(  
  id int primary key,  
  student_id int,  
  section_id int,  
  grade varchar(1)  
);  
  
CREATE TABLE course(  
  id int primary key,  
  prefix varchar(5),  
  course_num varchar(3),  
  prereqs_id int  
);
```

## C. Code

### 1. Introduction:

Languages we will be using in our project include, markup, stylesheets. HTML, Javascript and CSS for front-end. We will also be using PHP for back-end.

### 2. Frameworks:

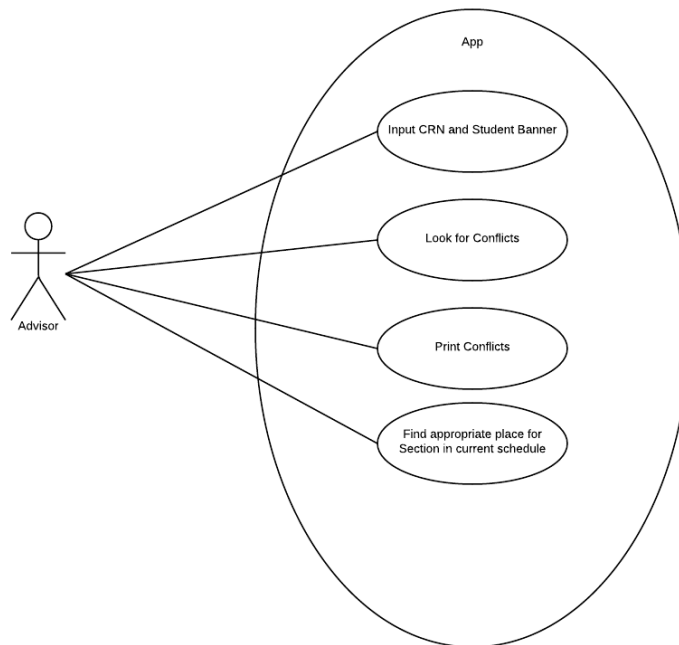
*Bootstrap:*

Bootstrap (<http://getbootstrap.com/>) is a web framework used to make front-end development easy and fast. This will help us create clean user interfaces.

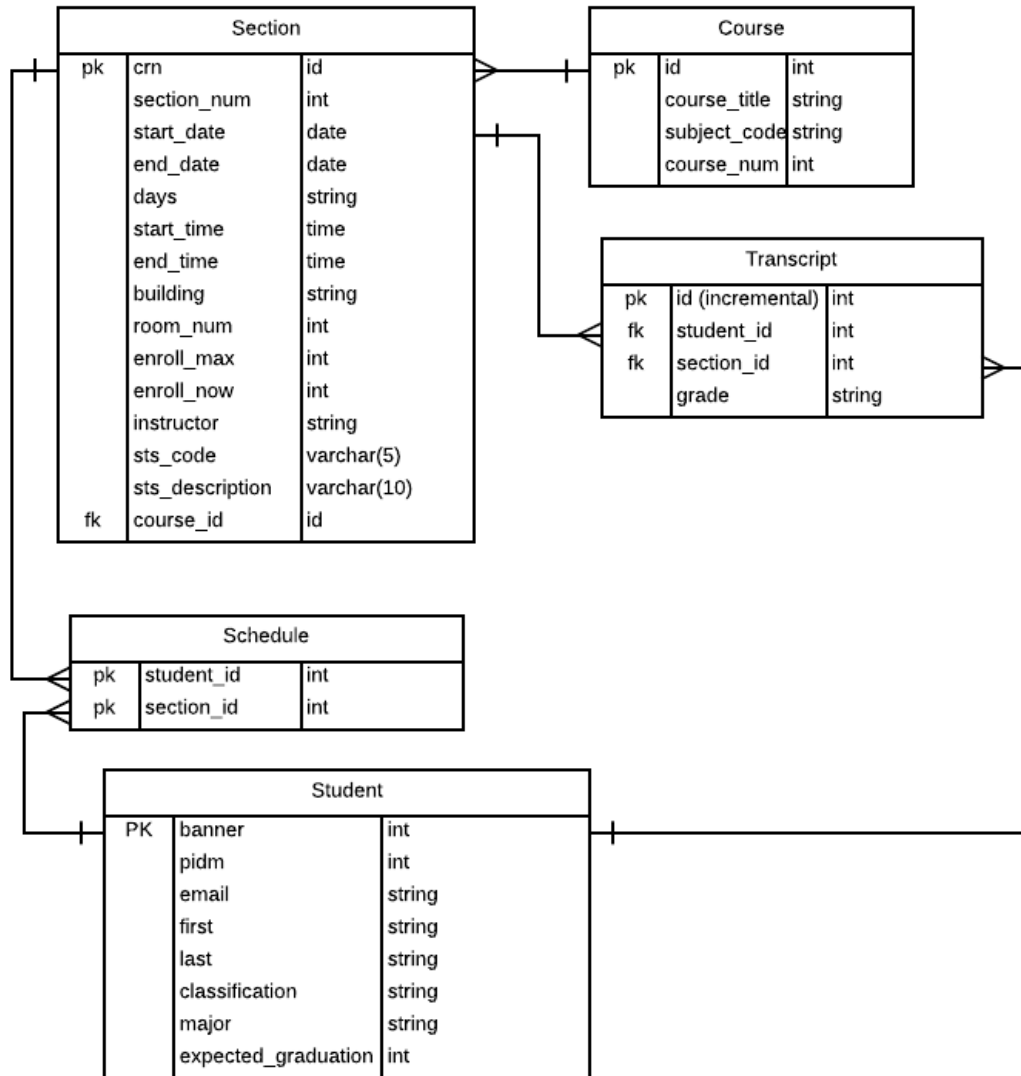
### 3. Google Calendar API:

We will use the FullCalendar API (<http://fullcalendar.io/docs/>) in our application in order to show the results of our clients queries into the system in a visually appealing way

### 4. User Case Diagram (Advisor):

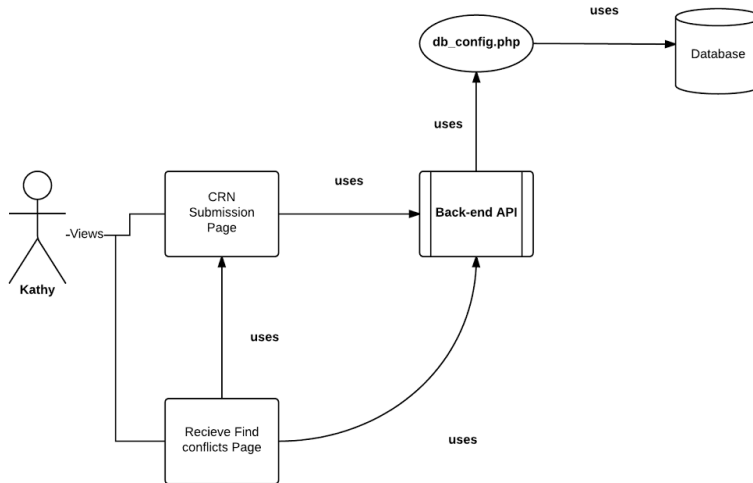


## 5. ERD Diagram:





## 6. UML Diagram:



## 7. Views:

- *Log in with administrative username and password:*  
User inputs their username and password and credentials are verified
- *Submit CRN View:*  
User Kathy submits CRN and desired time for course specified. Returns a Receive/ Find conflict view
- *Receive/ Find Conflicts View:*  
The module displays the course conflicts that have occurred based on what user submits via CRN View.

## 8. API (Application Programming Interface):

We currently have one API that could later be integrated into banner or even replaced by banner if needed.

- *Back-end/API:*  
We will create an API with available function calls that return required data from the database. These will essentially be a series of sql queries on the database.

## **D. Operation**

### **1. User Types:**

- Administrator: Data and User Management
- Advisor: searches conflicts for a schedule in a certain semester. (not the ones in the past)
- Students: just like advisors, but less output due to limited permissions.

### **2. Scenarios:**

- Inputs student data(banner, classification, first name, last name, and a list of sections)
- Search in database for conflicts by:
  - a. time
  - b. classroom availability
- Output conflicts for next semester.
- In Spring, output conflicts in summer.

