Project: StudyBuddy

Santa Clara University

COEN 275

Professor Leyna Zimdars

Team Members: Thilina Mallawa Arachchi, Pawan Kumbhare, Visalakshi Gopalakrishnan

8-9-2016

Table of Contents

1. Introduction

1.1. Purpose

Breaking into social circles in many aspects of life can be a daunting task, especially in the fast-paced world of technology, both fueling and stemming the issue in today's world. Some websites like Facebook help by providing a platform for wider social engagement that facilitates interactions in today's information age..

The social problem we attempt to tackle takes place In the classroom. Students learn from group social interactions with their peers and teachers in class, but they have trouble translating this experience outside the classroom. We believe this to be because no effective tool exists to aid studious group interactions, outside of the classroom environment, while facilitating students varying social skillsets and study preferences.

At Santa Clara University, students are presented with online collaboration tools, such as the in-house Camino portal, however these tools are not conducive to forming intimate and collaborative social networks to aid in group learning. We present our idea, StudyBuddy, as a tool to decouple the social barriers from creating study groups to enhance the learning experience.

StudyBuddy is our attempt at using OOADP principles to create a novel social application to aid students in an academic institution to facilitate finding and meeting potential group members.

1.2. Team

The StudyBuddy development team consists of 3 core members: Thilina (Thil), Pawan and Visalakshi (Visa). Visa, our idea originator, was instrumental in defining the overall vision and behaviour analysis for StudyBuddy. Pawan, our coding guru, provided

a solid technical foundation to roadmap our development. Thil, our traceability coordinator, ensured the initial vision mapped throughout our development activities.

After the initial brainstorming and development of functional requirements, the project was divided into 3 major functionality and workload division- Login/Search, Manage Group/Notifications and MyProfile/Signup. Each team member was responsible for one functional behaviour, setting and contributing to the software development process for that functional behaviour.

Our software development process helped define and highlight the team's roles, responsibilities, strengths, and ability to handle any unexpected developments and project timeline demands.

2. OO Approach

2.1. Requirement Planning

2.1.1. Our Problem Domain

StudyBuddy's problem domain is influenced from the social networking, and educational domains. The team looked to internal resources within Santa Clara University, such as Camino, and those used externally, such as Facebook, to establish current capabilities and requirements that shape social grouping.

From these sources we determined how users interact with social grouping applications, and the common behaviours that are carried out to ensure a smooth social experience. It was here we found out that our application was quite novel because of its niche application, and required a complete understanding of the domain and the behaviours of actors that shaped our endeavour.

To flesh out the details we conducted open brainstorming sessions, among the team, to determine what a student would want in a group matching application. We then

conducted a few interviews with friends, among the Santa Clara University student base, to test our hypothesis and firm up our domain analysis. Some examples of interview feedback and domain mapping is as below:

User Needs	Domain Mapped Feature
“I want to find someone in my class who would like to study with me.”	Search within a group
“I want to let other people know that I’d like to form a group.”	Setting user preferences
“I only want to study with people that I know.”	Friends, known connections

Looking to our Problem Domains helped to set our Glossary of Terms (2.1.2) and shape our initial Project Scope (2.1.3).

2.1.2. Glossary of Terms

Term	Definition
Login	Gaining access to the system using credentials
Signup	Setting up credentials for system access
Student	Individual users of the system
Search	Looking for study group matches
Group	An existing study group
Notification	System generated notification for a group requests
Manage group	Ability to lock/unlock/approve access to the student’s

	groups, and group notifications
Join group	The action of joining a new or existing group
Create group	The action of creating a new group
Leave group	The action of leaving an existing group
Delete group	The action of deleting a group only if the member count is less than 2
Accept / Deny request	Accept or deny requests to create a new group or join an existing group
Lock / Unlock group	Lock or unlock an existing group, locked groups prevent others from joining
Past Course	A list of course history for a student, that is used in matching search requests if the student has indicated to help future students
Current Course	A list of courses the student is currently enrolled in and allowed to search for groups
Add/Delete Course	Allow student to add/delete course from current quarter
Profile	Registered user profile
Edit	Allow user to edit profile information except email and Student ID
Update	Update edited profile information in database
Reset	Reset the target frame and all information related to it
Logout	End session

2.1.3. Setting the scope

Being a greenfield project, our application was susceptible to scope creep. In the first 2 weeks of development, we noticed the trends of new feature add-ons becoming more prevalent in our team discussions. To mitigate this and ensure we met the 8 week project deadline, the team decided to constrain StudyBuddy to be a simple tool for study group creation, leaving the social interaction of meeting and collaborating outside the sphere of the application. We also incorporated a stringent coding process to help deal with feedback and improvements that would inevitably crop up, and to help mitigate any disruptions to development, as outlined in 2.4.1.

We developed our application scope by focusing on 5 major behaviours that were needed to provide a complete, yet simple, group formation experience on our platform:

1. **Login/Signup** - for access to StudyBuddy
2. **Search** - to look for study partners or groups
3. **Manage Groups** - to manage a student's current groups and group requests
4. **My Connections** - to keep a history of a student's memorable/favourite group buddies for future matching
5. **Profile** - to build the student's course and study preferences

2.2. Analysis

2.2.1. Developing Use-Cases

Mapping each behavior into a use case allowed the team analysis dependencies and flesh out interfaces that would be required in the code design phase of the project.

2.2.1.1. Login/Signup

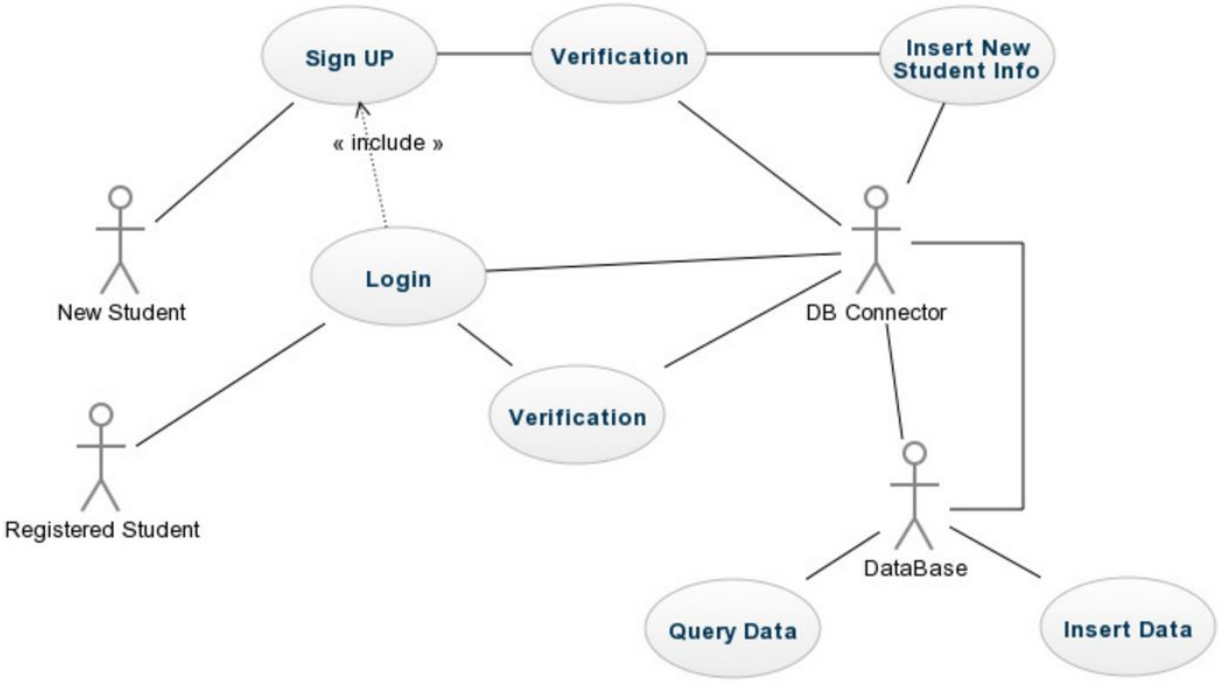

Figure 1 - Login/Signup Use Case

The ‘login/signup’ use case shows StudyBuddy’s access protocols. Signup executes and is dependent on login if the student has no login credentials. In the signup chain the student must provide their educational information in the correct format before being entered in the system. If the student has login credentials they are able to use the login chain to access StudyBuddy. The DB Connector actor makes sure the information that is passed is private and relevant to each chain in the access sphere.

