Executive Summary

At the University of Victoria (UVic), course registration is a difficult and stressful process for students. Although registration is somewhat straightforward for those individuals that manage to follow their curriculum throughout their degree, it can become an enormous headache for students who get behind in their courses due to illness, poor grades, or other reasons. The web applications provided by the University and third parties for registration have a number of problems that will be solved by the Graduate! Application, developed by Puzzle.

There is a website developed by a past student at UVic called ScheduleCourses.com which serves as a significant improvement to the University provided site. It allows students to select a list of courses, and then shift the course times selected around in order to make an ideal schedule. It provides a dynamic visualisation of your timetable which is very helpful for mitigating conflicting course times.

The most significant issue with ScheduleCourses.com is that there is no simple way to view how your selection of courses in a particular semester will affect your final goal of completing your degree program. Often, students will fail to register in a prerequisite during a particular term. This will cause them to be unable to take required courses in the future, putting them further behind in their degree. Similarly, courses are often offered in specific terms, and an unaware student may need to wait a year to take a particular course. Graduate! will take into account a user's program requirements and provide a means of visualizing how their selected registration will affect their final graduation date, and future schedule requirements. The Graduate! scheduling system will not only consider the student's program requirements and course scheduling constraints. It will taken into account a user's personal scheduling preferences as well.

This is a fairly large project which will require a lot of data entry regarding various program requirements at UVic. As such, Puzzle will develop a proof of concept that shows how Graduate! will work with the Software Engineering program at UVic. If this concept is satisfactory, it will then be possible to move forward with the other programs offered at the school. This document describes the user interaction features, conceptual design, technical design, management plan, and testing plan that will be employed by Puzzle when developing the Graduate! application.

S3a Technical Design Document

Graduate!

University Degree Planner

Created by: Puzzle

Authors:
Brendan Heal
Jonah Rankin
Ali Nobari
Spencer Mandrusiak

Table of Contents

Executive Summary 1

1.0 Project Summary 8

1.1 Important Features 8

1.2 Hardware 8

1.3 Performance 8

2.0 User Interaction 9

2.1 Application Views 9

2.1.1 Front Page 9

2.1.2 Login Page 9

2.1.3 Program Management Page 9

2.2 Use Case Diagram 11

2.3 User Use Cases 12

2.3.1 Registration 12

2.3.2 Login 14

2.3.3 Mark Courses Complete 15

2.3.4 Alternate - Mark Courses Complete 15

2.3.6 Move Courses 18

2.3.8 Setting Generation Preferences 21

2.4 Administrator Use Cases 22

2.4.1 Administrator Login 22

2.4.2 Disable User 23

2.4.3 Delete User 23

2.4.4 Create New Course 24

2.4.5 Edit Course 25

2.4.5 Delete Course 26

2.5 Glossary 28

3.0 Management Plan 28

3.1 Feature Breakdown 28

3.1.1 Course Scheduling Algorithm 28
3.1.2 Intuitive Interface Design 29
3.1.3 User Interface Layout 29
3.1.4 Data Collection/Scraping 29
3.1.5 Database Creation 30
3.1.6 Netlink Integration 30
3.1.7 User Authentication 30
3.2 Possible Implementations 30
3.2.1 Deployment 30
3.2.2 Persistent Storage 30
3.2.3 Server-side Technologies 31
3.2.4 Client-side Technologies 31
3.3 Minimal System 31
3.3.1 Summary 32
3.4 Team Structure and Organization 32
3.4.1 Web Application Development - Brendan Heal 32
3.4.2 Database Creation - Ali Nobari 33
3.4.3 User Interface Development - Jonah Rankin 33
3.4.4 Course Scheduling Algorithm - Spencer Mandrusiak 33
4.0 Conceptual Design 33
4.1 Application Flow Diagram 33
4.2 Activity Diagram 34
4.3 Program Management Page Statechart Diagram 36
4.4 Use Case Create-Read-Update-Delete Matrix 37
4.5.1 Model-View-Controller 38
4.5.2 Observer Pattern 38
4.5.3 Façade Pattern 39
4.5.4 Singleton Pattern 39
5.0 Testing 1
5.1 Introduction 1
5.2 Test Automation and Integration 1
5.3 Test Driven Development 2
5.4 Test Cases 2

5.4.1 Test Case 1: Registration 2
5.4.1.1 Description 2
5.4.1.2 Assumptions 2
5.4.1.3 Pre Conditions 2
5.4.1.4 Event Flow 2
5.4.2 Test Case 2a: Program Requirements (Registration) 3
5.4.2.1 Description 3
5.4.2.2 Assumptions 3
5.4.2.3 Pre Conditions 3
5.4.2.4 Event Flow 3
5.4.3 Test Case 2b: Program Requirements (Account Preferences) 3
5.4.3.1 Description 3
5.4.3.2 Assumptions 3
5.4.3.3 Pre Conditions 3
5.4.3.4 Event Flow 4
5.4.4 Test Case 3a: Schedule Customization (Course Move) 4
5.4.4.1 Description 4
5.4.4.2 Assumptions 4
5.4.4.3 Pre Conditions 4
5.4.4.4 Event Flow 4
5.4.5 Test Case 3b: Schedule Customization (Course Removal) 5
5.4.5.1 Description 5
5.4.5.2 Assumptions 5
5.4.5.4 Pre Conditions 5
5.4.5.4 Event Flow 5
5.4.6 Test Case 4a: User-caused Conflict (Prerequisite Violation) 5
5.4.6.1 Description 5
5.4.6.2 Assumptions 6
5.4.6.3 Pre Conditions 6
5.4.6.4 Event Flow 6
5.4.7 Test Case 4b: User-caused Conflict (Term Offering Violation) 7
5.4.7.1 Description 7
5.4.7.2 Assumptions 7

5.4.7.3 Pre Conditions 7
5.4.7.4 Event Flow 7
5.4.8 Test Case 4c: User-caused Conflict (Enrollment Limit Violation) 7
5.4.8.1 Description 7
5.4.8.2 Assumptions 8
5.4.8.3 Pre Conditions 8
5.4.8.4 Event Flow 8
5.4.9 Test Case 5a: Schedule Fitting Algorithm (Course with Prerequisites) 8
5.4.9.1 Description 8
5.4.9.2 Assumptions 8
5.4.9.3 Pre Conditions 8
5.4.9.4 Event Flow 8
5.4.10 Test Case 5b: Schedule Fitting Algorithm (Courses that are Prerequisites) 9
5.4.10.1 Description 9
5.4.10.2 Assumptions 9
5.4.10.3 Pre Conditions 9
5.4.10.4 Event Flow 9
5.4.11 Test Case 5c: Schedule Fitting Algorithm (Course offered once per year) 9
5.4.11.1 Description 9
5.4.11.2 Assumptions 9
5.4.11.3 Pre Conditions 9
5.4.11.4 Event Flow 9
5.4.12 Test Case 5d: Schedule Fitting Algorithm (User preferences cannot be fulfilled) 10
5.4.12.1 Description 10
5.4.12.2 Assumptions 10
5.4.12.3 Pre Conditions 10
5.4.12.4 Event Flow 10
5.4.13 Test Case 5e: Schedule Fitting Algorithm (Fitting multiple courses) 11
5.4.13.1 Description 11
5.4.13.2 Assumptions 11
5.4.13.3 Pre Conditions 11
5.4.13.4 Event Flow 11
5.4.14 Test Case 5f: Schedule Fitting Algorithm (Cannot satisfy program requirements) 11

5.4.14.1 Description 11
5.4.14.2 Assumptions 12
5.4.14.3 Pre Conditions 12
5.4.14.4 Event Flow 12
5.4.15 Test Case 5g: Schedule Fitting Algorithm (Fitting courses to an empty schedule) .. 12
5.4.15.1 Description 12
5.4.15.2 Assumptions 12
5.4.15.3 Pre Conditions 12
5.4.15.4 Event Flow 12
5.4.16 Test Case 6: Course Overrides 13
5.4.16.1 Description 13
5.4.16.2 Assumptions 13
5.4.16.3 Pre Conditions 13
5.4.16.4 Event Flow 13
5.4.17 Test Case 7: Saving User Data 13
5.4.17.1 Description 13
5.4.17.2 Assumptions 13
5.4.17.3 Pre Conditions 14
5.4.17.4 Event Flow 14

1.0 Project Summary

The Graduate! web application will help students at the University of Victoria plan their degree by providing students with tools to create custom tailored schedules, which accommodates each individual's specific needs. Many University programs have highly specific degree requirements and the University of Victoria is no different. Just Right Showers has expressed interest in commissioning a system that will assist UVic students by planning their entire degree. Puzzle's Graduate! system will help users by allowing them to customize certain aspects of the schedule, such as having no evening classes or only four classes in a semester, to accommodate for the busy lifestyle of a university student. This will give students more freedom when planning out their degree and also help them keep track of what they have or have not completed. Another way Graduate! benefits students is it will automatically regenerate a new schedule if a student marks that they have failed a course; therefore the student will not have to deal with the burden of rearranging their schedule if such an unfortunate event occurs.

1.1 Important Features

Graduate! has many important features such as:

- Generate a schedule for a specific degree/program. Users maintain one program schedule. To facilitate schedule experimentation, users will be able to undo changes. All changes a user makes are immediately saved.
- Allow users to customize aspects of the schedule such as number of courses per term, time preferences, and specify any work terms to avoid classes being scheduled during that time
- Make automatic adjustments to the schedule if a user makes alterations, changes their preferences, or fails a course
- Allow a user to manually enter desired courses such as when students have not declared their program
- Adheres to University restrictions on courses such as term offerings, pre and co-requisites, maximum credits per term, etc.
- Allow user to view, update and save the schedule to their profile, as well as make modifications at a later date

1.2 Hardware

Our software will run on all major operating systems including: Mac OS X, Linux, and Windows. Graduate! will be designed to run on any modern HTML5 compliant browser including: Chrome, Firefox, Safari, and Internet Explorer.

1.3 Performance

Graduate! is expected to be available at least 98% of the time in a year (approximately 18 days of down time) and leave users without access to their schedules for no more than 48 hours at a time. While Graduate! is not safety critical software, it is important that scheduling services be

available for users and that course registration will not be compromised because of a failure of the Graduate! system.

The course scheduling algorithm is expected to complete in less than 5 seconds, and during processing it must provide the user with feedback.

At Puzzle we value our users privacy and the final product will utilize industry standard encryption and security practices to protect user information.

2.0 User Interaction

2.1 Application Views

2.1.1 Front Page

The *Front Page* is the landing page for the application with options to register, login, and get more information about Graduate!.

2.1.2 Login Page

The *Login Page* contains a form for users to enter their Email and Password. The **Cancel** button returns the user to their previous page — most commonly the *Front Page*. The Login button proceeds with login validation. If the submitted credentials are valid the user is taken to the *Program Management Page*. The *Login Page* may alternately appear as a modal window over-top the previous view.

2.1.3 Program Management Page

The *Program Management Page* is the home view for the application. From this view a logged-in user will be able to:

- Review their schedule including completed classes, the current semester and future semesters.
- Add, modify, and remove courses
- Define program preferences
- Auto-generate schedules based on their preferences.