## **E. Miscelanea**

This project would ideally be distributed to students, not just administrators and advisors so that students can check conflicts with their own schedule. We can easily abstract the portion that finds conflicts from the entire banner database and every other student schedule so that only that bit of code would be utilized by these unauthorized figures. Therefore, the only changes that would be made would be authorization. This would accept student username and password and see they are not faculty, and automatically limiting what's outputted to be just information that pertains to them,

Debugging is always a handful, but hopefully, by following the incrementality principle of design we can test with portions of the project to avoid searching through every bit we designed for some small crack.

## II. SPECIFICATIONS

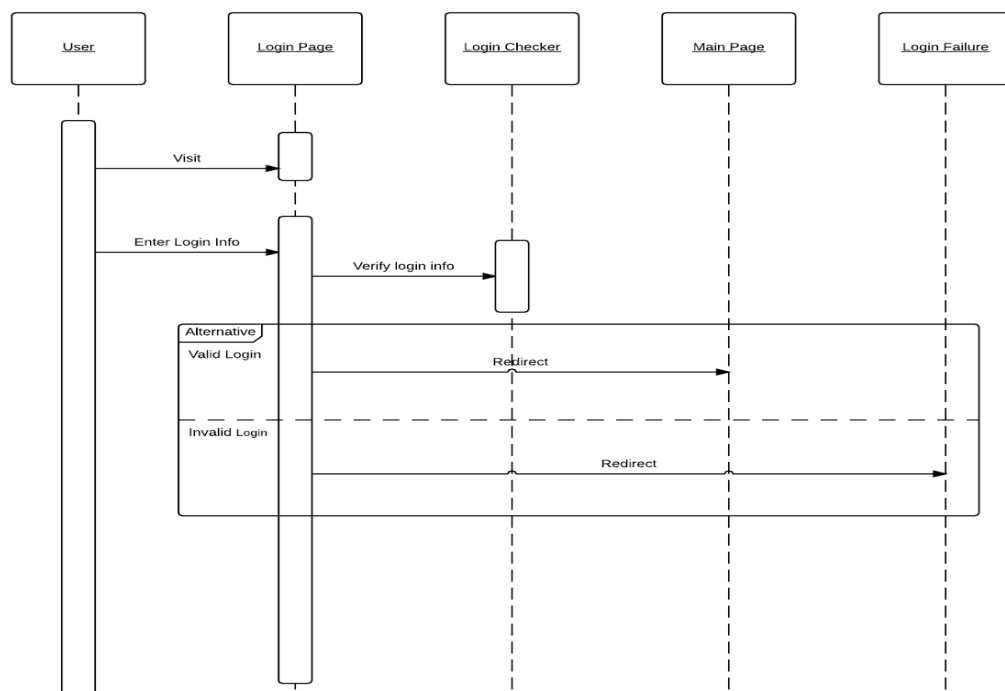
### A. Overview

There is no automated way of comparing student schedules to see the best time to move a class. The client specified that after students are enrolled in a class it is difficult to move class times. These class times conflict with other classes. This project will provide Karen with a web based application that checks student schedules to see the best time to hold a rescheduled course.

### B. Scenarios

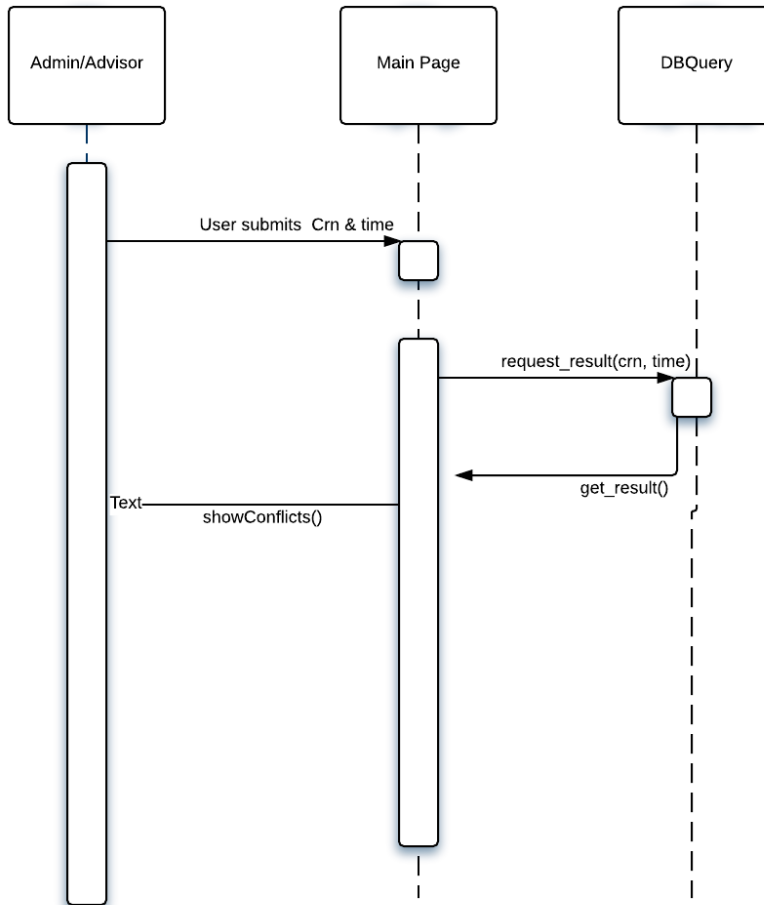
- Advisor is trying to log in but forgot her password. She clicks on the link “Forgot your Password?” Then her email address is requested. She types in her email address and then the password reset request has been sent to her email.
- Same thing happens if she forgets her username.

**Sequence Diagram for username and password validation:**



- Advisor or administrator enters in course number and time of day. New view appears showing a grid with time on the vertical axis and days of the week on the horizontal axis. Each grid slot will show a color of severity based on the level of conflicts.

**Crn and time submission:**



### C. Extra Features

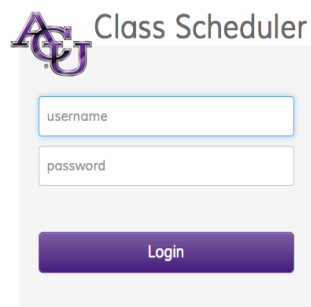
- If time allows it, we will include, along with student schedule conflicts, conflicts with classrooms based off of class size and/or room availability.
- Allow professors to test out class time changes rather than make advisors and administrators do it.
- Make the application usable by students to test out adding a class to their schedule when getting ready to register.

### D. Views

- Log-in page for administrators/advisors
- Submit CRN view should contain a form box for the CRN and a drop down box for Kathy to be able to select time desired for the specified course