2.2.1.2. Search

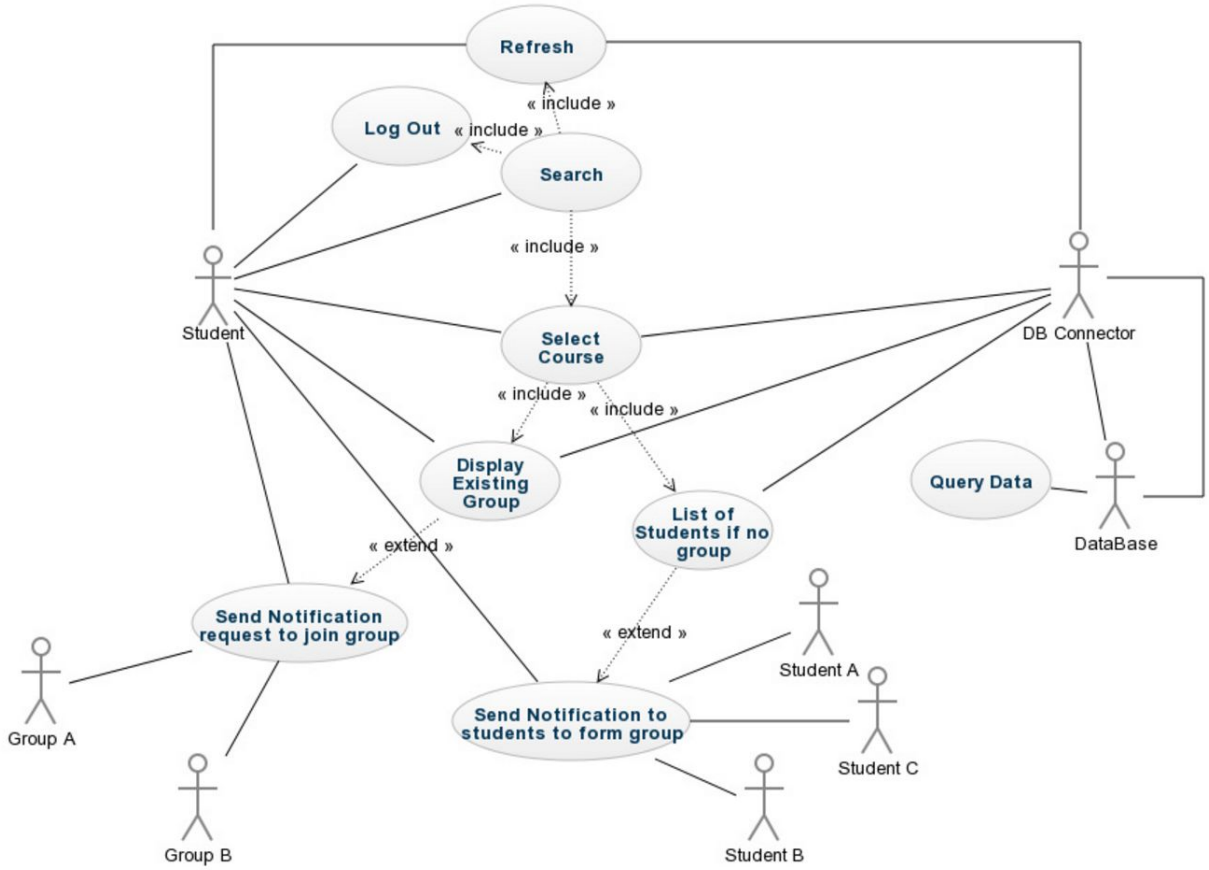

Figure 2 - Search Use Case

The ‘search’ use case shows StudyBuddy’s matching protocols. A student can view groups or matched students by the course they are taking, then send notifications to join if they wish. Sending notifications is not mandatory, the student can also leave the application after viewing results. The DB Connector actor makes sure the information that is passed is private and relevant to the search sphere.

2.2.1.3. Manage Groups

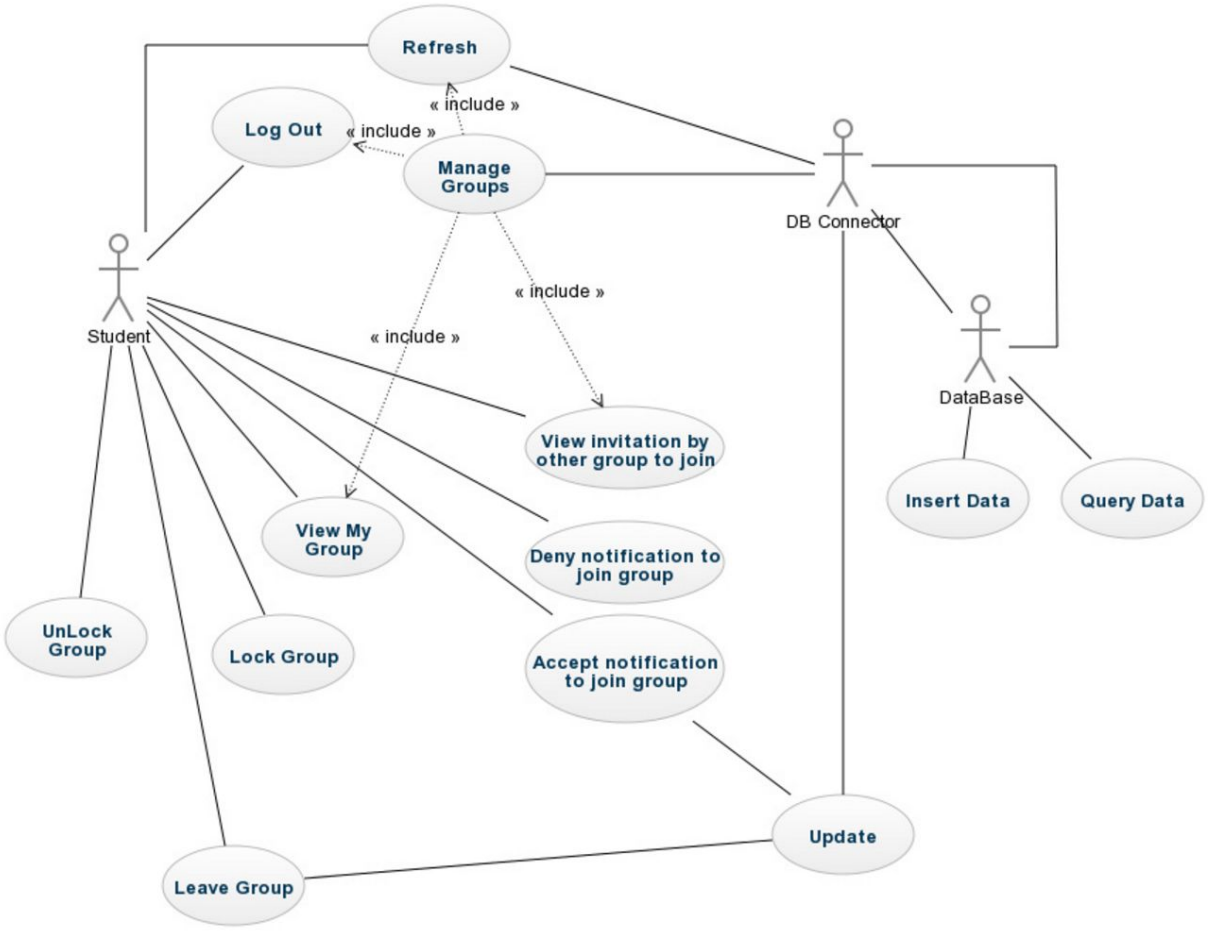

Figure 3 - Manage Group Use Case

The ‘manage group’ use case shows StudyBuddy’s group management protocols. A student can view their groups, group notifications, and new group requests. For existing groups the student should be able to leave, lock or unlock the group. For new group requests the student should be able to accept or decline the request. For existing group notifications the student should be able to view the notification and accept or deny it. The student can also leave the application after viewing their groups. The DB Connector actor makes sure the information that is passed is private and relevant to the group management sphere.