Program Managment Page

Course Management Menu

Program Requirements
Required Courses
Technical Electives
Natural Science Electives
Complimentary Studies
Co-op
Search...
Course Details
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus mollis sapien non nisi sodales pulvinar. Curabitur odio velit, porta sit amet lobortis sit amet, volutpat ut justo.
Fit Unscheduled Courses

Graduate!
Account Settings
Logout

2016	Summer 2016	Fall 2016	Spring 2017	Sum
unication and	Work Term 3	CSC 355 Digital Logic and Computer Organization	Work Term 4	SENG 426 Software Qua
ata Structures II		CSC 320 Foudations of Computer Science		SENG 440 Embedded Sy
nd Systems: I		CSC 360 Operating Systems		SENG 499 Technical Proj
gineering		CSC 370 Database Systems		ELEC 485 Pattern Recog
on		SENG 360 Security Engineering		SENG 474 Data Mining

Courses

Term

Figure 1: Mock-up of Program Management View

2.2 Use Case Diagram

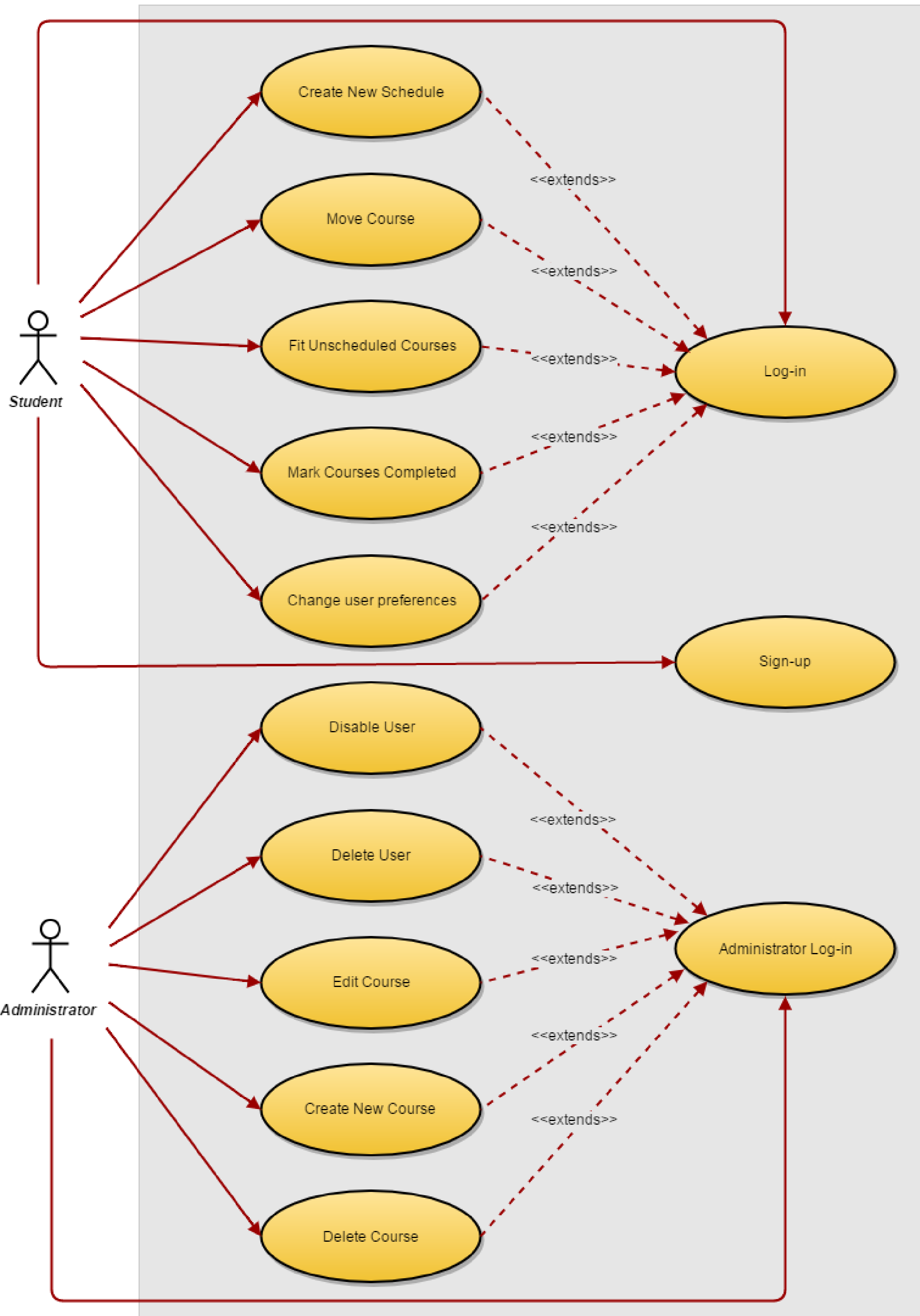

Figure 2: Use Case Diagram

2.3 User Use Cases

2.3.1 Registration

When a user first accesses the web application they arrive at the *Front Page*; users have the option to *Login* or *Register*. When the registration button is pressed the registration process begins:

1. The user enters their account information including: Email, Password, and University. After entering their information, the user presses the **Next** button to proceed to the second registration step.
2. The user specifies their course and program information. First, the user selects their program. Choosing a program populates the required courses list. Optionally, the program can be unspecified and all classes must be added manually. Second, the user adds other courses, as well as specifying electives.[1] Third, the user marks all the courses in the list that they have completed. After all program information has been entered, the user finalizes account registration by pressing the **Finish Registration** button.[2,3]
3. After finishing the registration process, the user is redirected to the *Program Management Page*.[4]

[1] It is not required for users to specify all electives to register. Initially electives will be specified as generic classes such as *Technical Elective* or *Complementary Study*. In order to schedule an elective, it must be a specific course. This can be done after the registration process.

[2] Users will have the option to modify program details and add or remove courses after registration.

[3] The course and program specification process may be simplified through integration with a user's UVic account. This would allow for automatic retrieval program information and completed courses.

[4] Account confirmation emails are not a high priority for initial prototypes. They will be implemented if time permits. If account confirmation is implemented, users will be sent to the login page and notified that they must check their email and confirm their account.

Error Handling

Incomplete Registration: If a user attempts to register for Graduate! without filling out all required sections of the form, they will not be able to register. A notification will pop-up informing the user that they have not filled out all required sections. They will continue to receive this notification until they have filled in the missing sections.

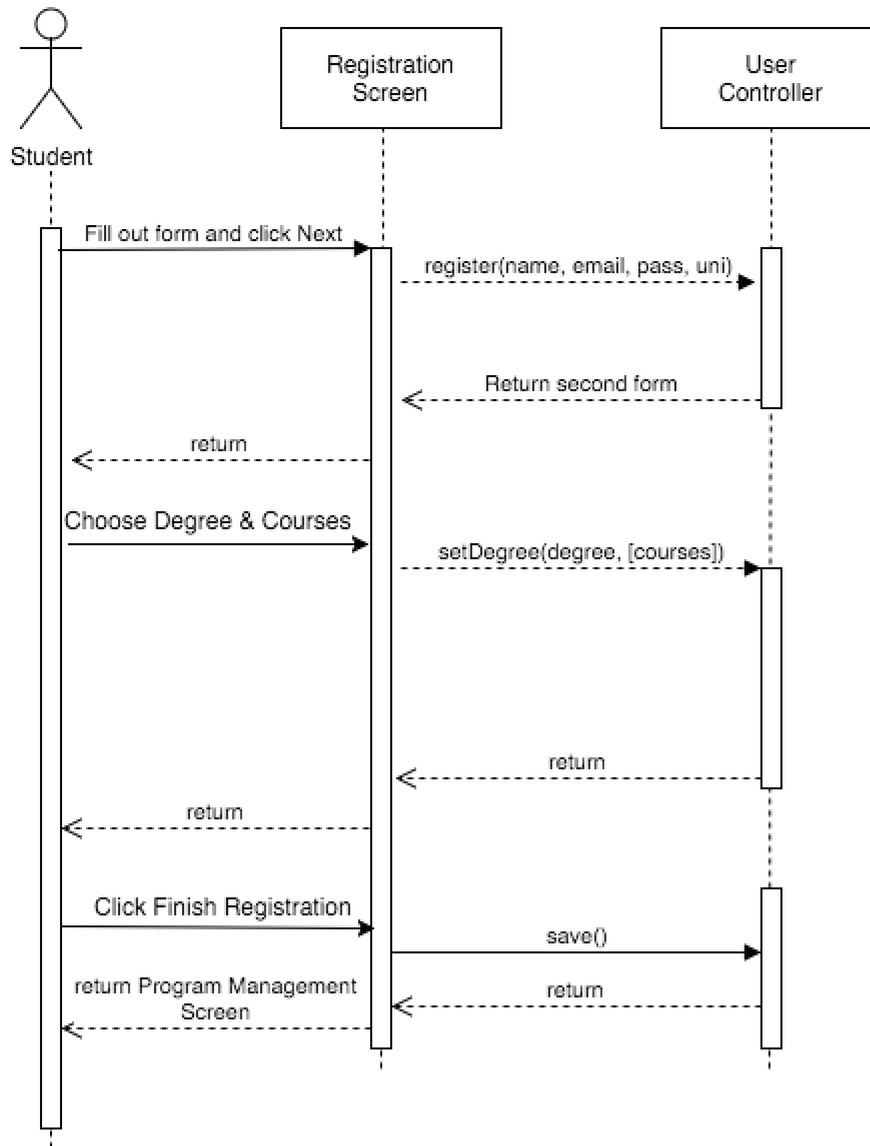

Figure 3: Registration Sequence Diagram

Pseudo Code

```
1: procedure REGISTER(name, email, password, university)
2:     if name == nil or email == nil or password == nil or university == nil then
3:         Notify the user a required field is missing
4:     else
5:         Push fields to database as new user
6:     end if
7:
8:     if program != nil then
9:         Pull courses from program to schedule from database
10:    end if
11:
12:    if elective entered then
13:        Add elective to schedule
14:    end if
15:
16:    if course_taken is pressed then
17:        Mark course.taken as true
18:    end if
19:
20:    if finish_registration is pressed then
21:        Push schedule to user account
22:        Redirect to Program Management Page
23:    end if
24: end procedure
```

2.3.2 Login

When arriving at the applications *Front Page*, pressing the **Login** button brings the user to the *Login Page*. On the *Login Page,* the user enters their *email* and *password.* They then press the **Login** button, which redirects them to the *Program Management Page*. Alternatively, on the *Login Page* the **Cancel** button returns the user to the *Front Page*.

Sample Interaction: Jane has just arrived at the *Front Page* of Graduate!. Since she already has an account, Jane selects the **Login** button, bringing her to the *Login Page.* Jane enters her email and password, followed by pressing the **Login** button. Once authorized, Jane is brought to the *Program Management Page*.

Error Handling

Incorrect Login: If a user attempts to login with an invalid username or password, they will be denied access to Graduate!. A notification will be displayed informing the user that the username or password they entered is invalid.

Cancel: If the user selected the login button and does not want to login, they may select **Cancel** to return to the front page.

Pseudo Code

```
1: procedure LOGIN(email, password)
2:     if email == nil or password == nil then
3:         Notify the user a field was not entered correctly.
4:     else
5:         User u=get_user(email)
6:         if u.password==password then
7:             Redirect(User Home)
8:         else
9:             Notify the user a field was not entered correctly.
```

2.3.3 Mark Courses Complete

Once registration has been successfully completed, Graduate! will automatically mark courses as completed based on the user's transcript. Graduate! will update completed courses each time the user logs in to the application.

Sample Interaction: Jay has already started using Graduate! and has a generated schedule. Because Jay has just finished his term and passed his courses, one of which is "SENG 321", he navigates to the *Program Management Page*. When Jay arrives at the page, he selects **View Schedule** and notices that Graduate! has already marked "SENG 321" as complete.

Pseudo Code

```
1: procedure MARK_COURSES(user, Program)
2:     user_courses=get_program_courses(Program)
3:     for course in user_courses do
4:         (course.taken=true)
5:     User u= getUser(user)
6:     u.setCourses(user_courses)
7:     Save()
```

2.3.4 Alternate - Mark Courses Complete

Upon registration, the user will be prompted to select all courses which they have already completed. Alternatively, the user may go to the *Program Management Page* and select **Add Completed Course**, where they will search for the appropriate course.

Once they have found the desired course, the user marks a checkbox indicating the course has been completed.

Sample Interaction: Jay has already started using Graduate! and has a generated schedule. Because Jay has just finished his term and passed his courses, one of which is “SENG 321”, he navigates to the *Program Management Page*. When Jay arrives at the page, he selects the **Add Completed Course** button, which displays a list of all courses relevant to Jay’s degree. Jay searches for “SENG 321” and selects the checkbox indicating he has completed the course.

Error Handling

Add incorrect course: If a user marks a course as complete that should not be completed, the user must simply uncheck the box, indicating the course is not completed.