- Find conflict view should display course conflict. Specifically a calendar should be displayed from our calendar api that shows conflicting times in red usable times in green. In the red, if the user clicks it, all of the conflicts would appear

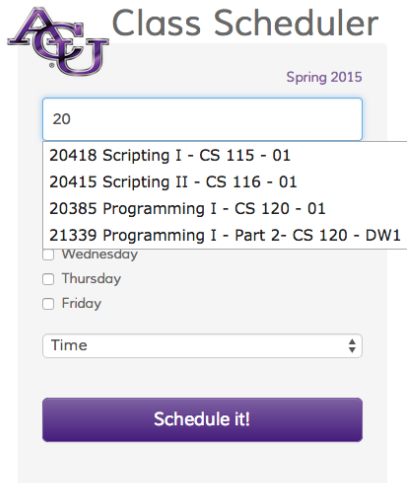
### E. View Samples

#### 1. Log In View:



The image shows a login interface for the ACU Class Scheduler. At the top left is the ACU logo, followed by the text "Class Scheduler". Below this, there are two input fields: one labeled "username" and another labeled "password". At the bottom of the form is a purple button with the text "Login".

## 2. CRN, Time, and Day Submission View:



**A&U Class Scheduler**  
Spring 2015

20

20418 Scripting I - CS 115 - 01  
20415 Scripting II - CS 116 - 01  
20385 Programming I - CS 120 - 01  
21339 Programming I - Part 2- CS 120 - DW1

☐ Wednesday  
☐ Thursday  
☐ Friday

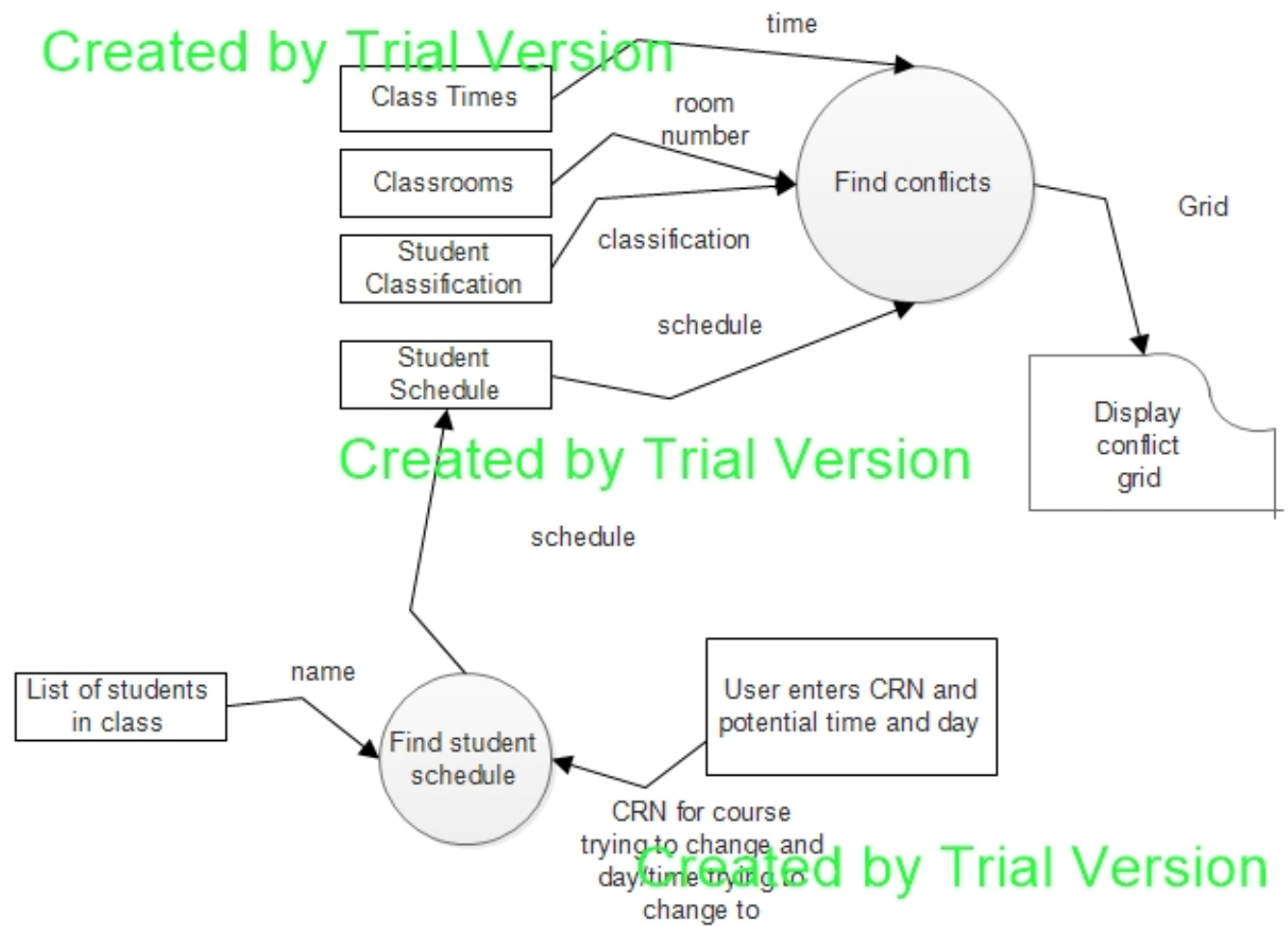
Time

**Schedule it!**

## 3. Calendar Conflict View:

September 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1 All Day Event	2	3	4	5	6
7 Long Event	8 4p Repeating Event	9	10	11 Conference	12 10:30a Meeting 12p Lunch +3 more	13 7a Birthday Party
14	15 4p Repeating Event	16	17	18	19	20
21	22	23	24	25	26	27
28 Click for Google	29	30	1	2	3	4
5	6	7	8	9	10	11

**F. Data Flow Diagram**

### III. VERIFICATIONS

#### A. Black Box Testing

1. **Feature:** The user should be able to log into the system.

Scenario: User submits correct username and password

Given user is on login page  
And user types correct username  
And user types correct password  
And user clicks the login button  
Then the user logs into the system  
Then scheduler view page should display

Scenario: User submits incorrect username

Given user is on login page  
And user types correct username  
And user types some password  
And user clicks the login button  
Then page returns invalid username or password error

Scenario: User submits incorrect password

Given user is on login page  
And user types some username  
And user types correct password  
And user clicks the login button  
Then page returns invalid username or password error

Scenario: User submits password and no username

Given user is on login page  
And user types username  
And user clicks the login button  
Then page no specified username error