2.2.1.4. My Connections

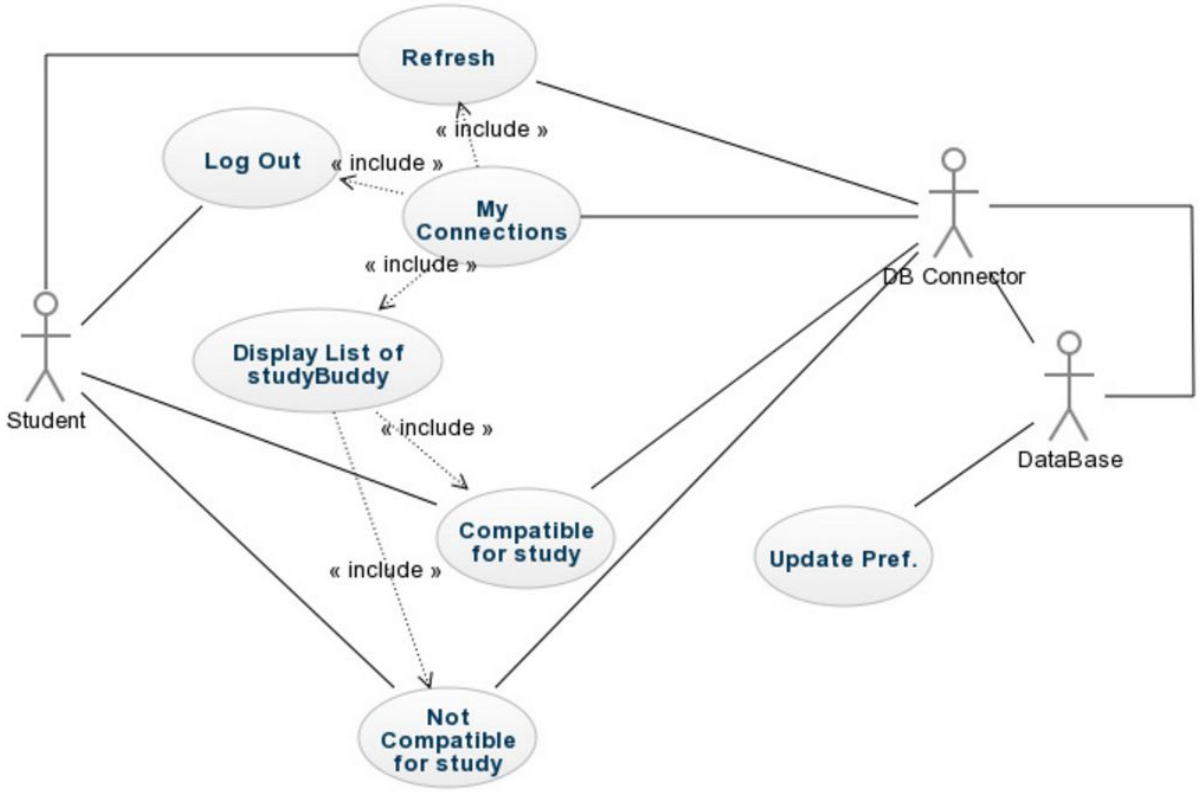

Figure 4 - My Connections Use Case

The ‘my connections’ use case shows StudyBuddy’s connection rating protocols. A student can view their past group partners in a list and either rate them ‘liked’ for future matching or ‘unliked’ to prevent any future group matching with that connection. The student can also leave the application after viewing their connections. The DB Connector actor makes sure the information that is passed is private and relevant to the ‘my connection’ sphere.

2.2.1.5. Profile

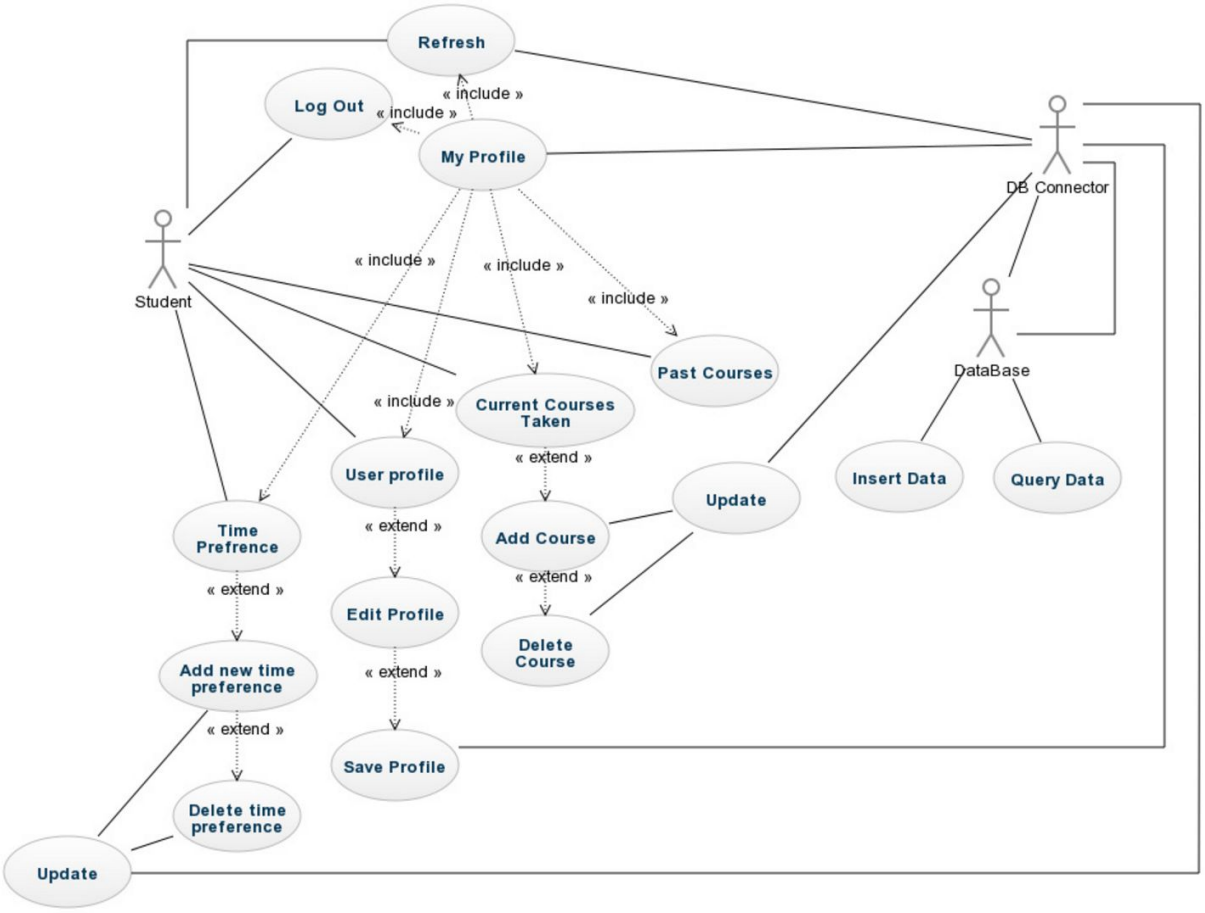

Figure 5 - My Profile Use Case

The ‘my profile’ use case shows StudyBuddy’s user preferences protocols. A student must add their profile, current courses and time preferences in order to be able to search for group partners. The student may choose to update their information at a later date once their initial profile has been created. The student can also leave the application after viewing their profile. The DB Connector actor makes sure the information that is passed is private and relevant to the ‘my profile’ sphere.

2.2.2. Mapping Interaction in the Activity Diagram

In order to understand the possible system states, dependency interactions between use cases, and functions required at each state, an activity diagram for each major behaviour in StudyBuddy was analysed:

2.2.2.1. Login / Signup

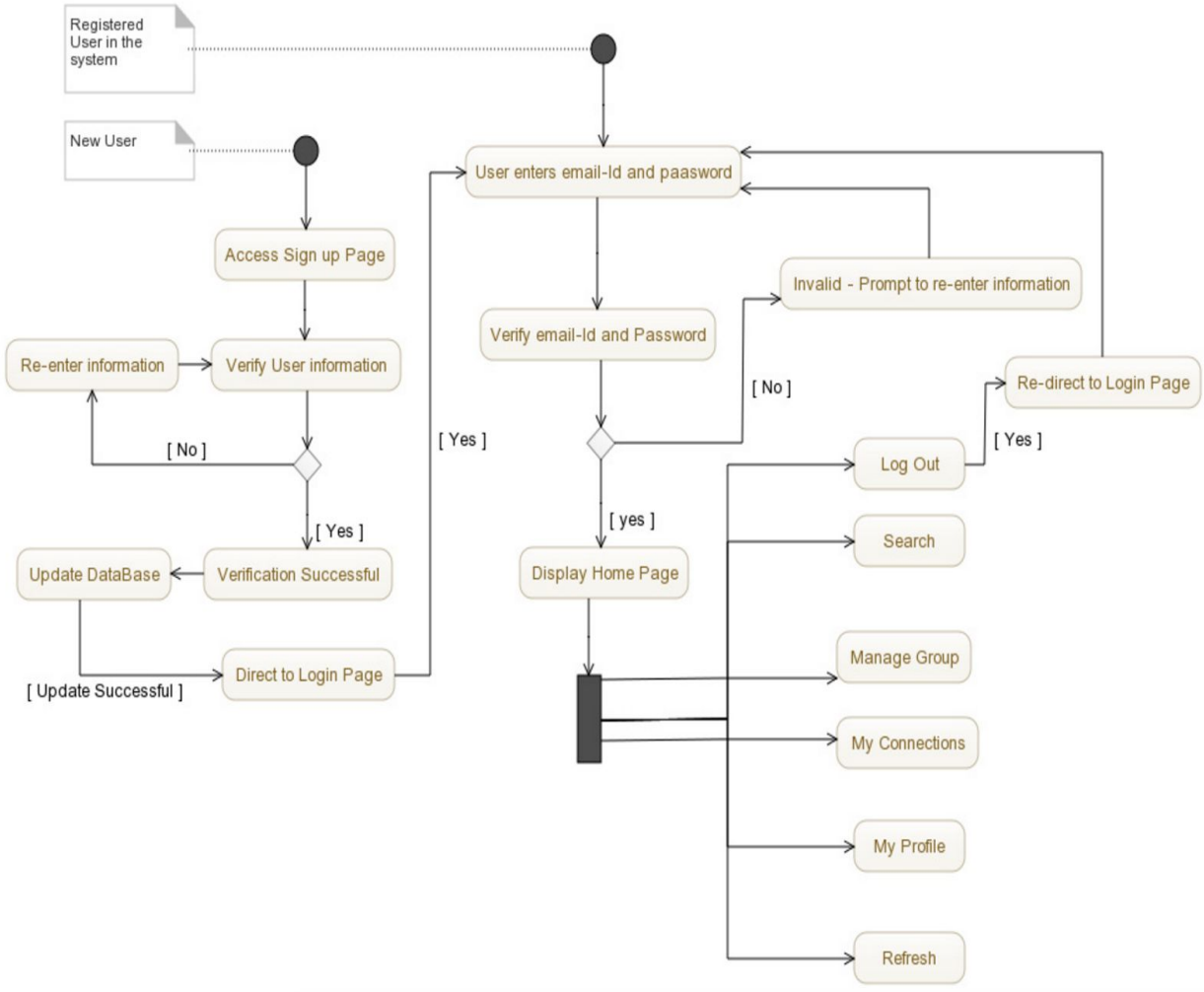

Figure 6 - Login/Signup Activity Diagram

2.2.2.2. Search / Manage Group

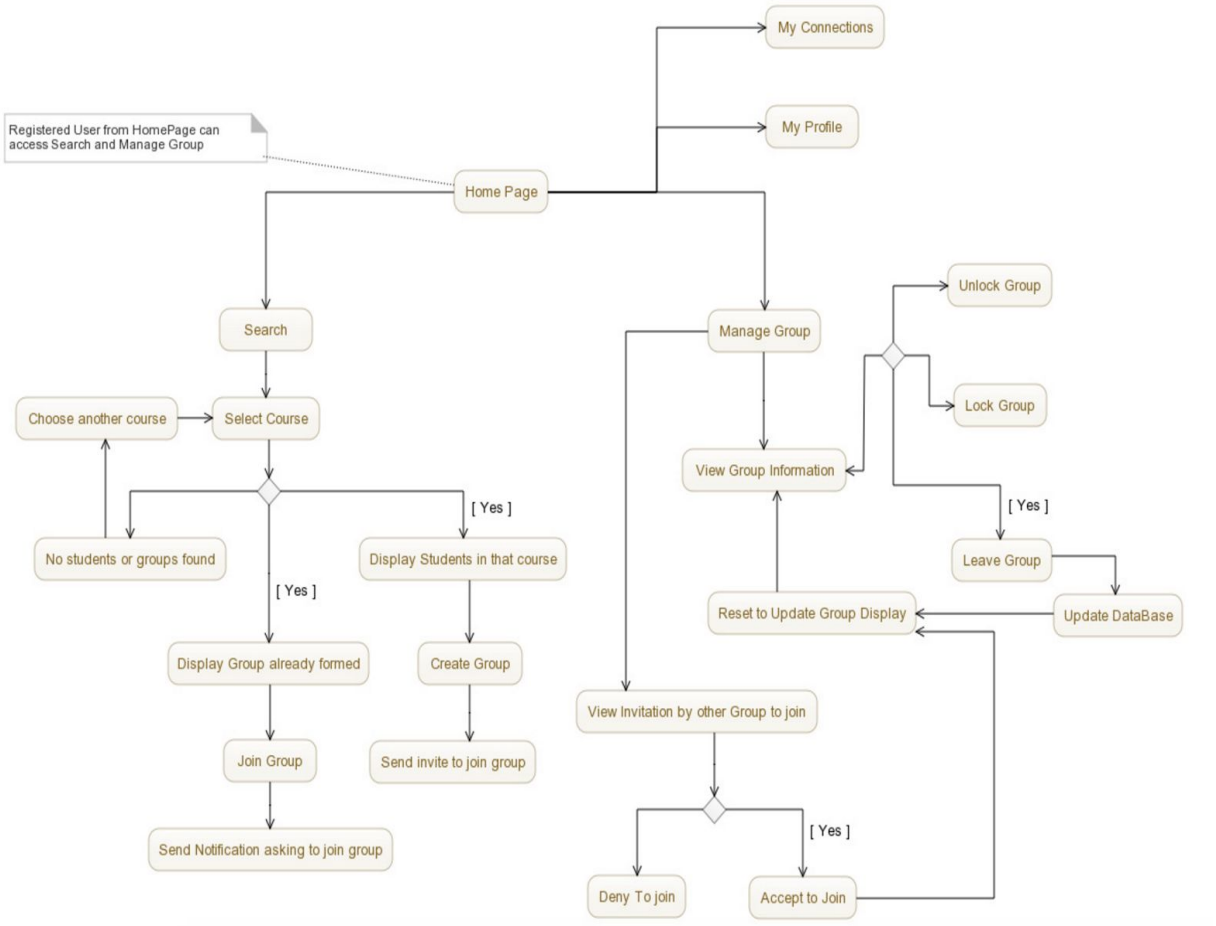

Figure 7 - Search / Manage Group Activity Diagram

2.2.2.3. My Connections / My Profile

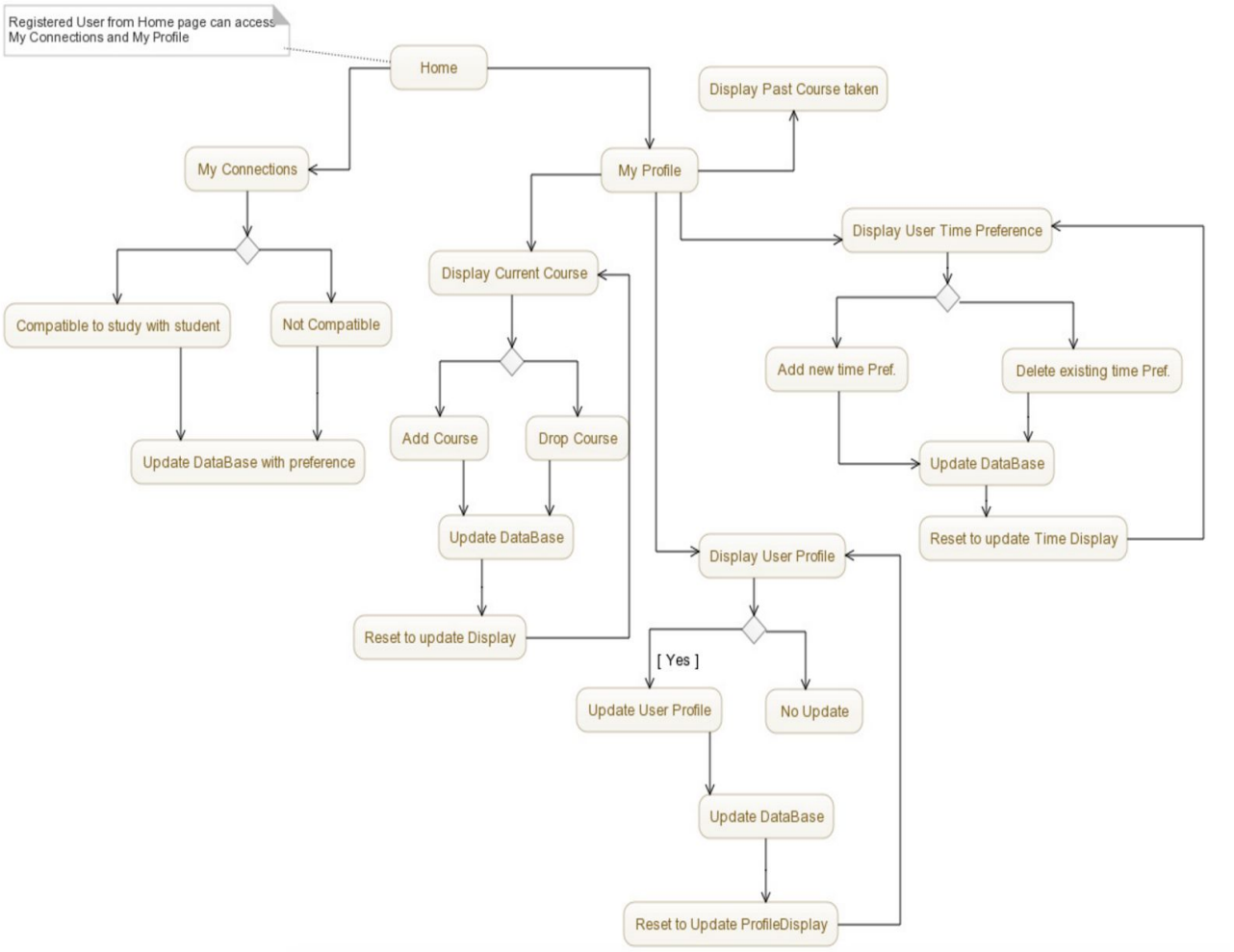

Figure 8 - My Connections / My Profile Activity Diagram

2.3. Design

2.3.1. Understanding the programming language and choice of architecture: The Model View Controller

Prior to delving into class design, the team investigated the Java programming language and industry best practices in UI heavy applications that would help us to structure our application, and map the responsibilities and interactions for each part in that structure.