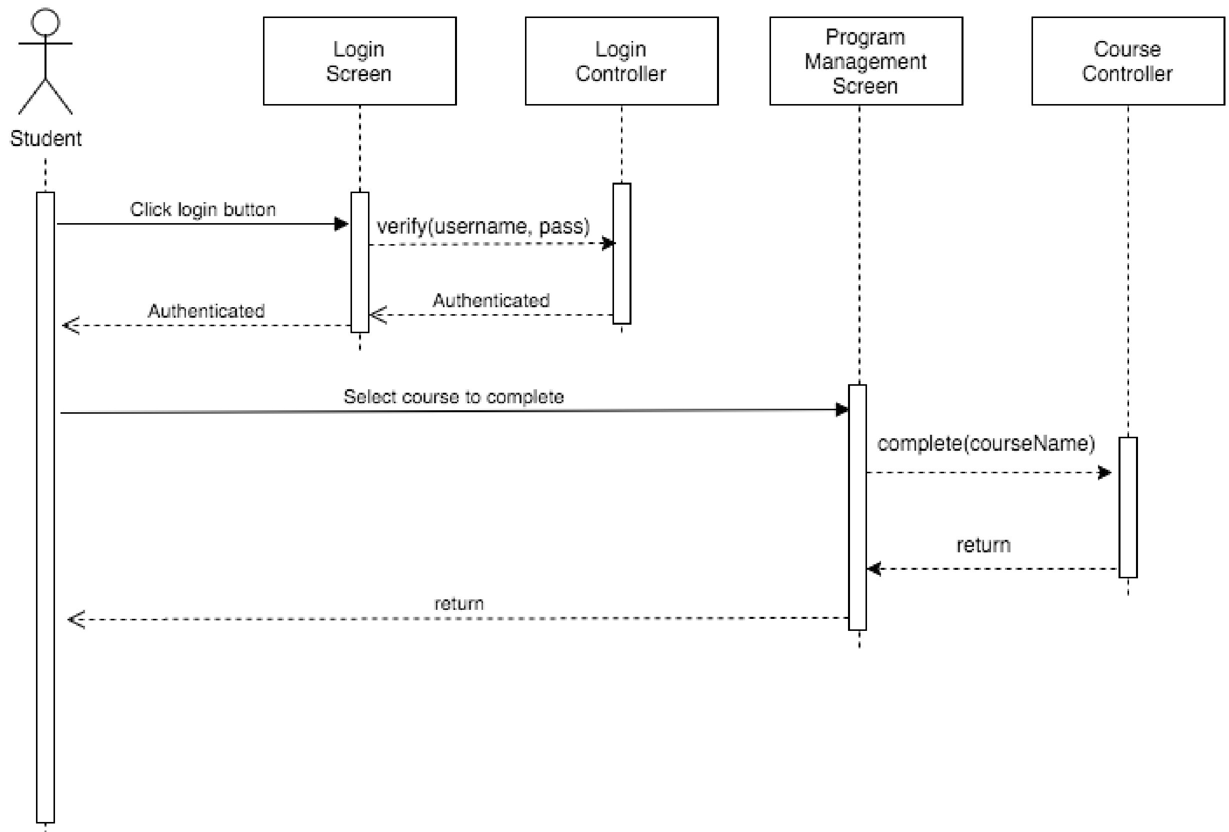

Figure 4: Mark Course Complete Sequence Diagram

Pseudo Code

```
1: procedure MARK_COURSES(user,marked_courses)
2:     (user_courses=get_courses(user))
3:     for course in user_courses do
4:         if course in marked_courses then
5:             (course.taken=true)
6:         else
7:             (course.taken=false)
8:     Save()
9:     Redirect(Schedule Management)
```

2.3.5 Add Courses to Schedule

A registered user will be able to add courses to their schedule. On the *Program Management Page*, the user opens the *Course Management Menu*. From the *Course Management Menu* the user can select courses as specified by their program requirements or by search. To add courses to a term the user drags the course from the *Course Management Menu* to the desired term on the *Program Management Page*.

Initially the *Program Management Page* will contain no courses. On screen prompts will direct first time users to the *Course Management Menu*. The prompts for first time users will introduce them to the basic functions of Graduate! including:

- The Course Management Menu
- Selecting electives
- How to add courses to a term
- How to move courses between terms
- How to use the Fit Unscheduled Courses option to automatically generate a schedule according to specific preferences

In the case where a student adds a course when its prerequisite is not scheduled[1], the system will display clear visual identification[2].

[1] Prerequisite conflicts may occur for courses that have multiple unmet prerequisites. In the case where the prerequisites do not apply to fulfilling degree requirements a notification will appear. For example: A computer science student wants to take POLI 410 which has the prerequisites POLI 310A and POLI 310B. A notification will inform the users that if they take POLI 410 the prerequisites may not be applicable to the completion of their degree.

[2] Clear identification may include a red tint, exclamation icon, and/or pop-up notification.

Sample Interaction: Jay has recently finished registering for Graduate!. When he arrives at the *Program Management Page*, after logging in, a pop-up window directs his attention to the Course Management Menu button, he clicks it and the Course Management Menu appears. Jay wants to add MATH 122 and CSC 225 to the same term as his friends so he drags them from the Menu to the Summer 2017 term.

Pseudo Code

```
1:  procedure ADD_COURSE(course,term)
2:      Course c = get_course(course)
3:      if !c.hasOffering(term) then
4:          Display that course is not offered in term
5:      else
6:          c.term=term
7:          if !user.schedule.CheckPrereqs() then
8:              Display unsatisfied prerequisite
9:          if !user.schedule.CheckPreferences() then
10:             Display unsatisfied preference
11:     Save()
```

```
1:  procedure PROGRAM MANAGEMENT PAGE
2:      Switch Function_Select:
3:      if Electives then
4:          Elective menu expand
5:      if Help then
6:          Help menu expand
7:      if User Preferences then
8:          Preference menu expand
9:      if Move Course then
10:         Move_Course(term,course)
11:     if Add Course then
12:         Add_Course(term,course)
13:     if Fit Schedule then
14:         Fit(schedule)
15:     Save()
```

2.3.6 Move Courses

On the *Program Management Page*, courses can be shifted between semesters by drag-and-drop. The user will be notified graphically if they are moving a course to an incorrect semester. This notification will appear for:

1. Missing prerequisites
2. Course not offered during the semester[1]

[1] Criteria 2 depends on the release of course schedules — usually 4-8 months ahead of their offering. If a course is moved beyond this date, the system will use the historical data to determine if the course is likely to be offered. A notification will also appear informing the user that offerings are subject to change.

Sample Interaction: Jane decides she no longer wants to take “CSC 225” in the summer semester. Thinking ahead, Jane determines the only other available semester for this course is in the fall. Once Jane arrives at the *Program Management Page,* she drags “CSC 225” from her summer semester schedule to her fall semester schedule. Graduate! verifies the course is available during this semester, and that Jane has the proper prerequisites. Upon success, Graduate! automatically saves the new schedule.

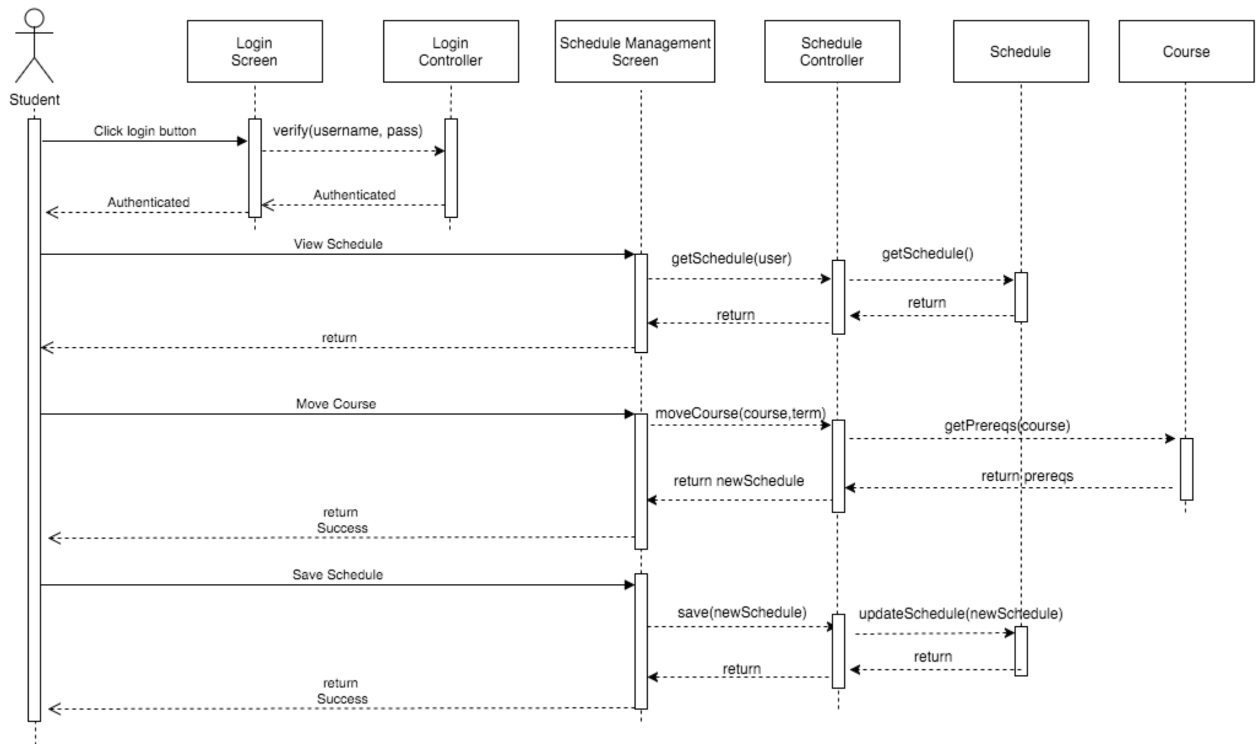

Figure 5: Move Courses Sequence Diagram

Pseudo Code

```
1: procedure MOVE_COURSE(course,term)
2:     Course c = get_course(course)
3:     if !c.hasOffering(term) then
4:         Display that course is not offered in term
5:     else
6:         c.term=term
7:         if !user.schedule.CheckPrereqs() then
8:             Display unsatisfied prerequisite
9:     Save()
```

2.3.7 Fit Unscheduled Courses

The schedule generation performed by Graduate! will fit existing unscheduled courses to the existing program schedule. To fit their unscheduled courses a user first navigates to the *Program Management Page*. In the Unscheduled Courses panel, the user clicks the **Fit Courses** button. A pop-over notification will inform the user that schedule generation is being performed. When complete, the notification will update allowing the user to undo the changes made. Fitted courses will be temporarily tinted[1] so that users can see the changes to their schedule.

In the case that a schedule cannot be generated, such as if the criteria is too restrictive, the user will be informed that only a partial match was achieved. The notification will detail the criteria which was and was not considered and users will have to ability to undo any changes made. As with successful generation, courses added to the schedule will be temporarily tinted[1].

[1] Tinted courses will exist while the schedule notification is visible. Once the user closes the notification all courses will appear normally.

Sample Interaction: Jane has recently completed all of the information required to create a schedule. There are a few courses which she knows she wants to take in the coming semester because her friends are also taking them. From the Unscheduled Courses panel she drags the courses she wants to take into her program schedule. For her remaining unscheduled courses Jane presses the Fit Courses button. A notification pop-up up saying her courses are being scheduled, in a few seconds the notification informs her that a match was found. She can easily observe the changes to her schedule by seeing the tinted courses. Jane likes her schedule, and she closes the notification.

Pseudo Code

```
1: procedure FIT_COURSES(schedule,user)
2:     temporary_schedule=user.schedule
3:     Notify user that schedule is being generated
4:     user.schedule=Fit(schedule,user)
5:     validate(schedule)
6:     Display course conflicts
7:     Save()
```

```
1: procedure UNDO
2:     user.schedule=temporary_schedule
3:     validate(schedule)
4:     Display course conflicts
5:     Save()
```

2.3.8 Setting Generation Preferences

When a user clicks the **Fit Unscheduled Courses** button a settings window will appear. On this window the user will be able to specify their target graduation date and their desired numbers of courses per term. Users will be able to prioritize their preferences; prioritization will assist the course generation algorithm in finding partial matches. After all settings are specified a user can generate their schedule by pressing the **Generate Button**. User's close the preferences window by pressing **Cancel**.

A user's preferences will be retained and reused the next time a schedule is generated.

Sample Interaction: Jay has been using Graduate! for some time now. Jay decides for his next semester he does not want more than four classes, so he navigates to the *Project Management Page* and opens the *Course Management Menu*. Jay presses **Fit Unscheduled Courses** button then sets the “Maximum Courses per Term” option to “4” and changes it to the top priority. He then clicks **Generate**, and a schedule with only four courses per term is generated.

Pseudo Code

```
1: procedure USER_PREFERENCES(user,prefs)
2:     for preference in prefs do
3:         if Validate(preference) then
4:             user.setPref(preference)
5:         else
6:             Display invalid preference notification
7:     Save()
```

2.4 Administrator Use Cases

2.4.1 Administrator Login

When arriving at the applications *Front Page*, pressing the **Login** button brings the administrator to the *Login Page*. On the *Login Page,* the administrator enters their *email* and *password.* They then press the **Login** button, which redirects them to the *Administrator Page*. Alternatively, on the *Login Page* the **Cancel** button returns the administrator to the *Front Page*.

Sample Interaction: Steve has just arrived at the *Front Page* of Graduate!. Since he already has an administrator account, Steve selects the **Login** button, bringing him to the *Login Page.* Steve enters his email and password, followed by pressing the **Login** button. Once authorized, Steve is brought to the *Administrator Page*.

Error Handling

Incorrect Login: If an administrator attempts to login with an invalid username or password, they will be denied access to Graduate!. A notification will be displayed informing the administrator that the username or password they entered is invalid.

Pseudo Code

```
1: procedure ADMIN_LOGIN(email, password)
2:     if email == nil or password == nil then
3:         Notify the user a field was not entered correctly.
4:     else
5:         Admin u=get_admin(email)
6:         if u.password==password then
7:             Redirect(Admin Home)
8:         else
9:             Notify the user a field was not entered correctly.
```

2.4.2 Disable User

Once the administrator has successfully logged in, they select **View Users**. This will display a list of all current users in the system. The administrator then scrolls through the list to find the username desired. Alternatively, the administrator may simply enter the desired username into the search box in the *View Users* section. Once the user has been found, the administrator selects the profile, followed by **Disable User**.

Sample Interaction: Steve is an administrator for Graduate!, therefore when he logs in, he has all options that an administrator has. Steve is directed by his boss to disable the user with the username “user123”. He selects **View Users**, and types “user123” into the search box. Steve selects the profile, and disables it by pressing **Disable User**.