Scenario: User submits username and no password

Given user is on login page  
And user types password  
And user clicks the login button  
Then page no specified password error

2. **Feature:** The user inputs CRN, requested time, days (Monday, Tuesday, etc.) and sent to database

Scenario: Successful storing input from webpage to PHP file

Given there are no current schedule conflicts with inputted CRN



When I inputted 123456, 10:00-11:00, Monday (checked), Wednesday (checked), Friday (checked)  
 Then the variables (CRN, beg\_time, end\_time, days[i]) should have 123456, 10:00,11:00, [Monday Wednesday Friday] respectively

### 3. **Feature:** Finding Conflicts

Scenario: Data from PHP conflicted to time data in database

Given data from php

When there is a time conflict

Then time that is conflicted is stored in a PHP variable

Scenario: Data from PHP conflicted to room\_num in database

Given data from php

When CRN class room number conflicts to another CRN classroom number

Then room number that is conflicted is stored in a PHP variable

Scenario: Data from PHP conflicted to days in database?

Given data from php

When there is a day conflict (no time availability on that day)

Then days that are conflicted are stored in a PHP array

### 4. **Feature:** Conflict possibilities. User inputs CRN, beginning and end time, days of the week to test

Scenario: No conflicts with student schedules

Given database of student schedules

And course catalogue

When user inputs course number and testing time and day

Then new screen appears dictating no conflicts with the option of viewing the calendar grid of severities

And The time block specified will be outlined and shaded white to dictate no severity

Scenario: Conflict with senior's schedule, not needed for graduation

Given database of student schedules

And course catalogue

And catalogue year

When user inputs course number and testing time and day

Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red

And The time block specified will be outlined and shaded yellow to dictate a low severity

Scenario: Conflict with senior's schedule, needed for graduation, offered next semester student will attend school

Given database of student schedules  
And course catalogue  
And catalogue year  
And graduation date  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded orange to dictate a high severity

