Identified Features
(in order of importance)

1. The project manager can create new projects and define the group size for each project. (FUNDAMENTAL)

2. The project manager can inform the system to fetch a list of eligible members from a database, which are then made into "members" of the system. (FUNDAMENTAL)

3. The system can construct groups based on group size and any matching criteria (if any). (FUNDAMENTAL)

4. The project manager can decide which members can NOT work together, and which members MUST work together, in teams.

5. Members can decide on which other members they want to work with, and which other members they do NOT want to work with (unless otherwise set by the project manager).

6. The project manager can create a list of self-evaluation questions for members to provide answers to. Questions can be added and removed at different points in time.

7. Members can perform self-evaluation questionnaires as provided by the project manager.

8. The system can fetch grades and courses completed by students (for student members).

9. Members can set personal schedules for times they are and aren't available.

10. The system will fetch class schedule (for student members).

11. The project manager can change the composition of groups after they have been automatically made by the system.

The primary features that must be in place before any executable architecture is constructed are the ability to **create a new project**, and the ability to **create "members"** from any list of eligible members. The ability to **compare members based on the required criteria** (i.e. scheduling, self-evaluation answers, etc.) will also be required before the system can function. Without these features that have been identified as fundamental, there can be no automatic team generation, the main function of the software.

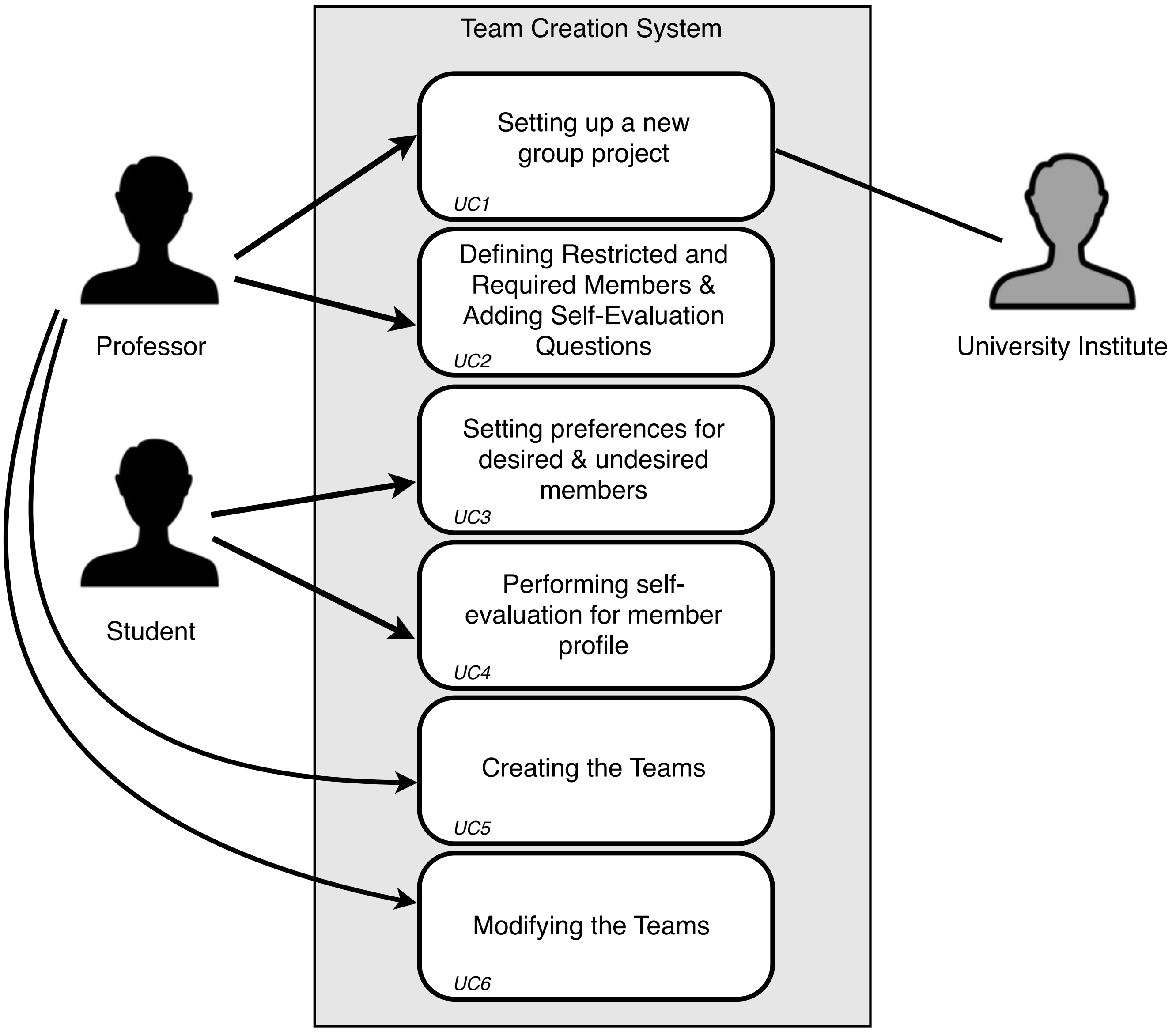
While self-evaluation questions and answers, and student dictated preferences for partners are important for the completion of this software, they can be implemented after, as they are required only for a ***specific*** matching strategy. This strategy can be changed based on project manager requirements.