Pseudo Code

```
1: procedure DISABLE_USER(email)
2:     if email == nil then
3:         Notify the user a field was not entered correctly.
4:     else
5:         User u=get_user(email)
6:         if u.exists() then
7:             u.disabled=true
```

2.4.3 Delete User

Once the administrator has successfully logged in, they select **View Users**. This will display a list of all current users in the system. The administrator then scrolls through the list to find the username desired. Alternatively, the administrator may simply enter the desired username into the search box in the *View Users* section. Once the user has

been found, the administrator selects the profile, followed by **Delete User**. They will be prompted with, “Are you sure?”. The administrator will select **Yes**.

Sample Interaction: Since Sarah is an administrator, when she logs in she has administrator options. Sarah’s boss has instructed her to delete the user with the username “userSmith”. She selects **View Users**, and enters “userSmith” into the search box. Sarah then selects the user profile, and selects **Delete User**. Graduate! prompts her with a notification saying, “Are you sure?”. Sarah selects **Yes**.

Pseudo Code

```
1: procedure DELETE_USER(email)
2:     if email == nil then
3:         Notify the user a field was not entered correctly.
4:     else
5:         User u=get_user(email)
6:         if u.exists() then
7:             UserController.delete(u)
```

2.4.4 Create New Course

After successfully logging in to Graduate!, an administrator will select **View Courses**. In the upper right hand corner, they will select **Add Course**. This will display a form for the administrator to fill out with fields such as *Course Name, Course Description, Prerequisites,* and *Co-requisites*. Upon completion of filling out the form, they will save the information by clicking **Finish & Save**. Alternatively, if the administrator does not want to save the filled in information, they will select **Cancel**.

Sample Interaction: Sarah discovers the course “SENG 321” has not yet been added to the Graduate! system. She first logs into her administrator account, and selects **View Courses**. Once the page loads, she selects **Add Course**. Sarah fills out the required information for the course and looks it over for any errors. Since she is satisfied with her work, she selects **Finish & Save**.

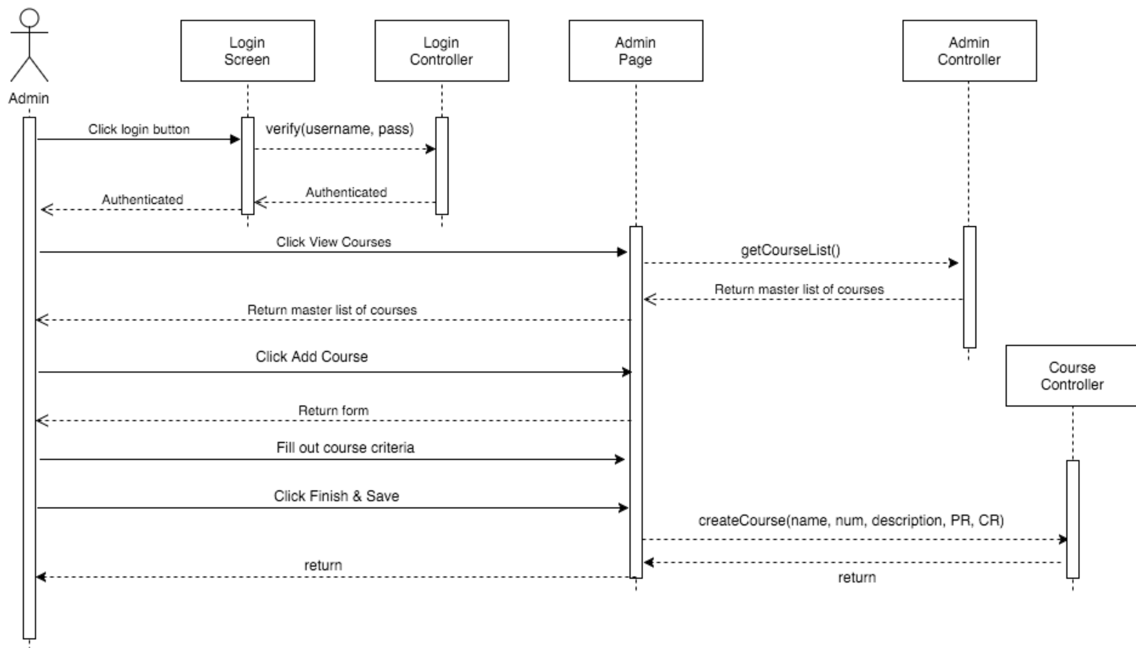

Figure 6: Create New Course Sequence Diagram

Pseudo Code

1: **procedure** ADD_COURSE($Name, Description, Prereqs, Coreqs$)
2: **if** $name ==$ nil **or** $Description ==$ nil **then**
3: Notify the user a field was not entered correctly.
4: **else**
5: Course $c =$ Course.new($Name, Description, Prereqs, Coreqs$)

2.4.5 Edit Course

Once the administrator has successfully logged in, they select **View Courses**. This will display a list of all courses in the Graduate! system. The administrator can either scroll through the list of courses until they find the desired course, or they can simply type the name of the course into the search box on the *View Courses* page. The administrator selects the appropriate course, followed by **Edit Course**. When the administrator is finished editing, they select **Save.**

Sample Interaction: Steve notices an error in the spelling of the course name “SNG 321”,so he logs into his account with administrator privileges. Steve selects **View Courses** and scrolls through the list until he finds “SNG 321”. He selects the course, followed by **Edit Course.** Steve then corrects the error and changes the course name to “SENG 321”. Now that Steve has finished his corrections, he finishes by selecting **Save.**

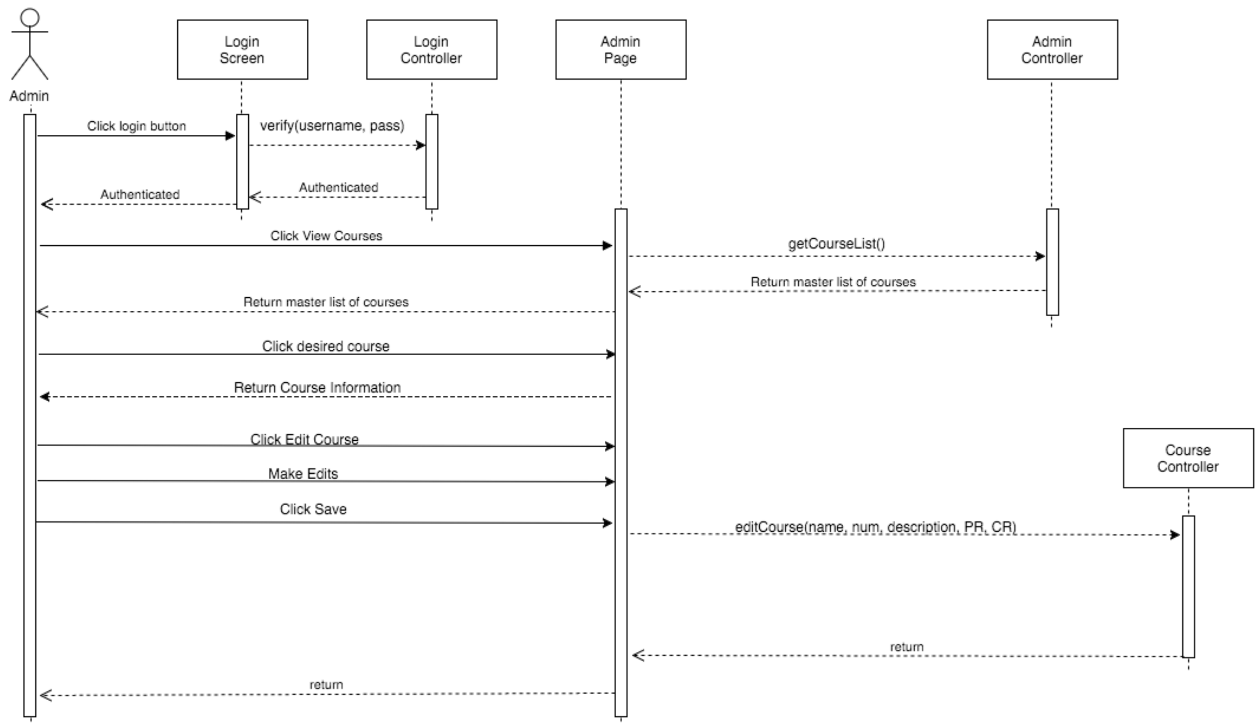

Figure 7: Edit Course Sequence Diagram

Pseudo Code

1: **procedure** EDIT_COURSE($Orig_Name$,$Name$, $Description$, $Prereqs$, $Coreqs$)
2: **if** $name$ == nil **or** $Description$ == ni l**or** $Orig_Name$ == nil **then**
3: Notify the user a field was not entered correctly.
4: **else**
5: Course c=get_Course($Orig_Name$)
6: **if** c.exists() **then**
7: $CourseController$.update(c,$Name$, $Description$, $Prereqs$, $Coreqs$)

2.4.5 Delete Course

Once the administrator has successfully logged in, they select **View Courses**. This will display a list of all courses in the Graduate! system. The administrator can either scroll through the list of courses until they find the desired course, or they can simply type the name of the course into the search box on the *View Courses* page. The administrator selects the appropriate course, followed by **Delete Course**. They will be prompted with, “Are you sure?”. The administrator will confirm by selecting **Yes.**

Sample Interaction: While going through the course list, Sarah notices that “ECON 103C” is still listed as an available course. Because Sarah knows this course is no

longer offered at the University of Victoria, she logs into her account with administrator privileges. She selects **View Courses** and types “ECON 103C” into the search box. Sarah then selects the course, and clicks **Delete Course**. Graduate! prompts her with a notification saying, “Are you sure?”. Sarah confirms by selecting **Yes**.

Error Handling

Deleting an active course: An error should warn the administrator that they are about to delete a course that is being used by students and is as well offered by the University. If the administrator sees fit, they can delete the course from the database.

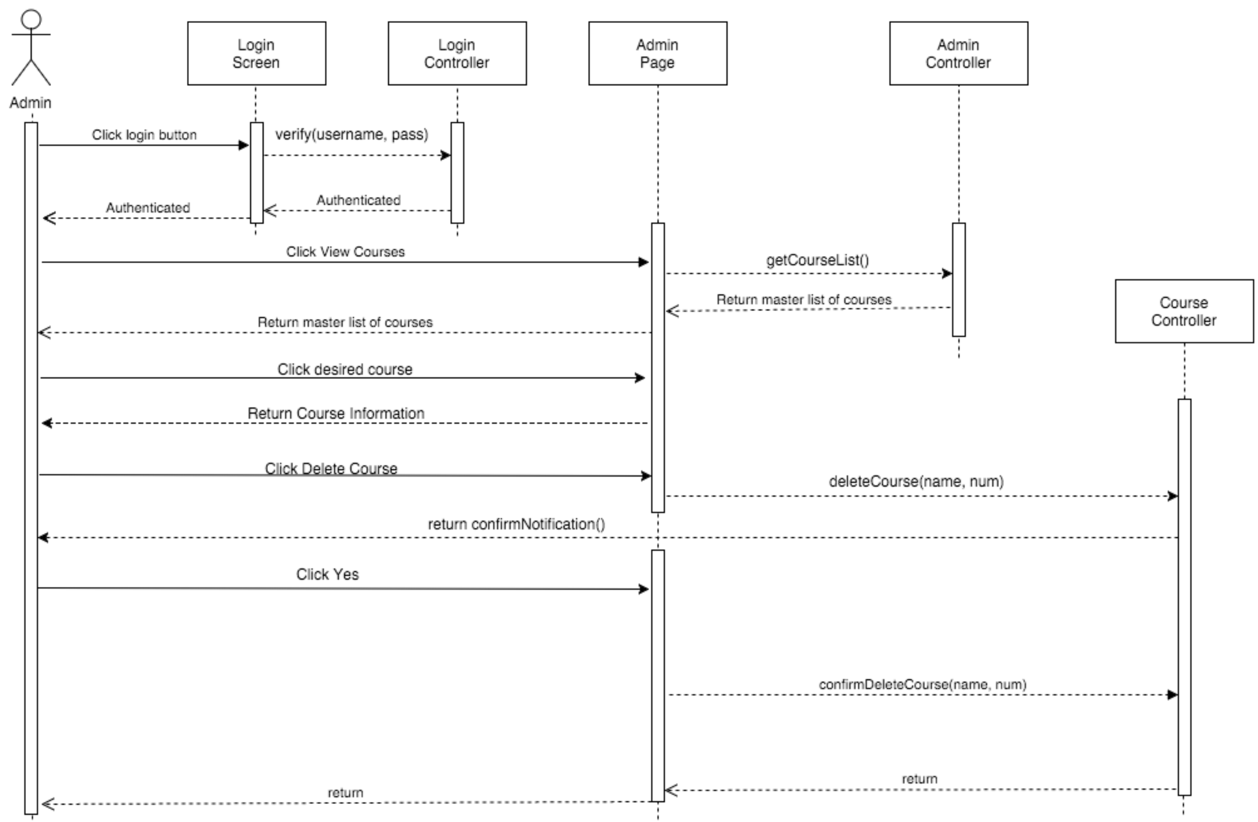

Figure 8: Delete Course Sequence Diagram

Pseudo Code

```
1: procedure DELETE_COURSE(name)
2:     if name == nil then
3:         Notify the user a field was not entered correctly.
4:     else
5:         Course c=get_Course(name)
6:         if c.exists() then
7:             CourseController.delete(c)
```

2.5 Glossary

Framework: A collection of software libraries that provide generic functionality that can be customized or extended. A framework provides a fast way to implement a commonly needed function.