Scenario: Conflict with senior's schedule, needed for graduation, not offered next semester student will attend school

Given database of student schedules  
And course catalogue  
And catalogue year  
And graduation date  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded red to dictate the highest severity

Scenario: Conflict with junior's schedule, not needed for graduation

Given database of student schedules, course catalogue, and catalogue year  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red. The time block specified will be outlined and shaded yellow to dictate a low severity

Scenario: Conflict with junior's schedule, needed for graduation

Given database of student schedules  
And course catalogue  
And catalogue year  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded orange to dictate a high severity

Scenario: Conflict with sophomore's schedule, not needed for graduation

Given database of student schedules  
And course catalogue  
And catalogue year

When user inputs course number and testing time and day  
Then new screen appears with calendar grid with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded green to dictate a very low severity

Scenario: Conflict with sophomore's schedule, needed for graduation

Given database of student schedules  
And course catalogue  
And catalogue year  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded yellow to dictate a low severity

Scenario: Conflict with freshman's schedule, not needed for graduation

Given database of student schedules  
And course catalogue  
And catalogue year  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded green to dictate a very low severity

Scenario: Conflict with freshman's schedule, needed for graduation

Given database of student schedules  
And course catalogue  
And catalogue year  
When user inputs course number and testing time and day  
Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red  
And The time block specified will be outlined and shaded yellow to dictate a low severity

Scenario: Conflict with multiple students

Given database of student schedules  
And course catalogue  
And catalogue year  
And graduation date  
When user inputs course number and testing time and day

Then new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red

And The time block specified will be outlined and shaded the color of the highest severity amongst the students

## 5. Step Definitions implemented:

```
Given(/^there are no current schedule conflicts with inputted CRN$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
When(/^I inputted (\d+), (\d+) (\d+)\-(\d+) (\d+), Monday \(\checked\), Wednesday\(\checked\),
Friday \(\checked\)$/) do |arg1, arg2, arg3, arg4, arg5|
  pending # express the regexp above with the code you wish you had
end
```

```
Then(/^the variables \(\text{CRN}, \text{beg\_time}, \text{end\_time}, \text{days}[i]\) should have (\d+), (\d+) (\d+),(\d+)
(\d+), \[Monday Wednesday Friday\] respectively$/) do |arg1, arg2, arg3, arg4, arg5|
  pending # express the regexp above with the code you wish you had
end
```

```
Given(/^database of student schedules$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Given(/^course catalogue$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
When(/^user inputs course number and testing time and day$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Then(/^new screen appears dictating no conflicts with the option of viewing the calendar grid
of severities$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Then(/^The time block specified will be outlined and shaded white to dictate no severity$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Given(/^catalogue year$/) do
  pending # express the regexp above with the code you wish you had
end
```

end

Then(/^new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^The time block specified will be outlined and shaded yellow to dictate a low severity\$/) do

pending # express the regexp above with the code you wish you had  
end

Given(/^graduation date\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^The time block specified will be outlined and shaded orange to dictate a high severity\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^The time block specified will be outlined and shaded red to dictate the highest severity\$/) do

pending # express the regexp above with the code you wish you had  
end

Given(/^database of student schedules, course catalogue, and catalogue year\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^new screen appears with calendar grid, with every grid either white, green, yellow, orange, or red\.. The time block specified will be outlined and shaded yellow to dictate a low severity\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^new screen appears with calendar grid with every grid either white, green, yellow, orange, or red dictate a very low severity\$/) do

pending # express the regexp above with the code you wish you had  
end

Then(/^The time block specified will be outlined and shaded green to dictate a very low severity\$/) do