We found the Model View Controller (MVC) architectural pattern to be well thought out in the method it handled data and logic in its Model, presented data to the user in Java's supported format via its View, and handled user requests and resources to carry them out in its Controller. We mapped the MVC elements in our application by:

1. Creating a 'bean' package in our application that held the .java files that performed the logic as required by our major behaviours. (Model)
2. Having all UI presented in the .form files. (View)
3. Allowing users to interact and perform tasks via the UI JFrames that encapsulate behaviours in our 'jform' .java files.

2.3.2. Transforming Use Cases to Class Designs

Our use cases encapsulated the major behaviours of the system, this allowed us to implement behaviour based implementation of classes under the MVC paradigm - logging in and signing up, searching for groups, managing group semantics, liking connections, and setting user preferences. StudyBuddy's simplified class foundation is shown in Figure 9.

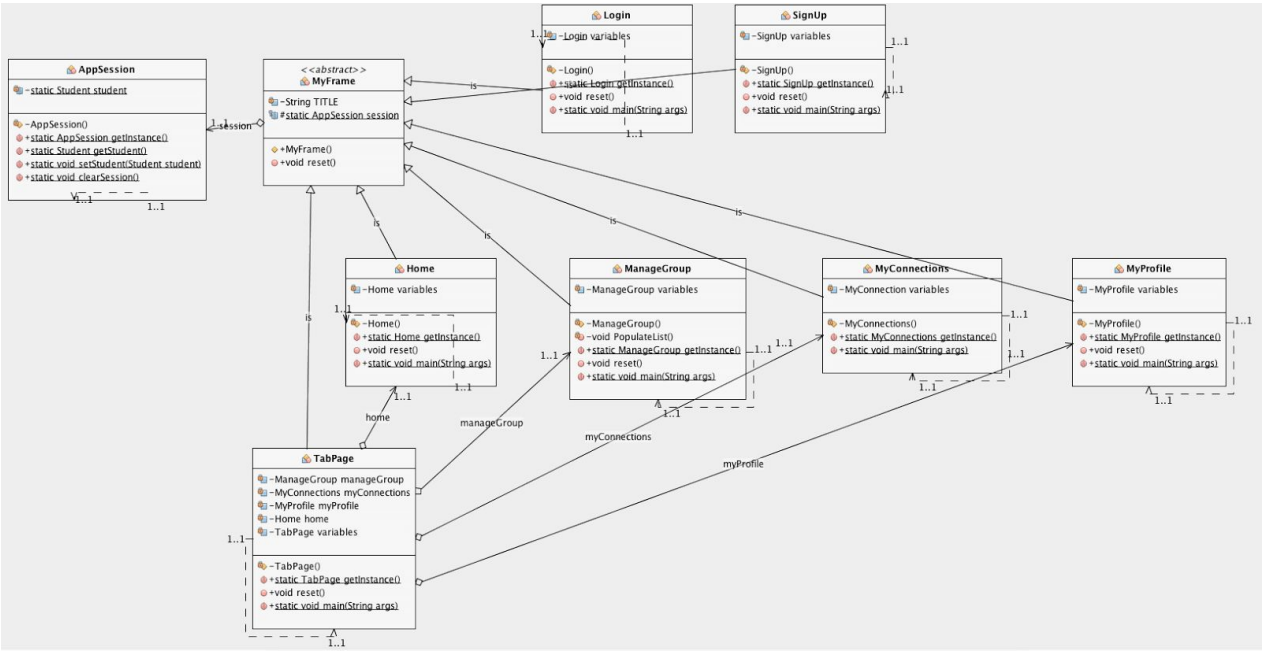

Figure 9 - StudyBuddy core class diagram, simplified.

MyFrame class is the abstract class that our behaviors inherit from and implements its own versions of the superclass's methods for reset(). The TabPage class holds the instantiated objects for our behaviours allowing the student to switch between each one. Once the student carries out sub activities in within the behavior, they can call the behavior's reset() method to update the specific behaviors information and UI, or the superclass's reset() method to update information across behaviors.

Each behaviour had its own interface to the databases. This is shown in simplified form in Figure 10 for My Profile.

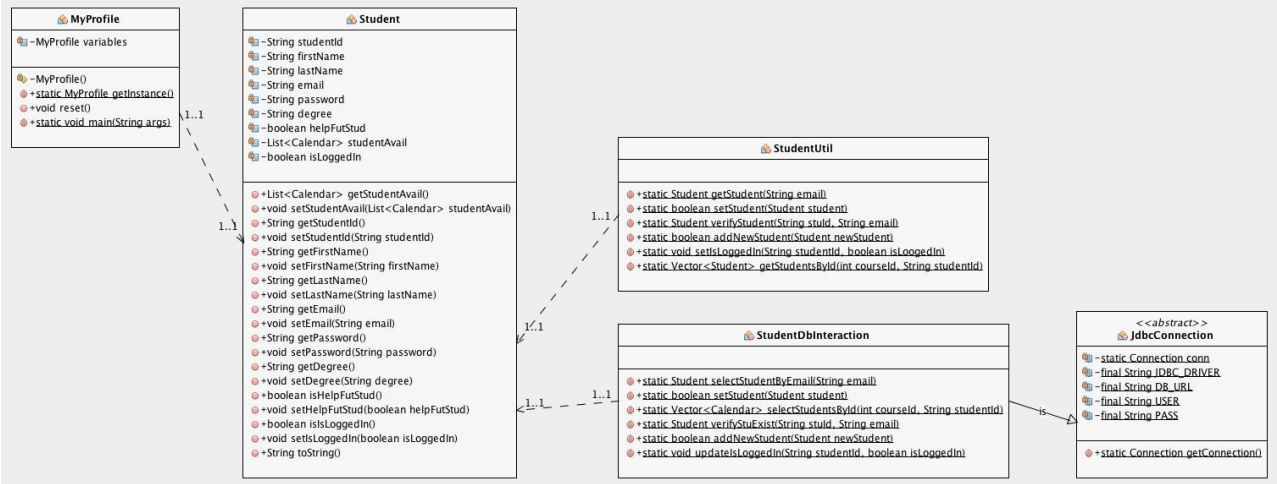

Figure 10 - The MyProfile class interaction with a Student database

The MyProfile class primarily deals with the Student object and it is via this object that data for a particular student is massaged and updated to a database, via the StudentUtil class that calls the StudentDbInteraction class that contain the MySQL database queries to be run.

2.3.3. Incorporating the 5C's of Class Design

2.3.3.1. **Cohesion**:

"Thus, the class Dog is functionally cohesive if its semantics embrace the behaviour of a dog, the whole dog, and nothing but the dog." - Grady Booch

Our approach embodied Booch's adage, where each class represents a functionality in its entirety:

Class **MyProfile** deals with all the behaviours of the student that one would expect, as outlined in our requirements when dealing with profile creation and preference updating. The class MyProfile is responsible for keeping track of registered student's current/past courses taken, student time preference and student profile information. This class only facilitates modification to above related behaviours, and works with our databases, containing student data, in the same scope.

Class **Home** is responsible with all the behaviours related to group search functionality.

Search for groups is based on the student's current courses as reported by MyProfile. The class encompasses a student's desire to join or create a new group with fellow students in their class.

Our database interaction classes like **CalenderDbInteraction**, **CourseDbInteraction** etc. are responsible for facilitating access to database containing information relevant to that class seeking it. For instance CalanderDbInteraction will only interact with the Calendar database containing the student ID and their preferred times. If more indepth information on the course is required for that particular student, the querying classes must interface with the CourseDbInteraction class.

All of StudyBuddy's class design has been implemented while keeping in mind the functional cohesion aspect, in which all the methods of that class work together to provide a well defined behaviour.

2.3.3.2. Completeness:

Completeness in class design was achieved by capturing all of the meaningful characteristics of the modelled behaviour. We used our Domain Knowledge in naming the Class by the functionality it represents, including its encapsulated behaviors. Each class captures all the behaviours in the form of methods that completes the characteristics of those behaviours.

Example :

Class **ManageGroup** defines the functionality of managing a student's groups. The method in the class, such as ManageGroupRenderer(), handle all presentation and UI interaction for the class behaviors of locking, unlocking, notification handling, joining and leaving pertaining to groups as well as event handling from group and notification selection.

2.3.3.3. Convenience:

Our project revolves around modularity. Each module is designed to capture sufficient behaviour and characteristics of the functional requirements, to permit efficient and complete meaningful interaction between modules. All classes have been designed to complete tasks

assigned to it.