Bootstrap: A HTML, CSS, JavaScript framework that facilitates fast and clean user interface design

Operating System: System software that manages computer hardware and software resources and provides common services for computer programs (reference: https://en.wikipedia.org/wiki/Operating_system)

Use Case: A list of steps defining the interaction between a role and a system, to achieve a goal (reference: https://en.wikipedia.org/wiki/Use_case)

Web Application: A client-server software application in which the client (or user interface) runs in a web browser (reference: https://en.wikipedia.org/wiki/Web_application)

3.0 Management Plan

3.1 Feature Breakdown

The core features of the Graduate! application are its course scheduling algorithm, intuitive user interface, user authentication, course and user data storage, course information collection, and Netlink integration.

3.1.1 Course Scheduling Algorithm

To obtain an “optimal” program schedule multi-stage processing will be performed. The first stage generates options that satisfy University constraints, course prerequisites and terms in which courses are available. The remaining stages generate schedules which satisfy the constraints provided by the user such as graduation data and courses per term. User specified constraints will be processed in the order specified. Users will be able to

prioritize their preferences (refer to Change User Preferences use case). If the algorithm cannot satisfy all the provided constraints the system will provide a schedule satisfying their highest priority constraints (the user will be notified that some preferences could not be satisfied). If there would be multiple schedules satisfying all constraints the first schedule found will be returned (If all preferences are satisfied the algorithm will determine that the schedule is “optimal” and in the interest of performance the result will be returned immediately). In the case that two outcomes are equally favoured the system will refer to the University Program Course Schedule to decide upon an optimal schedule.

3.1.1.1 Implementation Details

A vertex graph will be constructed using a table of the courses that need to be scheduled, the courses that are already scheduled, and their prerequisites. By traversing this graph in topological order a list of valid course schedules will be created in such that no course is taken before its prerequisite. Next, the list will be processed such that courses are available only in the terms in which they are offered, while still maintaining the prerequisite constraint. Variations of the schedules will be examined ensuring that user preferences, as prioritized, are satisfied. If a schedule is generated that satisfies all preferences it will be returned. If no schedule satisfies all preferences the system will backtrack until a schedule is found which satisfies the user's highest priority schedules.

3.1.2 Intuitive Interface Design

The creation of an easy to learn and use interface is critical in the creation of this application. To achieve this goal, Graduate! will utilize modern web application features including drag-and-drop to modify schedules. The user with receive constant feedback regarding the state of the system and have to ability to undo any changes they make. If a user attempts to reschedule a course such that it violates a prerequisite there will be clear indication on screen that the action cannot be completed; the courses will revert to their previous valid position or show clearly that they are not scheduled at the right time. It will be possible for users to override course prerequisites so that department course accommodations can be input.

3.1.3 User Interface Layout

To maximize the screen area a side, or top, menu bar will provide the user with quick access to additional features such as the selection of new courses, the modification of schedule preferences, and changes to account information. Refinement of the user interface will occur through rapid prototyping utilizing styling framework tools provided by bootstrap.

3.1.4 Data Collection/Scraping

The data for both the courses and their pre-requisite/co-requisite can be collected from the UVic databases. The information collected also includes the term in which the courses are

offered in and the lab sections that come with it. The data is then transferred onto the personal server, where the course scheduler will be retrieving the data from.

3.1.5 Database Creation

A database will be created that stores all the relevant information for the Graduate! application. The database will need to store the collected program requirements in order to associate the courses based on prerequisites, corequisites, labs, electives and alternates. Similarly, the course offerings will be stored based on the particular semesters they are offered. User information and their saved schedule will also be stored in the database.

3.1.6 Netlink Integration

Using your Netlink account, the service is able to view the courses that a user has already taken, or is enrolled in. This way, it can better tailor the remainder of the degree scheduling to the user. After determining the courses the user has both taken and is enrolled in, those courses can be removed from the overall group of courses that are going to be used to create the schedule for the user. This removes redundancies in what courses the user has to take. As well, pre-requisites can be determined for what courses are going to be scheduled, in case the user took some courses out of the average ordering of the courses they were supposed to take.

3.1.7 User Authentication

Users will initially register on the site and provide some required information to the Graduate! application. They will then be able to login with their selected username and password. All sensitive user information will be stored in a safe, encrypted form. The login page will provide password recovery.

3.2 Possible Implementations

There are a number of technologies suitable for prototyping the Graduate! web application.

3.2.1 Deployment

The application will be deployed using Heroku (www.heroku.com). The service provides an excellent support for multiple web development frameworks and facilitates rapid prototyping. Heroku provides free-of-charge small-scale databases and simple deployment using Git.

3.2.2 Persistent Storage

Heroku provides integrated add-ons for a variety of database management systems. Given the experience of the development team, PostgreSQL or MongoDB will be used.

3.2.3 Server-side Technologies

The requirements for the back-end of the Graduate! web application can be fulfilled by Ruby on Rails, Django, or Node.js. All of the candidate frameworks provide the routing, templating, database object relational mapping, and REST api integrations which are critical to the development of Graduate!.

3.2.4 Client-side Technologies

To create an engaging user interface, the application will rely on a JavaScript library such as Angular, JQuery, or D3. Additionally, to rapidly develop a clean interface, Bootstrap will be employed.

Angular is an extensive front-end web development framework supporting templates and routing. Angular may not be necessary given the choice of Server-side technology.

Implementing a drag-and-drop interface is highly desirable; it will make moving courses between semesters more intuitive and make the application friendly for mobile users. There are several technologies for displaying interactive graphics including SVG (scalable vector graphics), HTML5 Canvas, and JQueryUI.

The use of SVG is particularly promising because they are highly dynamic and would provide the ability to display custom schedules as an interactive graph. To interact with SVG graphics JQuery can be used, but alternative libraries such as D3 (https://d3js.org/) may be better suited as it is specifically designed for the creation of interactive graphics.

3.3 Minimal System

This section describes the minimal system that will be provided by Puzzle by the end of the term. Puzzle is going to develop a proof of concept of the Graduate! application, that shows how it will work for a student taking the Software Engineering program at the University of Victoria. Once the proof of concept is complete, adding other programs should be a simple matter of data entry. As there is only a month of development time available, it may not be possible to develop a fully functioning scheduling application by the project deadline. As such, Puzzle will focus on implementing solutions to the major weaknesses that are present in the current systems.

ScheduleCourses.com shows how a timetable builder system can be developed for a given semester, so our minimal system will not include such a feature. Rather, our program will leverage the program requirements provided in order to develop a plan for which courses will be taken in which semester, without the specific times being evaluated. This decision was made in part because it is difficult to develop and is already available, and because at this time, the University of Victoria only provides specific times for courses at most eight months in advance. This does mean that our suggested schedule will occasionally result in a course conflict, but at this time there is not a way to avoid this.

Our program will focus on providing students with a simple way to manage their program and

view how their scheduling choices affect their course load and final graduation date. It will make it easy for students to fit electives in their schedule, and provide a list of possible electives available that can be taken in a given semester. This allows students to work out a schedule that gives them the electives they desire, as opposed to taking whichever elective happens to fit their schedule. The University provides information about which courses are offered in which terms. Our minimal system will also include an authentication system for users to register and login to the system. Each user will be able to select their program and maintain a saved version of their personalized schedule.

Netlink Integration will not be included in the minimal system. Rather than automatically updating based on a student's completed courses, users will manually select their completed courses.

3.3.1 Summary

Graduate!'s main focus is to make planning a student's degree easier and more customizable, while pertaining to the University's restrictions such as prerequisites and term offerings. In the initial proof of concept, only the Software Engineering program at the University of Victoria will be included. The minimal system provided will include a web application that supports user authentication and dynamic schedule creation and visualization based on course offerings provided by the University.

3.4 Team Structure and Organization

The various components of the project have been broken down into tasks that will be taken on by individual team members of our group. The tasks include everything necessary in order to deliver the minimal system, but will be carried out with the final system in mind.

3.4.1 Web Application Development - Brendan Heal

This task involves the design and development of the web application including the selection of the database, front-end and back-end frameworks, configuration and deployment.

Phase 1: System Design

The web application will be designed with the selected front and back-end frameworks in mind.

Phase 2: Development and Deployment

A basic user authentication will be developed and deployed early so that the remaining development can occur.

3.4.2 Database Creation - Ali Nobari

This task involves the collection of data from the University of Victoria and creation of a database for the web application.

Phase 1: Data Collection

Two important pieces of information will be collected from the University of Victoria's website:

1. The program requirements for the Software Engineering program.
2. The course offerings by term.

Ideally, a scraper will be implemented that can access information from UVic's calendar page.

Phase 2: Database Creation

A database will be created which allows comprehensive access to the data collected as well as user information and user schedules.

3.4.3 User Interface Development - Jonah Rankin

A user interface will be developed according to the client's requirements.

Phase 1: Prototyping

User interface prototypes will be developed and presented to the client. Upon satisfaction with the user interface proposed, phase 2 will begin.

Phase 2: Interface Implementation

The views will be implemented using the selected front-end framework.

3.4.4 Course Scheduling Algorithm - Spencer Mandrusiak

An algorithm will be implemented to solve the course schedule optimization problem.

Phase 1: Research

The course scheduling algorithm will be researched in order to ensure an optimal solution is selected.

Phase 2: Implementation

The course scheduling algorithm will be implemented and rigorously tested.

4.0 Conceptual Design

4.1 Application Flow Diagram

The application flow diagram is a representation of the various views that will be included in the system and the links between them. The grey boxes are tasks that can be achieved from the

Program Management page. This diagram is useful for understanding navigation through the Graduate! web application.

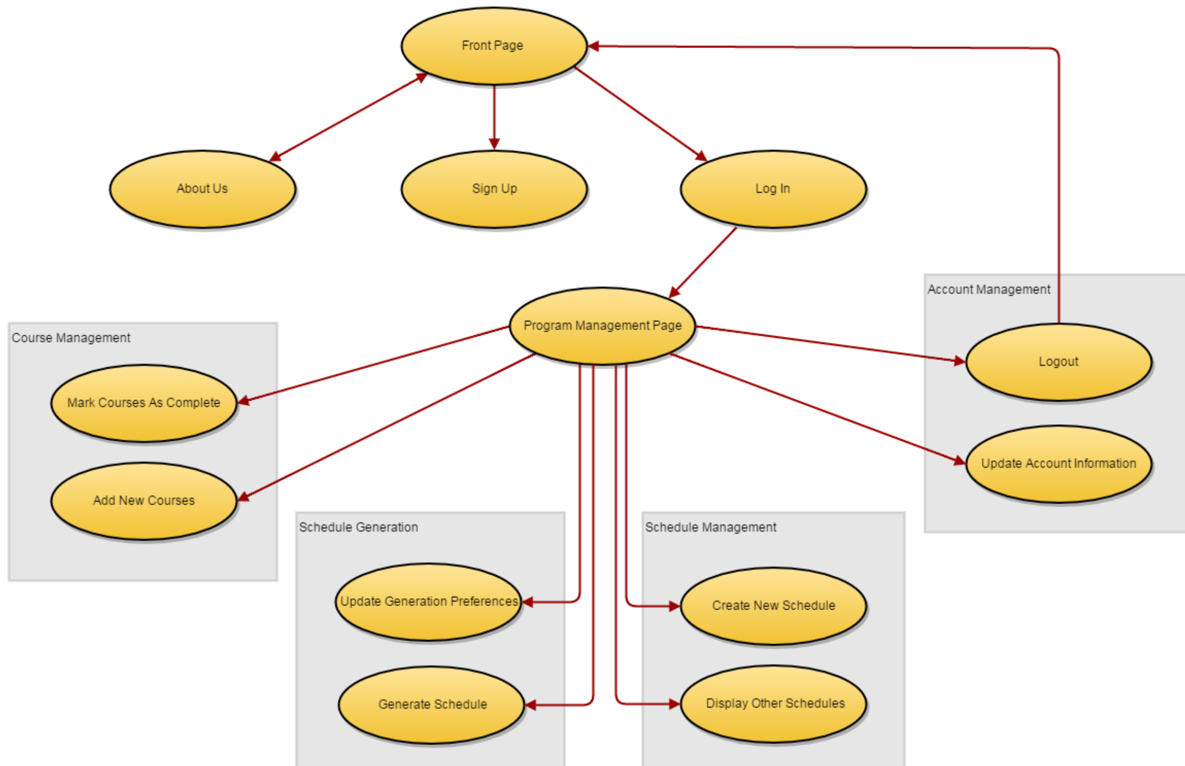

Figure 9: Application Flow Diagram

4.2 Activity Diagram

This activity diagram serves as a high level representation of workflow through the application. It describes the flow of various user actions throughout the Graduate! application. From the front page view, the user can access the Login, Signup and About page. Once a user has signed up with the site, they can login. Upon logging in successfully, the user will be redirected to their Program Management page. This page will contain the drag and drop interface for manipulating the user's schedule, and a collapsible side menu which provides tools for interacting with the schedule. From the program management view, the user can manage their course, and schedule settings. The collapsible side menu also contains the schedule generation function. This will generate a schedule tailored to the preferences specified by the user. The program management view includes a button to bring a user to the Account settings page, where they can update information about their account, such as their degree program in the event of a change.