```
    pending # express the regexp above with the code you wish you had
end
```

```
Then(/^The time block specified will be outlined and shaded the color of the highest severity
amongst the students$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user is on login page$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types correct username$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types correct password$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user clicks the login button$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Then(/^the user logs into the system$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Then(/^scheduler view page should display$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types some password$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Then(/^page returns invalid username or password error$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types some username$/) do
```

```
    pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types username$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Then(/^page no specified username error$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Given(/^user types password$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Then(/^page no specified password error$/) do
  pending # express the regexp above with the code you wish you had
end
```

## 6. Results of implemented Cucumber

Feature: Connect to database

```
Scenario: User connects to DB # features/database_test.feature:3
  ["information_schema"]
  ["test"]
    Given I am connected to a database # features/step_definitions/todo_steps.rb:83
      #<MySQL::Result:0x007ff11448b798>
    Tables_in_valutation
```

Feature: The user should be able to log into the system.

```
Scenario: User submits correct username and password # features/login.feature:3
  Given user is on login page #
features/step_definitions/todo_steps.rb:88
  And user types correct username #
features/step_definitions/todo_steps.rb:92
  And user types correct password #
features/step_definitions/todo_steps.rb:97
  And user clicks the login button #
features/step_definitions/todo_steps.rb:102
  Then the user logs into the system #
features/step_definitions/todo_steps.rb:106
  TODO (Cucumber::Pending)
./features/step_definitions/todo_steps.rb:107:in `/^the user logs into the
system$/
features/login.feature:8:in `Then the user logs into the system'
  Then scheduler view page should display #
features/step_definitions/todo_steps.rb:110
```

```

Scenario: User submits incorrect username # features/login.feature:11
  Given user is on login page #
features/step_definitions/todo_steps.rb:88
  And user types correct username #
features/step_definitions/todo_steps.rb:92
  And user types some password #
features/step_definitions/todo_steps.rb:114
  TODO (Cucumber::Pending)
  ./features/step_definitions/todo_steps.rb:115:in `/^user types some password$/'
  features/login.feature:14:in `And user types some password'
  And user clicks the login button #
features/step_definitions/todo_steps.rb:102
  Then page returns invalid username or password error #
features/step_definitions/todo_steps.rb:117

Scenario: User submits incorrect password # features/login.feature:18
  Given user is on login page #
features/step_definitions/todo_steps.rb:88
  And user types some username #
features/step_definitions/todo_steps.rb:121
  TODO (Cucumber::Pending)
  ./features/step_definitions/todo_steps.rb:122:in `/^user types some username$/'
  features/login.feature:20:in `And user types some username'
  And user types correct password #
features/step_definitions/todo_steps.rb:97
  And user clicks the login button #
features/step_definitions/todo_steps.rb:102
  Then page returns invalid username or password error #
features/step_definitions/todo_steps.rb:117