Example:

For group searching to be convenient, at the very minimum behaviours that define that class - such as searching for a group based on students current courses, display groups if found, and the ability to join groups, should be available. This behaviour can then be extended to have an option to form group if no group exists. The behaviour can further be extended to have option for sending notification to groups of intent to join them.

2.3.3.4. Clarity and Consistency:

Clarity and consistency are essential for successful project coordination and interfacing, especially when multiple team members are involved in development. Apart from glossary of terms, the team incorporated a standard for coding and naming conventions to ensure that the project can be easily understood by both the team and potential non-team members.

Consistency in form of the MVC architecture was applied to our project to ensure a very specific flow pattern was maintained throughout the project. Clarity is achieved through naming convention of packages and contained classes.

Source Packages were named based on the utility. Package **edu.scu.studybuddy.bean** has classes named after the tables hosted in remote MySQL database. For example, Table Student has all information about a student - Student Id, courses taken, etc. Therefore the class is in the bean package is named ‘Class Student’ and all the private data of that class is encapsulated. Similarly Package **edu.scu.studybuddy.dbinteraction** has classes named after the corresponding tables in our MySQL database that a class may need to access. Class **CalenderDbInteraction** will only access table calendar from the database, and similarly Class **StudentDbInteraction** will only access **Table Student** from database.

To maintain consistency a uniform protocol to access and modify data was defined. **Class Search** in order to access **Course Table** in the database, first has to call **Class CourseUtil** which in turn would establish connection to database via **Class**

CourseDbInteraction.

Clarity was ensured through naming convention of methods and variables. Method **sendJoinGroupRequest()** means exactly as it is named. That is, to send a request to join group. Variable **jTStuId** means it is a **jTextfield** that holds student Id.

2.3.4. Sequence Diagrams

To help define the boundaries of each behaviour, and its dependance on other classes within its sphere of influence, sequence diagrams for each behaviour was developed. The diagrams also show the interactions of the MVC model.