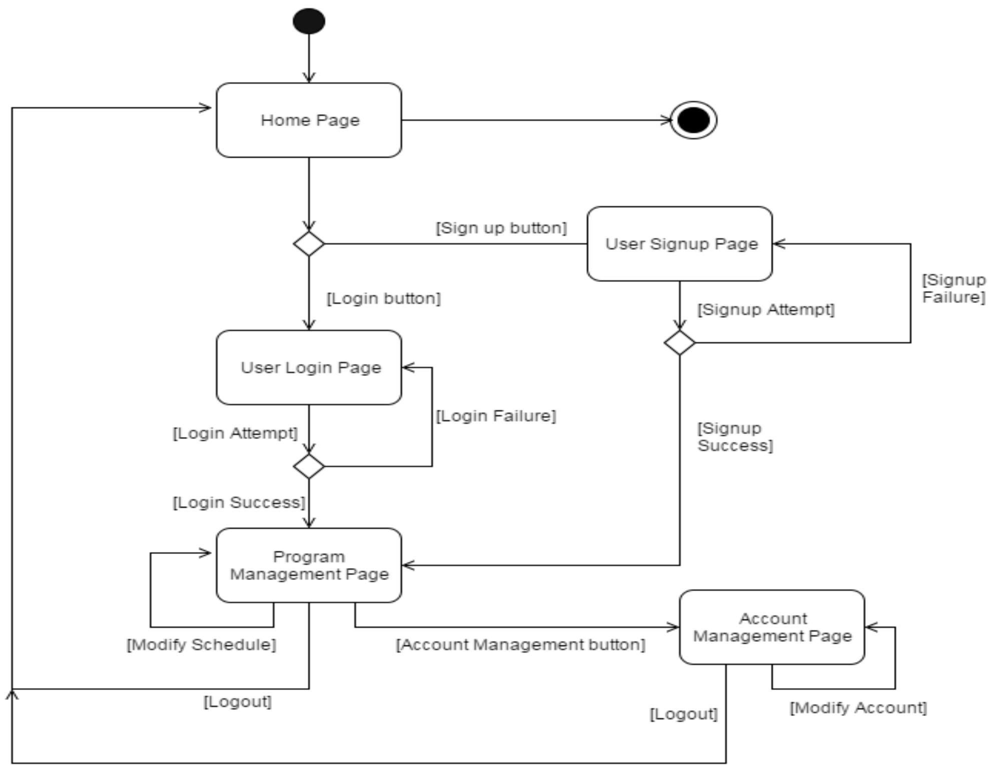

Figure 10: Activity Diagram

4.3 Program Management Page Statechart Diagram

This statechart diagram is a detailed description of the Program Management Page. It shows the behaviour of the page depending on the action that is performed by the user, while using the web application. It can be seen as a complete description of the Program Management Page's functionality.

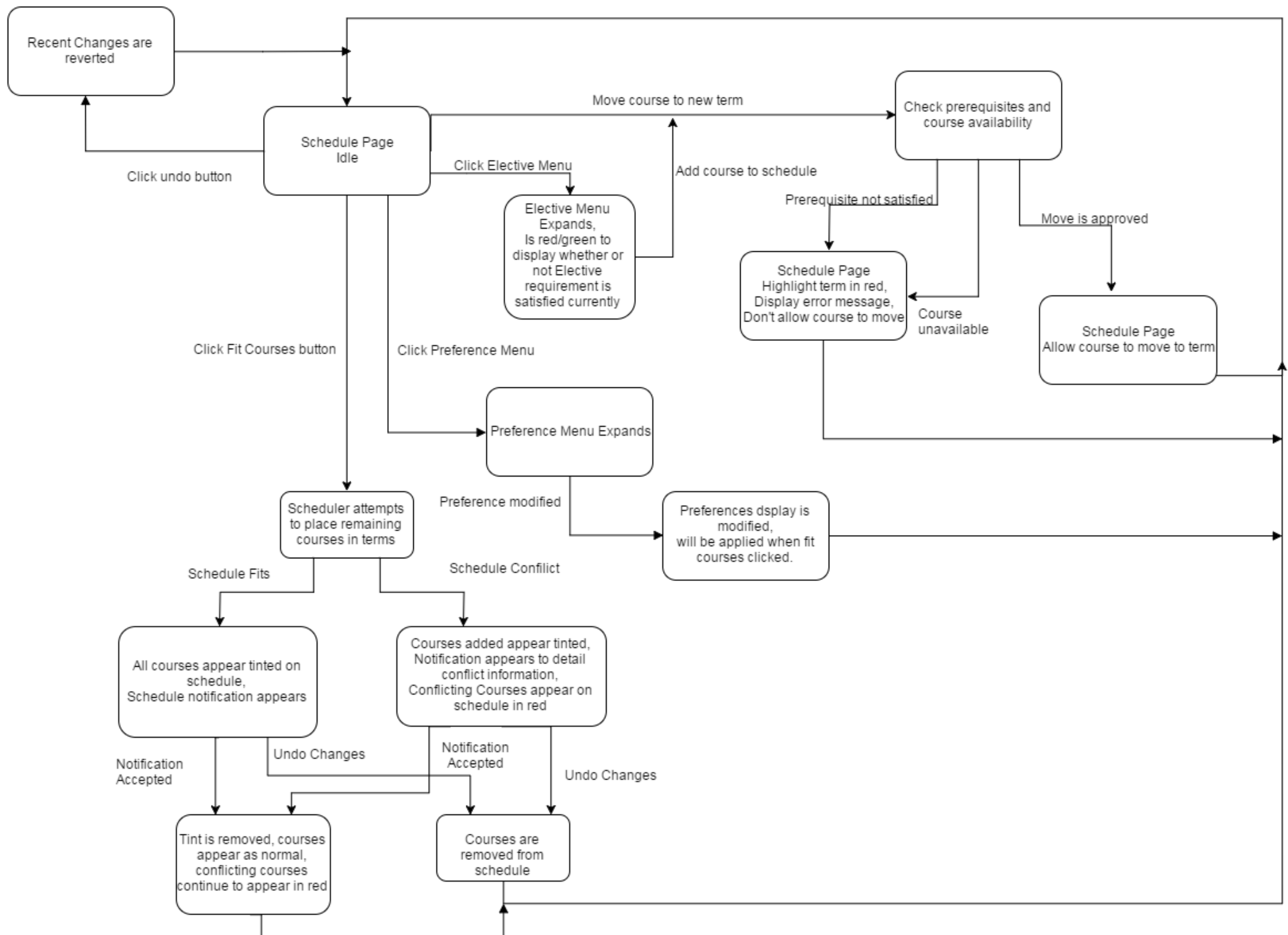

Figure 11: State Diagram for Program Management Page

4.4 Use Case Create-Read-Update-Delete Matrix

The figure below shows how the various user use cases interact with data objects either through Create, Read, Update or Delete (CRUD).

	User	UserPrefs	UserCourses	Term	Schedule
Set preferences	**R**	**RU**			
Move course	**R**		**UD**	**C U**	**U**
Override Prerequisite	**R**		**U**		
Fit Unscheduled Courses		**R**	**RU**	**C U**	**U**
Sign-up	**C**	**C**			**C**
Specify Program	**R**		**C U**	**C U**	**U**
Mark Course Complete	**R**		**U**		

C *- create,* ***R*** *- read,* ***U*** *- update,* ***D*** *- delete*

Table 1: CRUD Matrix

4.5 Class Diagram

The class diagram is a representation of the classes used in Graduate, and their inter-relationships, functions and attributes. In order to achieve the desired Model-View-Controller(MVC) architecture, a number of design patterns will be used.

4.5.1 Model-View-Controller

Model-view-controller(MVC) is a software architectural pattern that is commonly used when developing web applications. MVC facilitates the development of user interfaces by dividing the software application into three interconnected components. The first components are the *models,* which are strongly tied to the database and manage system data directly. Furthermore, a *view* is the output representation of the application, and defines the look and feel of the application by rendering information from the model. Finally, *controllers* are used to send commands to the views and models, often based on some user input.

In our class diagram, each model is listed along the bottom. There is one model for each database entities, and each model defines accessor methods for all the entities attributes. The following data entities will be stored in the system:

- ***Student*** : Student user information.
- ***Admin***: Administrator user information.
- ***Schedule:*** Schedule information. Each student user will have a schedule. Schedules will be composed of course offering selections made by the user and the schedule generation algorithm.
- ***Preferences***: User preference information, each student will have a set of preferences.
- ***Course***: Course information such as prerequisites and program requirements.
- ***Offering***: Course offerings. Each course is offered in one or many timeslots.
- ***Professor***: Course professor information.

The model facade is a facade class which acts as a simplified interface to the model. It is a singleton to ensure that concurrent changes to the same database element can not occur. The various controllers in our system will interact with this singleton object, which in turn saves itself to a database backend. There will be controllers for log-in, user registration, schedule management and generation, and course management. These controllers contain functions which are invoked when the user interacts with the website, and often involve either creating, reading, updating, or deleting a model object. For instance, the user signup page will invoke the registration controller to create new users.

4.5.2 Observer Pattern

The observer pattern defines two acting classes, the subject and the observer. The subject maintains a list of dependent observers, and notifies them when a state change occurs. In this implementation of MVC, all views will be observers, and the models will be subjects. Therefore,

when a model is changed, it will notify all dependent views of the change, and they will be updated accordingly.

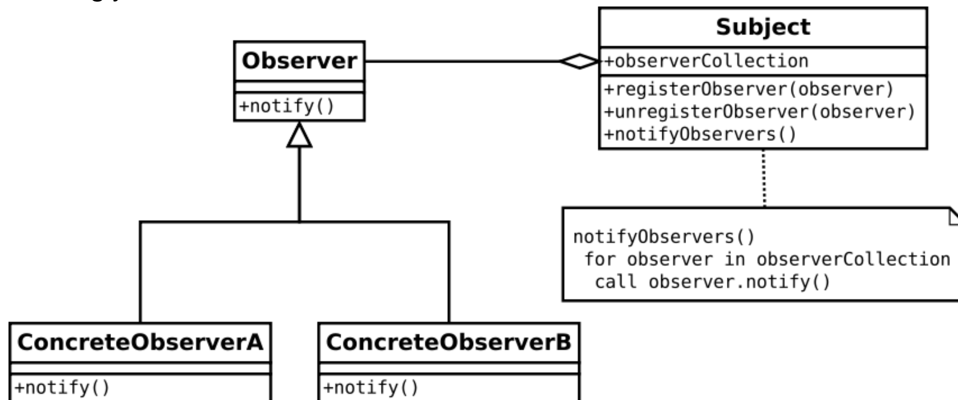

Figure 12 Observer Pattern Example

4.5.3 Façade Pattern

The façade pattern defines a façade object, which acts as a simplified interface to a large body of code. In order to facilitate the management of the various system models in our design, a model façade will be included which acts as an interface for accessing the models in the database. Therefore, the model façade contains a representation of the entire database. Controllers will make changes to the model facade, and these changes will be saved to the database.

4.5.4 Singleton Pattern

The singleton pattern ensures that prevents instantiation of a class. This ensures that only a single instance of a singleton class can exist. In order to ensure a consistent state for the model façade, it will be a singleton. This means that all users will interact with the same instance of the model façade.