Scenario: User submits password and no username # features/login.feature:25
  Given user is on login page #
features/step_definitions/todo_steps.rb:88
  And user types username #
features/step_definitions/todo_steps.rb:125
  TODO (Cucumber::Pending)
  ./features/step_definitions/todo_steps.rb:126:in `/^user types username$/'
  features/login.feature:27:in `And user types username'
  And user clicks the login button #
features/step_definitions/todo_steps.rb:102
  Then page no specified username error #
features/step_definitions/todo_steps.rb:129

Scenario: User submits username and no password # features/login.feature:31
  Given user is on login page #
features/step_definitions/todo_steps.rb:88
  And user types password #
features/step_definitions/todo_steps.rb:133
  TODO (Cucumber::Pending)
  ./features/step_definitions/todo_steps.rb:134:in `/^user types password$/'

```



```

features/login.feature:33:in `And user types password'
And user clicks the login button          #
features/step_definitions/todo_steps.rb:102
Then page no specified password error      #
features/step_definitions/todo_steps.rb:137

6 scenarios (5 pending, 1 passed)
25 steps (10 skipped, 5 pending, 10 passed)
0m0.409s

```

## B. Module Testing

### 1. Sign-in-Testing

Tests sign-in page if the sign-in data is submitted to the PHP files

### 2. CRN-sending-Testing

Tests if the CRN data got sent to the PHP files

### 3. Color-Coded-Testing

Tests the FullCalendar for appropriate color of certain conflicts displayed on it.

The testing fails if the Calendar has a wrong color for a certain conflict.

## C. Integration Testing

### 1. Log-in Testing

Tests the login PHP files for logging in attempts. If the user enters the correct username and password, then he/she is taken to the index page. If the user enters the incorrect username or password, then the page would display “Incorrect username or password” and the number of attempts would decrement.

### 2. Data-Input-Validation-Testing

Tests the PHP files that connect to database for data validation (CRN) in the our current database. If the test can find the section with the CRN, then it would print “OK”. If the test finds the CRN even though it’s not in the table, then these PHP files would fail the test.

### 3. Conflict-Search-Testing

#### Time-Conflict-Search

Tests the conflict searching PHP file for time-conflict findings. With the inputted CRN, time, and days data, if the PHP file successfully finds the time conflict with another CRN in the database on the same day, then this file passed. This list of students and their class is what we want the output to be. Since it works and passed the test, we can assume that this is a list of students and their classes conflicted with the time input.

1. CRN:10853|Course:CHEM113|Time:900-950|Name:Jesus Glenn|Classification:Freshman
2. CRN:10773|Course:ENGL111|Time:900-950|Name:Steven Roberts|Classification:Freshman
3. CRN:10677|Course:CORE115|Time:900-950|Name:Vincent Jackson|Classification:Sophomore
4. CRN:10542|Course:BIBL101|Time:900-950|Name:Joan Henderson|Classification:Freshman
5. CRN:10775|Course:ENGL111|Time:900-950|Name:Erin Berger|Classification:Freshman
6. CRN:10775|Course:ENGL111|Time:900-950|Name:Christine Osborn|Classification:Freshman

#### 4. Conflict-Calendar-Display-Testing

Tests if the conflict-storing PHP file displays all conflicts on the FullCalendar. It fails the test if not all conflicts are displayed.

### D. System Testing

System testing was done as a result of other testing. For example, part of the system testing includes black box testing so we used the cucumber code for this. We also have the project on each of our servers in case one crashes.

### E. Acceptance Testing

Although the front end hadn't been connected to the backend, we took our front end into Karen to see if it was to her liking and explained what the output would be had the two pieces been connected. Her feedback was that she would like to input the classroom number on the front end and know not only the students who have conflicts but also be able to see which course has the most conflicts. She also dictated that major, classification, and frequency of the course are her priorities. Her last request was that we include photos of the classrooms so she can see the setup of each room. Obviously our initial draft of the project was rejected. Before our next presentation, we will change the output to include the option of viewing which course has the most conflicts and, time permitting, include photos of classrooms that can be viewed.

### F. Performance Testing

This portion of testing has only been done on a portion of the data (about 1/10) from the csv file as we don't want to have to debug with a plethora of information. However, with the subset of data that we've worked with, conflicts were found in under a second. With such a fast return, it's difficult to get an exact time by hand but even if we were to assume that incorporating the entire csv file would increase the time tenfold, it'd still be within the 3-5 second range we originally planned for. If for

whatever reason, there's a colossal increase in time for results once we dump the entire csv, it would be safe to say that we need to rework our processes/function calls and maybe even re-analyze our database setup by going over our ERD Diagram and DFD's. Theoretically, this would be done by November 38th, giving us time to try to make it more efficient but recognizing that a functional product that is a little slow is better than a late product, or even worse, a lack of one.

## **G. GOMS Testing**

### **1. Keystroke Level Model**

**Tester: Our Team and one other user.**

Press a key or button

Best typist = .08 seconds

Good typist = .12 seconds

Average skilled typist = .20 seconds

Average non-secretary = .28 seconds

Typing random letters = .50 seconds

Typing complex codes = .75 seconds

Worst typist = 1.2 seconds

Point with a mouse (excluding click) = 1.1 seconds

Move hands to keyboard from mouse (or vice-versa) = .4 seconds

Mentally prepare = 1.35 seconds

### **2. Goals**

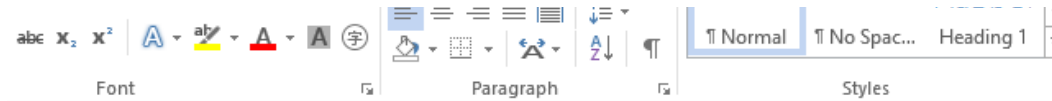
**a. Task:** User types username and password and clicks login button

- **Task groups:** User prepares his or herself, User moves, mouse types username and password

**b. Task:** User types in crn and enters a time

- **Task groups:** User prepares his or herself, User moves, mouse types username and password

### **3. Results of GOMS Tests (Users' Average)**



Task	Time	GOMS	Group of tasks	Sequence of tasks	GOMS task breakdown
1 login	4.50	4.71	enter username, enter password, click button	type in username, type in password, move mouse and click login button	$K*12 + MC + MM + MHKM + MP$
2 choose semester/year	4.48	5.55	click and scroll *2, click button	(move mouse to semester and year, click, move mouse, click to choose) *2, move mouse and click login	$MM*2 + MC*5 + MP$
3 choose class information to schedule	14.23	8.93	type, click and scroll, click, click and scroll, click button	type in course number/crn, move mouse and click to choose instructor and click, click days, click and scroll to choose time and click, click button to schedule	$K*6 + MC*8 + MHKM + MM*4 + MP$

The user will then enter the calendar. We are still undecided on how this will work (what the user will have the ability to do in this area). This means we have no idea how to quantify this or test it, yet.

Time totals

The results of the GOMS test shows us that our front-end isn't exactly perfect for users as it's taking them longer than projected to navigate through our website. This implies we will need to rework the front-end slightly to make it more user friendly.

## H. Status Reports

Since the request of a verification document, weekly status reports have been submitted by alternating team members. Prior to the verification document however, status reports were much more infrequent. Within the reports made are updates on the project. For example, last week's report described the updates to the front-end of this web-application and this week's report discusses the PHP progress to finding conflicts and the integration of that back-end to the front end.

## IV. GLOSSARY

**Acceptance Testing** - give the overall system to the user and see if it meets their expectations and/or desires

**API** - application programming interface; helps in building software applications

**Black Box Testing** - method of testing software without requiring the innards/specific code ie. cucumber

**Course** - is the class that a particular student is taking, but does not include the section

**CRN** - unique course number that also differentiates sections

**Cucumber** - behavior driven development; run acceptance tests to test the software

**Data Flow Diagram(DFD)** - graphical representation of how the data flows among the various pieces of the system

**ERD Diagram** - Entity-relationship diagram; describes the process within the database and the information within the database

**Feature** - a summary of what the system should be able to do and then follows are step by step descriptions of what should happen; first step in using cucumber

**GOMS Test** - Goals, Operators, Methods, and Selections; method of testing human interaction and usability

**Incrementality** - a principle of software engineering that encourages the development in pieces at a time

**Integration Testing** - begin combining modules and testing their connection

**LAMP** - acronym for a web stack made of 4 components; Linux operating system, Apache server, MySQL database, and PHP coding

**Module Testing** - testing chunks of the system's code

**Performance Testing** - test the system's speed and efficiency

**Scenario** - a situation to test when running cucumber as it's a potential outcome in the system

**Schema** - outline of database queries/table creation

**Sections** - student is enrolled in a section of a course which has a time and professor assigned to that section

**Sequence Diagram** - interaction diagram that demonstrates how processes interact with each other and the order those interactions occur

**Status Reports** - weekly reports on the team's progress in regards to the project completion

**Step Definition** - a step in utilizing cucumber written in Ruby; follows the feature and scenario declarations

**System Testing** - test the overall system, theoretically with all of the pieces

**UML Diagram** - Unified modeling diagram; visualization of the system's architecture

**Use Case Diagram** - representation of the interaction between the user and system

**WAMP** - same as LAMP, but for Windows instead of Linux

**Zaboo 2.0** - name of Team 3's system/product revamped

## **V. Bibliography**

Ghezzi, Carlo, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Englewood Cliffs, NJ: Prentice Hall, 1991. Print.

"Writing a Good Software Design Document." *BitFormation Consulting* -. N.p., n.d. Web. 06 Oct. 2014.