2.3.5. Login/Signup

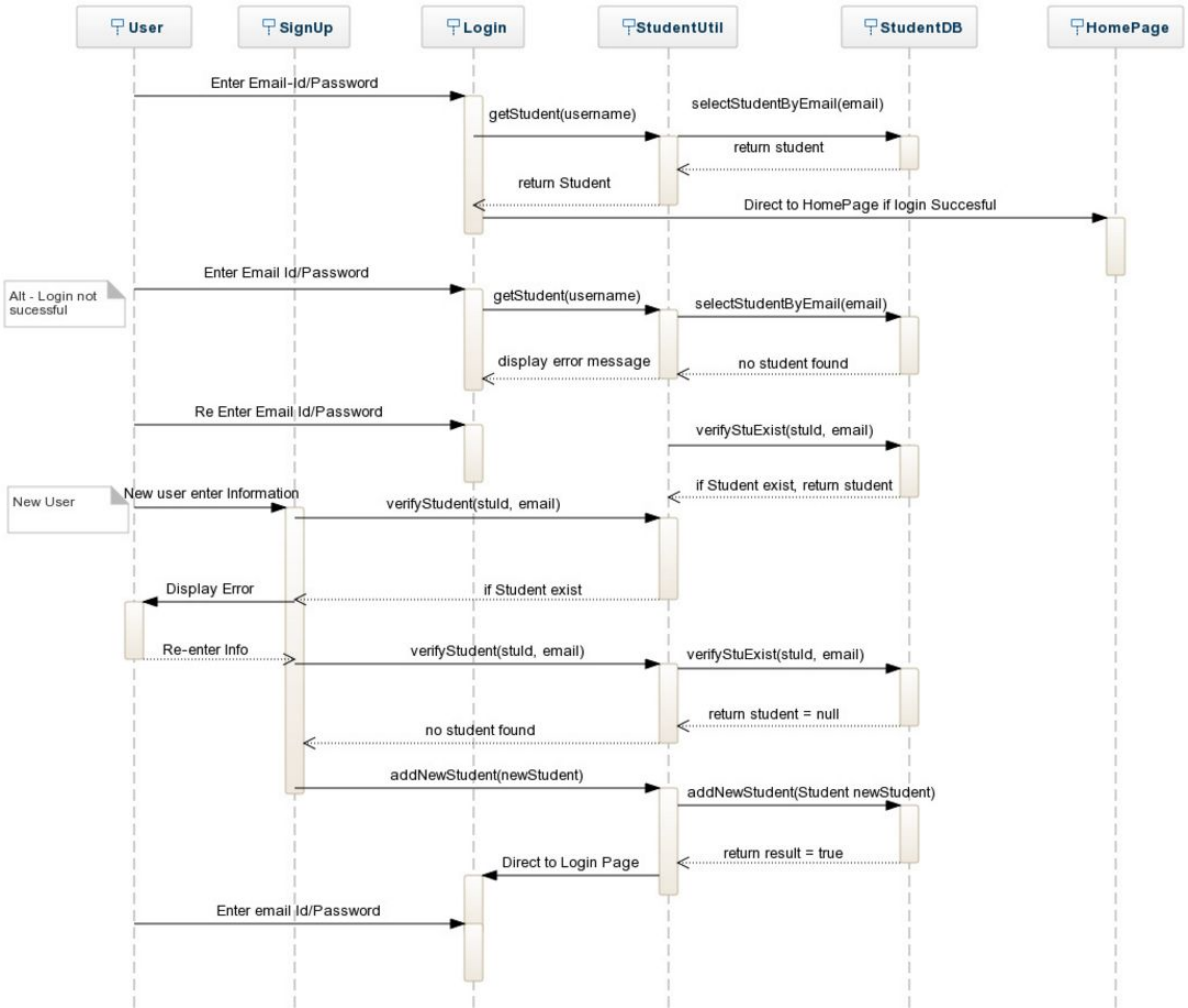

Figure 11 - Login/Signup Sequence Diagram

2.3.6. Search

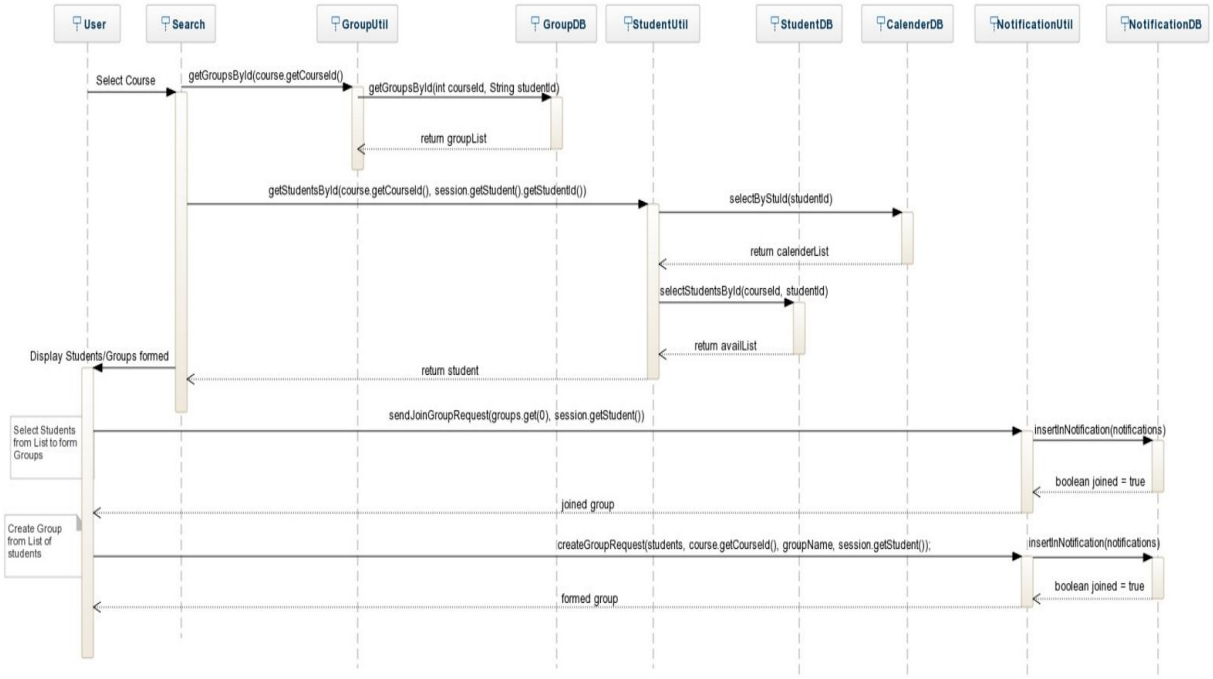

Figure 12 - Search Sequence Diagram

2.3.7. MyConnection

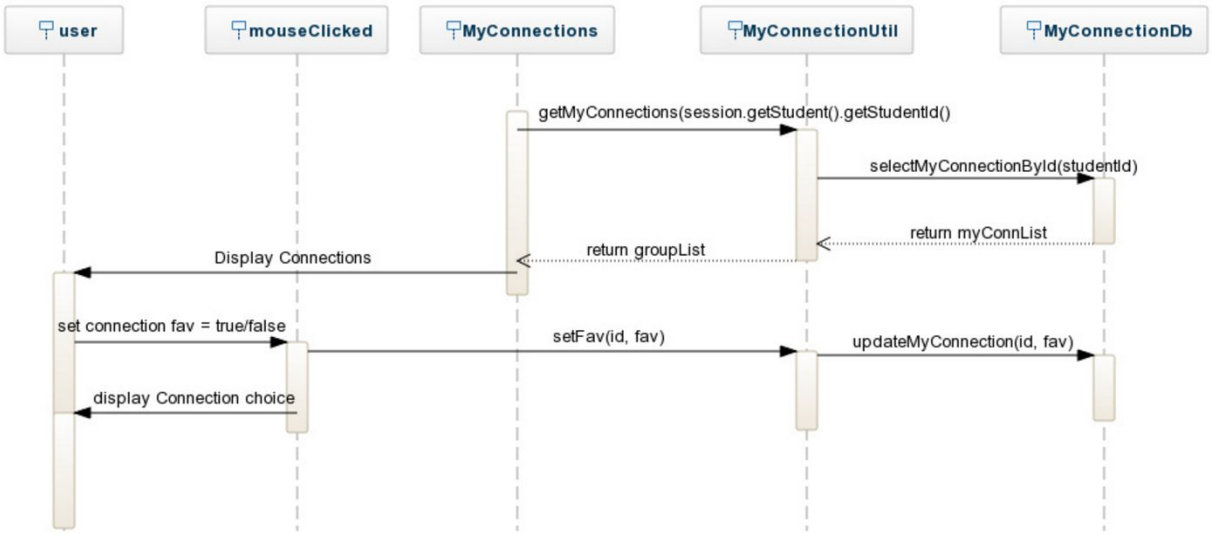

Figure 13 - My Connection Sequence Diagram

2.3.8. Manage Groups

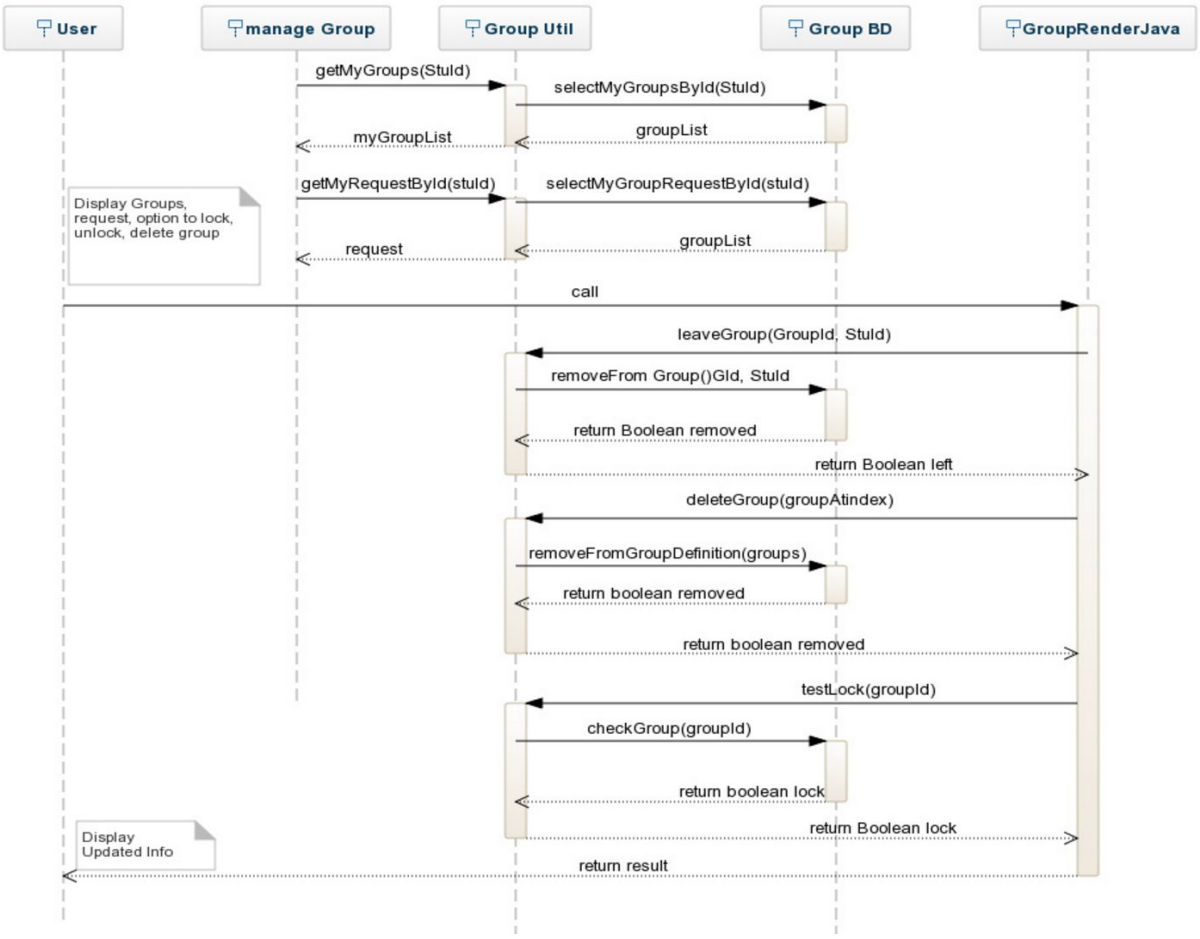

Figure 13 - Manage Group Sequence Diagram

2.3.9. My Profile

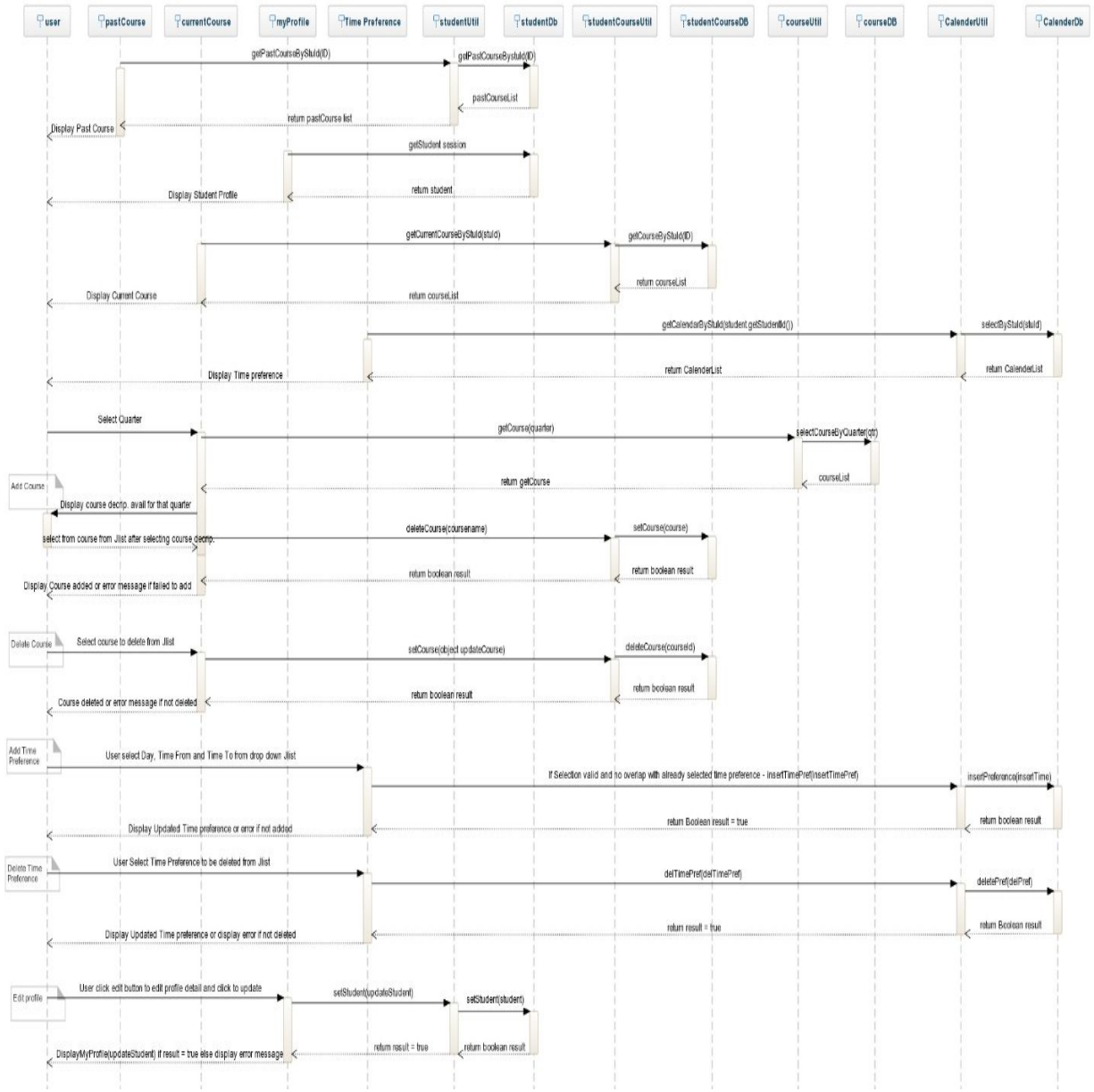

Figure 15 - My Profile Sequence Diagram

2.4. Implementation

2.4.1. The process of programing

StudyBuddy was an ambitious undertaking, in order to ensure coding efforts were distributed effectively across the team and improvements were introduced sustainably, a spiral software process was used with decentralised tasked coding. The MVC architecture allowed us to have well defined interfaces in our code, enabling sandboxed behaviour development. The spiral process was facilitated by using BitBucket as our versioning tool, and team members were encouraged to follow an iterative coding mantra where essential features were first implemented and others incorporated after validating against our initial scope.