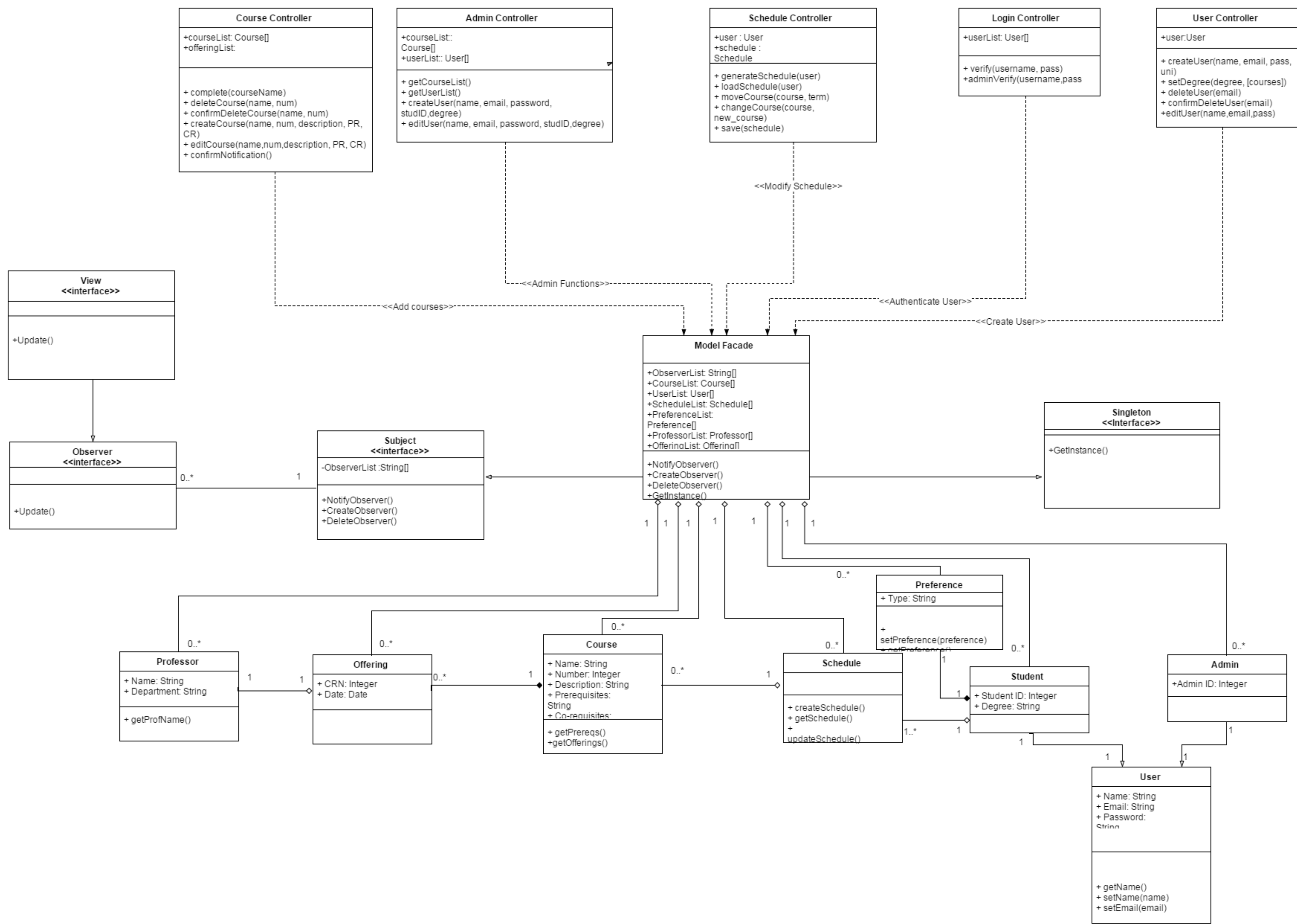

Figure 13 Class Diagram

5.0 Testing

5.1 Introduction

In order to ensure that Graduate! meets various user requirements and acceptance standards, a suite of tests will be developed for the system. These tests will be designed to cover all the functions of the system and are based on the test plan specified by the client, Just Right Showers. These tests will ensure that the system is functioning as expected and facilitate the detection of errors during development. The main features that will be tested are :

- **Registration**: Users can register and sign in to the Graduate! application.
- **Program Requirements**: The Graduate! application can access and process program requirements from the University of Victoria website.
- **Schedule Customization**: The Graduate! application allows a user to customize their schedule using a drag and drop interface.
- **Course Conflicts**: The Graduate! application detects course conflicts and scheduling problems and reports them to the user in an intuitive manner.
- **Schedule Fitting Algorithm**: The Graduate! application makes use of a schedule fitting algorithm that can generate an optimal schedule that meets program requirements, course offerings, and user preferences.
- **Course Overrides:** The Graduate! application allows users to specify overrides for course conflicts in the event of some department deployed override.
- **Saving User Data:** The Graduate! application dynamically saves a user's schedule and preferences to a database when they are modified.

5.2 Test Automation and Integration

As Graduate! is a web-based application, all of these tests can be automated using the Selenium Web Driver framework. In order to minimize the effort required to test the system, a suite of automated unit tests will be created and integrated into the build cycle of Graduate!. Each time the code base is updated, the suite of tests will be run and a test report will be generated that indicates the success or failure of each test case. This test report can and will be reviewed before the deployment of any given version of the Graduate! application. This will be done during the code review stage, where developers review each other's work before finally committing an update to the main codebase.

5.3 Test Driven Development

At Puzzle, we strongly believe in test-driven development. This means that before implementing a given feature, tests for the feature will be written. Each feature will be considered complete when the basic functionality is met, and all tests for the feature have been passed. This allows for concrete, incremental improvements to the application.

5.4 Test Cases

5.4.1 Test Case 1: Registration

5.4.1.1 Description

Test registration by creating a new account in the system. To ensure the account is successfully created, the user will logout and log back in with the newly created username and password.

5.4.1.2 Assumptions

The user does not attempt to create an account with a username already in the system.

5.4.1.3 Pre Conditions

User has navigated to the registration page and has the registration form displayed on screen.

5.4.1.4 Event Flow

Test Step	Input	Expected Result
User enters email, password, and university in registration form, followed by clicking **Next**	User email, password, and university User clicks **Next**	Second registration screen displayed with form to fill out program information, add additional courses, and indicate any completed courses
User selects type of degree, adds any additional courses, and selects at least one course complete. User selects **Finish Registration**	User degree, additional courses, and complete courses User clicks **Finish Registration**	User profile is saved and the user is redirected to the *Program Management Page*
User logs out of system	User selects **Log Out**	User is logged out and is redirected to the *Front Page*
User navigates to *Login Page* and logs in with newly registered username and password	User username and password	User logs in successfully and is redirected to the *Program Management Page* indicating registration was successful

Table 2: Registration Event Flow

5.4.2 Test Case 2a: Program Requirements (Registration)

5.4.2.1 Description

When a program is specified by the user, the “Required Courses” list in the Course Management Menu will be populated with the courses as specified in the most recent edition of the UVic course calendar.

5.4.2.2 Assumptions

The program is specified during registration process.

5.4.2.3 Pre Conditions

No program has been specified. The user has navigated to the registration page and it is displayed on screen. They have already submitted their email, password, and university as per step 1 of *Test Case 1* and pressed the **Next** button.

5.4.2.4 Event Flow

Test Step	Input	Expected Result
User selects type of degree and then clicks **Finish Registration**.	“Software Engineering” degree selected from the degree pull down menu then presses **Finish Registration**.	User arrives at the *Program Management Page*.
From the *Program Management Page*, the user opens the **Course Management Menu**.	The user opens the **Course Management Menu** by clicking the menu button ≡ on the *Program Management Page*.	Under the *Required Courses* list item the courses that appear correspond to those in the latest edition of the UVic Course Calendar for the type of degree selected.

Table 3: Program Requirements (Registration) Event Flow

5.4.3 Test Case 2b: Program Requirements (Account Preferences)

5.4.3.1 Description

When a program is specified, the “Required Courses” list in the Course Management Menu is populated with the courses specified in the most recent edition of the UVic course calendar.

5.4.3.2 Assumptions

The program is specified through the user’s account settings. In this test case the user already has specified their program. They change the program through their account settings.

5.4.3.3 Pre Conditions

The user’s account is already created. The user’s program has already been specified. The user is on the *Program Management Page*.

5.4.3.4 Event Flow

Test Step	Input	Expected Result
User opens their *Account Settings*.	Click **Account Settings** link in the navigation bar.	The *Account Settings* window appears.
From the *Program* pull down menu the user selects a new Program.	Click the *Program* pull down menu to open it. Select a new program. Press **Ok** on the *Account Settings* window to save and close the window.	The program is changed and the *Account Settings* window is closed.
User navigates to **Course Management Menu**.	The user opens the **Course Management Menu** by clicking the menu button ≡ on the *Program Management Page*.	Under the *Required Courses* list item the courses that appear correspond to those in the latest edition of the UVic Course Calendar for the type of degree selected.

Table 4: Program Requirements (Account Preferences) Event Flow

5.4.4 Test Case 3a: Schedule Customization (Course Move)

5.4.4.1 Description

Test the customization features of the schedule page and the dynamic saving of the user's modifications to the schedule. The action of moving a course to a different term will be tested.

5.4.4.2 Assumptions

A user is created on the system that has selected their degree program.

5.4.4.3 Pre Conditions

The user's account is already created. The user's program has already been specified. User is logged in and viewing the Program Management page.

5.4.4.4 Event Flow

Test Step	Input	Expected Result
User moves a course from one term to another valid term.	User clicks course, holds mouse button, and drags course to another term that is not restricted by prerequisites, course offerings, etc.	The course fits into its position in the term column and stays there. The system displays a "Saving…" dialog to the user.
The system saves the user's profile upon displaying the "Saved" dialog.		The schedule is saved.
The user refreshes the page.	User clicks refresh on browser.	The schedule is displayed with the user's modification to the course

		term included, indicating that is has been successfully saved in the system.

Table 5: Schedule Customization (Course Move) Event Flow

5.4.5 Test Case 3b: Schedule Customization (Course Removal)

5.4.5.1 Description

Test the customization features of the schedule page and the dynamic saving of the user's modifications to the schedule. The removal of a course from the schedule will be tested.

5.4.5.2 Assumptions

A user is created on the system that has selected their degree program.

5.4.5.4 Pre Conditions

The user's account is already created. The user's program has already been specified. User is logged in and viewing the Program Management page.

5.4.5.4 Event Flow

Test Step	Input	Expected Result
User clicks and begins dragging a course	User clicks course, holds mouse button, and drags out of its column.	A small garbage can symbol appears at the bottom of the screen
The user puts the course in the garbage.	Course is dragged over the garbage can and the mouse button is released.	The course disappears into the garbage can
The user refreshes the page.	User clicks refresh on browser.	The schedule is displayed with the user's modification to the course term included, indicating that is has been successfully saved in the system.

Table 6: Schedule Customization (Course Removal) Event Flow

5.4.6 Test Case 4a: User-caused Conflict (Prerequisite Violation)

5.4.6.1 Description

A user moves a class to an earlier term and causes a prerequisite violation. The course move will be accepted by the system but warnings will notify the user that one or more prerequisites have been violated. A warning appears to notify the user of the violation. The user cannot make further changes until the notification is dismissed and while the prerequisite is violated the course will be tinted red and display a warning icon ⚠.

5.4.6.2 Assumptions

User enters the prerequisite course in a later term than the course requiring the prerequisite.

5.4.6.3 Pre Conditions

User has an account, a generated schedule, and is on the *Program Management Page*. In their generated schedule, the course *CSC 225* will be placed in the *Summer 2016* term and course *CSC 226* will be placed in the *Fall 2016* term.

5.4.6.4 Event Flow

Test Step	Input	Expected Result
The user moves *CSC 226* into the *Spring 2016* term.	The user clicks and drags the course *CSC 226* into the *Spring 2016* term	A notification pop-up appears: *"CSC 226 has been scheduled before its prerequisite, CSC 225"*. The *CSC 226* course is moved to the *Spring 2016* term, it is tinted red and displays a warning icon, ⚠.
The user cannot make changes until the notification is dismissed.	User tries to move a different course and tries to open the **Course Management Menu** by clicking the menu button ≡ on the *Program Management Page*.	No changes to the page are made.
The user dismisses the notification.	The user presses the close icon on the notification, ×.	The pop-up notification disappears. *CSC 226* continues to be tinted red and display the warning icon, ⚠.
The warning text can be redisplayed by click on the warning icon.	The user clicks on the warning icon,⚠, for the CSC 226 course.	A pop-over appears displaying the text: *"CSC 226 has been scheduled before its prerequisite, CSC 225"*.
The pop-over is closed by clicking anywhere on the screen outside of the pop-over.	The user clicks a blank area on the navigation bar.	The pop-over closes.
The user moves *CSC 226* back to the *Fall 2016* term.	The users clicks and drags the course *CSC 226* from the *Spring 2016* term to the *Fall 2016* term.	The red tint and the warning icon are no longer displayed on *CSC 226*.

Table 7: User-caused Conflict (Prerequisite Violation) Event Flow

5.4.7 Test Case 4b: User-caused Conflict (Term Offering Violation)

5.4.7.1 Description

The user moves a course into a term where it is not offered. The system will not accept a term violation. When the user drags the course over a term in which the course is not offered the entire term will display a red shading. If the user releases the mouse button the course will revert to its initial position. A notification will appear: *“Course not offered in <Term Name>”*. This notification will not block user actions until dismissed, the notification will timeout and disappear after 5 seconds.

5.4.7.2 Assumptions

SENG 321 is not offered in the *Summer 2016* term.

5.4.7.3 Pre Conditions

The user is enrolled in *SENG 321* in the *Spring 2016* term.

5.4.7.4 Event Flow

Test Step	Input	Expected Result
The user drags *SENG 321* to the *Summer 2016* term	User clicks on *SENG 321* course in the *Spring 2016* term and drags it to the *Summer 2016* term. User does not release the mouse button	The *Summer 2016* term becomes shaded red.
The user tries to move *SENG 321* to the *Summer 2016* term	While *SENG 321* is over the *Summer 2016* term the user releases the mouse button.	The *SENG 321* course reverts back to its initial position in the *Spring 2016* term. A notification appears: *“Course not offered in <Term Name>”.*
Notification automatically closes.	No action	After 5 seconds the notification disappears.

Table 8: User-caused Conflict (Term Offering Violation) Event Flow

5.4.8 Test Case 4c: User-caused Conflict (Enrollment Limit Violation)

5.4.8.1 Description

The user moves a course to a term already containing the maximum number of courses (At UVic the maximum number of full-credit courses is 6). The system will not accept an enrolment limit violation. When the user drags a course over a full term the term will be shaded red. If a user releases mouse button the course will revert to its initial position and a notification will appear: “Enrollment limit for <Term Name> has been reached”. This notification will not block user actions until dismissed, the notification will timeout and disappear after 5 seconds.

5.4.8.2 Assumptions

Spring 2016 contains *CSC 320*, which is offered in both the Spring and Summer terms.

5.4.8.3 Pre Conditions

The *Summer 2016* term contains six full-credit courses.

5.4.8.4 Event Flow

Test Step	Input	Expected Result
The user drags *CSC 320* to the *Summer 2016* term	User clicks on *CSC 320* course in the *Spring 2016* term and drags it to *Summer 2016*. The user does not release the mouse button.	The *Summer 2016* term becomes red.
The user tries to move *CSC 320* to the *Summer 2016* term	When *CSC 320* is dragged over the *Summer 2016* term the user releases the mouse button.	The *CSC 320* course reverts back to its initial position in the *Spring 2016* term. A notification appears: *"Enrolment limit for <Term Name> has been reached".*
Notification automatically closes.	No action	After 5 seconds the notification disappears.

Table 9: User-caused Conflict (Enrollment Limit Violation) Event Flow

5.4.9 Test Case 5a: Schedule Fitting Algorithm (Course with Prerequisites)

5.4.9.1 Description

This test case verifies that the system will correctly schedule a single course after all of its prerequisites.

5.4.9.2 Assumptions

CSC 226 is a prerequisite for CSC 360. *CSC 360* is not offered in the *Summer* term.

5.4.9.3 Pre Conditions

There is no program specified, *CSC 226* and *CSC 360* are the only courses listed. The term for *CSC 226* is already chosen and it is placed in the *Spring 2016* term. The Course Management Menu is open.

5.4.9.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses	User presses **Fit Unscheduled Courses** button in the *Course Management Menu.*	*CSC 360* is automatically scheduled and it appears in the *Fall 2016* term, after *CSC 226.*

Table 10: Schedule Fitting Algorithm (Course with Prerequisites) Event Flow

5.4.10 Test Case 5b: Schedule Fitting Algorithm (Courses that are Prerequisites)

5.4.10.1 Description

This test case verifies that the system will correctly schedule a single course before all courses it is a prerequisite for.

5.4.10.2 Assumptions

CSC 226 is a prerequisite for CSC 360. *CSC 226* is not offered in the *Summer* term.

5.4.10.3 Pre Conditions

There is no program specified, *CSC 226* and *CSC 360* are the only courses listed. The term for *CSC 360* is already chosen and it is placed in the *Fall 2016* term. The Course Management Menu is open.

5.4.10.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses	User presses **Fit Unscheduled Courses** button in the *Course Management Menu*.	*CSC 226* is automatically scheduled and it appears in the *Spring 2016* term, before *CSC 360*.

Table 11: Schedule Fitting Algorithm (Courses that are Prerequisites) Event Flow

5.4.11 Test Case 5c: Schedule Fitting Algorithm (Course offered once per year)

5.4.11.1 Description

This test verifies that courses offered only once per year are placed in a valid term. The test involves integrating *SENG 299*, which is only offered in the *Summer* term, when they are already enrolled in a work term for the upcoming *Summer*. *SENG 299* should be scheduled for the following *Summer* term.

5.4.11.2 Assumptions

SENG 299 is offered only in the *Summer* term. The user has registered for *Work Term 1* in the upcoming *Summer* term.

5.4.11.3 Pre Conditions

Work Term 1 is placed in the *Summer 2016* term. The *Course Management Menu* is open. *SENG 299* is the only course that is unscheduled.

5.4.11.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses.	The user presses the **Fit Unscheduled Courses** button	*SENG 299* is scheduled in the *Summer 2017* term.

	in the *Course Management Menu.*	

Table 12: Schedule Fitting Algorithm (Courses offered once per year) Event Flow

5.4.12 Test Case 5d: Schedule Fitting Algorithm (User preferences cannot be fulfilled)

5.4.12.1 Description

This test verifies that behaviour of the system when the user specified course preferences cannot be fulfilled. The user specifies that they want to *graduate after the Spring 2017 term* and they only want to take *three courses per term*. The system cannot fit to both of the preferences specified by the user and will prioritize the graduation date constraint providing a schedule which has six courses per term and is finished after *Spring 2017*.

5.4.12.2 Assumptions

The current term being scheduled is *Spring 2016*. The student has just finished their entire third year (Term 3B, as defined in the Software Engineering course calendar) and has one work term remaining. The user has specified the Software Engineering program.

5.4.12.3 Pre Conditions

The program preferences have already been defined. The user has already indicated that *graduating after the Spring 2017 term* is a higher priority than *three courses per term*. No courses are scheduled after *Spring 2016*. The *Course Management Menu* is open.

5.4.12.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses.	The user presses the **Fit Unscheduled Courses** button in the *Course Management Menu*.	Courses are scheduled as per the UVic Software Engineering Course Calendar. Course Term 4a is scheduled for *Summer 2016*, *Work Term 4* is scheduled for *Fall 2016*, and Course Term 4b is scheduled for *Spring 2017*. A notification appears: “Unable to fulfill all course preferences. Cannot generate a valid schedule with three course(s) per term. Click here to Undo changes.”

Table 13: Schedule Fitting Algorithm (User preferences cannot be fulfulled) Event Flow

5.4.13 Test Case 5e: Schedule Fitting Algorithm (Fitting multiple courses)

5.4.13.1 Description

This test case covers the situation where multiple courses are need to be scheduled each with specific requirements. This test is designed to account satisfying the requirements of test cases 5a to 5d in a single action. The result should be the same as the combined effect each of the individual test cases.

CSC 225 and *CSC 226* will be fitted into the schedule. *CSC 110* will be marked as completed and *CSC 360* has been set for the *Fall 2016* term.

SENG 299 will be fitted into the schedule in the *Summer 2016* term and will violate the *only one course per term* preference since *CSC 226* will also be scheduled in the *Summer 2016* term.

5.4.13.2 Assumptions

No program is specified. The system assumes graduation requirements are fulfilled upon completion of *CSC 225*, *CSC 226*, *CSC 360*, and *SENG 299*. *SENG 299* is only offered in the Summer. *CSC 110* is a prerequisite for *CSC 225. CSC 225* is a prerequisite for *CSC 226*. *CSC 226* is a prerequisite for *CSC 360*.

5.4.13.3 Pre Conditions

User specifies that they want to graduate after *Fall 2016*, as their highest priority, and that they only want one course per term. The *Course Management Menu* is open.

5.4.13.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses.	The user clicks the **Fit Unscheduled Courses** button in the *Course Management Menu.*	CSC 225 is scheduled for Spring 2016. CSC 226 and SENG 299 are scheduled for Summer 2016. Graduation date still set for the end of Fall 2016 term. A notification appears: “Unable to fulfill all course preferences. Cannot generate a valid schedule with one course(s) per term. Click here to Undo changes.”

Table 14: Schedule Fitting Algorithm (Fitting multiple courses) Event Flow

5.4.14 Test Case 5f: Schedule Fitting Algorithm (Cannot satisfy program requirements)

5.4.14.1 Description

This test case verifies how the system handles fitting a schedule that cannot satisfy the requirements specified by the University. In this test case there is no valid position for

CSC 225, the upcoming semester is full and the following semester contains a course, *CSC 226*, which requires *CSC 225*. The system will fail to generate any valid schedule and will notify the user about the issue.

5.4.14.2 Assumptions

CSC 225 is required for taking *CSC 226*. Six courses are specified for the upcoming term, this satisfies the maximum enrollment limitation for that term.

5.4.14.3 Pre Conditions

CSC 226 is scheduled for the next semester. The current semester already has full enrollment, i.e. six courses are specified. *CSC 225* is unscheduled and will be automatically fit.

5.4.14.4 Event Flow

Test Step	Input	Expected Result
User fits unscheduled courses	The user clicks the **Fit Unscheduled Courses** button in the *Course Management Menu*.	No changes to the schedule occur. A notification appear: “Unable to schedule *CSC 225* because its prerequisite, *CSC 226*, is scheduled for the current term.”

Table 15: Schedule Fitting Algorithm (Cannot satisfy program requirements) Event Flow

5.4.15 Test Case 5g: Schedule Fitting Algorithm (Fitting courses to an empty schedule)

5.4.15.1 Description

This test case verifies that the system will generate a schedule matching the UVic course calendar is no preferences or course selection is specified.

5.4.15.2 Assumptions

The user has chosen the Software Engineering program. No program preferences have been specified. No courses have been manually scheduled. *Fall 2016* is the upcoming semester.

5.4.15.3 Pre Conditions

The user’s schedule is empty. The *Course Management Menu* is open.

5.4.15.4 Event Flow

Test Step	Input	Expected Result
User fits their unscheduled courses.	The user clicks the **Fit Unscheduled Courses** button in the *Course Management Menu*.	The schedule generated will match the schedule specified by the UVic Software Engineering course calendar.

Table 16: Schedule Fitting Algorithm (Fitting courses to an empty schedule) Event Flow

5.4.16 Test Case 6: Course Overrides

5.4.16.1 Description

This test verifies that a user is able to schedule a course before or with its prerequisite. It tests the use of the override function, where a user can save their schedule even though a course's prerequisite is not complete.

5.4.16.2 Assumptions

The user has not completed SENG 265. The user has received an override for SENG 321, whose prerequisite is SENG 265. The term selected for adding SENG 321 is a valid term.

5.4.16.3 Pre Conditions

User is logged in to Graduate! and is on the *Program Management Page* with the *Course Management Menu* displayed.

5.4.16.4 Event Flow

Test Step	Input	Expected Result
User will select **Required Courses** from the *Course Management Menu*	User clicks **Required Courses**	List of all required courses relating to the user's degree are displayed
User selects and drags SENG 321 into a valid term	User drags SENG 321 into the schedule	A notification is displayed indicating SENG 321's prerequisite has not been satisfied. Since the user has an override, a checkbox is displayed in the notification marked "Ignore"
User marks the "Ignore" check box	User clicks checkbox	SENG 321 is added to the specified term

Table 17: Course Overrides Event Flow

5.4.17 Test Case 7: Saving User Data

5.4.17.1 Description

This test case verifies that a user can make modifications to their schedule and have it save to their profile after the modifications have been made.

5.4.17.2 Assumptions

User has registered an account but has not made modifications to the originally generated schedule. The user did not select "unspecified" for type of degree. The course being added is in a valid term. User uses the same account for adding a course, as well as logging out and back in.

5.4.17.3 Pre Conditions

User is logged in and on the *Program Management Page* with the *Course Management Menu* displayed.

5.4.17.4 Event Flow

Test Step	Input	Expected Result
User will select **Required Courses** from the *Course Management Menu*	User clicks **Required Courses**	List of all required courses relating to the user's degree are displayed
User selects and drags a required course into a term	User drags a course into the schedule	The course populates in the desired term and automatically saves
User will log out of Graduate!	User clicks **Log Out**	User is redirected to the *Front Page*
User logs in to Graduate!	User email and password	User successfully logs in and is redirected to the *Program Management Page*
User views their schedule	No action	The course that was added before logging out is displayed in the correct term

Table 18: Save User Data Event Flow

FEEDBACK

Additional Testing Requirement

- Perform User Experimentation

Gather up to 3 friends/co-workers and test the UI

1) Create 3 distinct task scenarios with step by step Use Cases for each
2) Explain each scenario to a different person and ask them to perform the respective Use Case alone (so the other two people are unaware)
3) Do not guide them while they perform
4) Once all 3 complete their scenarios, switch up the scenarios so everyone ends up with a new case to work on.
5) However, this time do not show them the scenarios, rather explain what has to be done and let them work on their own.
6) During both phases, record the number of clicks, the time it takes to complete the scenario, and the number of times the tester makes a mistake that puts them a step backward.

This test will provide feedback on the design of the UI and navigation through the software.