2.4.2. Use of Patterns

We have used three design patterns in our project as described below:

2.4.2.1. Singleton Pattern

Every class extending JFrame in our project is a singleton class.That means only one instance of a window will get generated while running the application

Advantages : To avoid multiple instances of same window, which may lead to data inconsistency.

2.4.2.2. Observer and Strategy Pattern

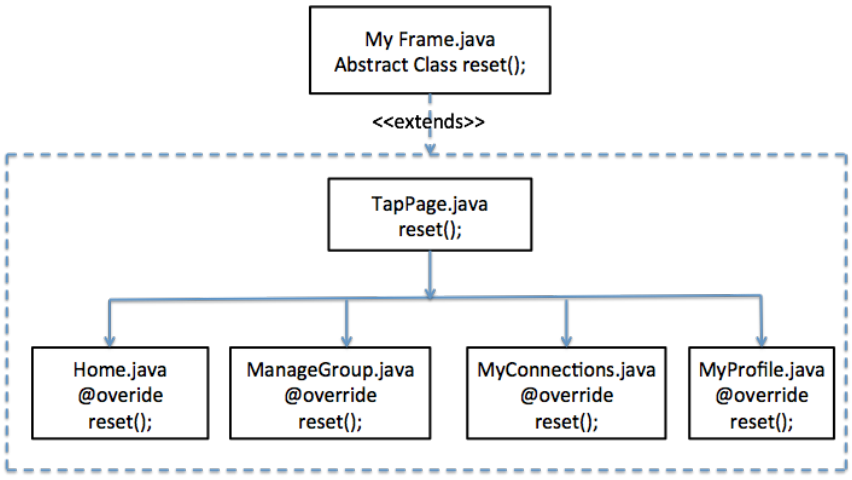

Figure 16 - Hybrid Observe/Strategy Pattern

Individually reset/refresh the GUI or reset/refresh using Tab Page. Here TabPage.java act as the observer. We have created a MyFrame abstract class defining an action and concrete classes extending the MyFrame, as explained in 2.3.2.

2.4.3. Traceability diagrams

We have maintained traceability matrix to keep track of all the diagrams and documents connectivity. Figure 17 shows a snippet of our team's matrix. If at any stage interfaces needed to be modified, or coding to be redistributed, the traceability diagram allowed rapid pickup of all essential elements pertaining to our major modeled behaviors.

UML	Sequence Diagram	Use Case Number	Use Case	Sub Use Case	Calendar.java	Course.java	GroupDefinition.java	GroupStudent.java	MyConnection.java	Notification.java	Student.java	StudentCourse.java
UML-1	SD-1	UC-1	Login	Login/Logout							X	
			SignUp	Add User							X	
UML-2	SD-2	UC-2	Search	Display Courses							X	X
				Group			X	X				
				Students							X	X
				Create New Group Request						X		
				Create Join Group Request						X		
	SD-4	UC-3	Manage Group	Display Requests						X		
				Accept Request						X		
				Deny Request						X		
				Lock / Unlock Group			X				X	
UML-3	SD-3	UC-5	My Connections	Display Old Buddies					X			
				Like Buddy					X			
	SD-5	UC-4	My Profile	Display Past Courses		X		X			X	X
				Display Current Courses		X		X				X
				Add Course								X
				Delete Course								X
				Update User Profile							X	
				Add Time Preference	X							
				Delete Time Preference	X							

Figure 17 - StudyBuddy Traceability Matrix

2.5. Testing

2.5.1. Unit Testing

For testing we have divided our process into three parts:

2.5.1.1. Sanboxed Unit Testing

Each team member has performed unit testing of the part they are developing.

2.5.1.2. JUnit Testing

Where possible some modules have been tested via JUnit testing. Figure 18 shows a JUnit test setup for our **getStudent()** method.

```
/**
 * Test of getStudent method, of class StudentUtil.
 */
@Test
public void testGetStudent() {
    System.out.println("getStudent");

    String email = "clei@scu.edu";

    Student expResult = new Student();
    expResult.setStudentId("W2224224");
    expResult.setEmail(email);
    expResult.setPassword("cl123");
    expResult.setFirstName("Cami");
    expResult.setLastName("Lei");
    expResult.setDegree("MS");
    expResult.setHelpFutStud(true);
    expResult.setIsLoggedIn(false);

    Student result = StudentUtil.getStudent(email);

    sameStudent(expResult, result);
}

static void sameStudent(Student stud1, Student stud2) {
    boolean success;
    if (stud1.getEmail().equals(stud2.getEmail())
            && stud1.getStudentId().equals(stud2.getStudentId())
            && stud1.getFirstName().equals(stud2.getFirstName())
            && stud1.getLastName().equals(stud2.getLastName())
            && stud1.getPassword().equals(stud2.getPassword())
            && stud1.getDegree().equals(stud2.getDegree())
            && stud1.getStudentAvail() == stud2.getStudentAvail()) {
        success = true;
    } else {
        success = false;
    }
    System.out.println("Students : \n" + stud1 + "\n" + stud2 + "\nAre : " + (success ? "Same" : "Not Same"));
}
```

Figure 18 - getStudent() JUnit test

2.5.1.3. Manual Overall Testing

Once all sandboxed development had been satisfactorily completed we performed an all encompassing integration test.

2.5.2. User Testing and feedback into design

At each integrated test stage we invited the feedback from users, such as friends and family. Feedback was fed into our spiral process and if time permitted improvements were made. One such feedback was on the UI for managing a group, user's felt the drop down and button clicking was overwhelming, so the team member in charge of manage groups (Thil) implemented Java 2D graphics methods to improve the UI aesthetics.

Also, by observing users the team found out many ways a user can make errors in data entry and potentially corrupt our data standard. In the sign-up process users attempted to enter out of format email information. The team member in charge of the majority of data entry into the system in the Login/Signup/Profile creation stages ensured that critical data formats were followed by denying corrupted data from entering our application and notifying users of the error.

2.6. Deployment

StudyBuddy has been packaged in a Java JAR file for distribution to users with an installed Java platform. Users can easily get up and running with the program, given the JAR has been loaded with the correct courses for the academic term.

For the purposes of testing the deployment, please use the following user logins or create your own.

Username: rmiller@scu.edu Password: rm123
Username: tara@scu.edu Password: ta123
Username: glee@scu.edu Password: gl123
Username: erane@scu.edu Password: er123

3. Challenges and Lessons Learned

We as a team wanted to work on a project that would give us a glimpse of how real world projects evolve right from conception to deployment. StudyBuddy provided that experience with the added challenge of being greenfield. Nailing down the initial behaviours to be focused on was difficult as the process was unshackled, doing it again we would place constraints in the form of more direct user assessment of features in the analysis phase to stop minor feature scope creep.

Another challenge, that arose from sandboxed development, was the firming of the data set and standard to use. It was not until week 4 that our database was well defined. This was because the team had to investigate the best method for sharing data across the application, and once MySQL was chosen as the solution the team had to also learn its intricacies. The implemented interface used (Netbeans) and versioning tool (BitBucket) also hampered efficiencies in this sphere, by not translating local databases among members, to mitigate this the team migrated from a local MySQL database to an online version.

Lastly, and most unfortunately, was the disruptions to the roles and responsibilities of team members due to loosing essential personnel. StudyBuddy started as team of five which whittled its way down to the three core members. Some exits from the team were graceful and others not so. Luckily, the software development process and documentation methods used, facilitated the dulling of such disruptions and aided other team members to pick up slack rapidly.

4. Future work

The team is excited to see if the application can be taken further and adopted by Santa Clara University and other educational institutions. To make this happen we would need to provide an interface for schools to modify course data and re-skin our application based on school. We are also investigating this possibility of incorporating chat like features for group members to converse while physically away.