Use Case Diagram

Starting with the initial three use cases, we have decided that it would be appropriate to separate all use cases into 2 use cases. From use case 1, there is now "***Setting up a new group project***" and "***Defining restricted and required members & adding self-evaluation questions***". From use case 2, there is now "***Setting preferences for desired & undesired members***" and "***Performing self-evaluation for member profile***". For use case 3, there is now "***Creating the Teams***" and "***Modifying the teams***".

Each of these cases describes one major step in setting up new teams for a given project that is defined by the project manager. The total set of use cases outlines the order of events for each new project from creation to team generation and modification.

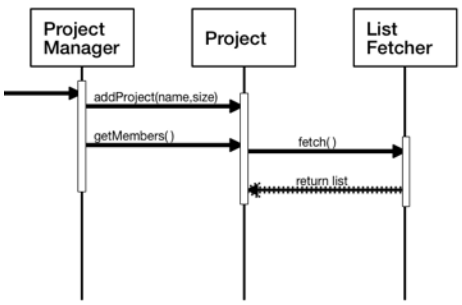
Given this, below is the Use Case Diagram:


Team Creation System
Setting up a new group project
UC1
Defining Restricted and Required Members & Adding Self-Evaluation Questions
UC2
Setting preferences for desired & undesired members
UC3
Performing self-evaluation for member profile
UC4
Creating the Teams
UC5
Modifying the Teams
UC6
Professor
Student
University Institute

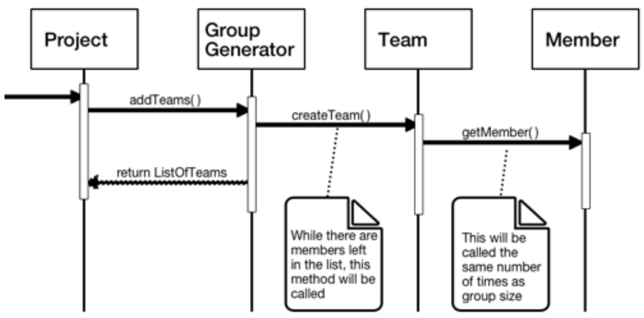
Sequence Diagrams

The following sequence diagrams help to illustrate the interactions between classes of our system for use cases 1 & 3, the main cases for iteration 1. These encapsulate the functioning of the system when constructing a new project, and when generating teams for each project

Use Case 1



Use Case 3


Project
Group Generator
Team
Member
addTeams()
createTeam()
getMember()
return ListOfTeams
While there are members left in the list, this method will be called
This will be called the same number of times as group size

Traceability Matrix

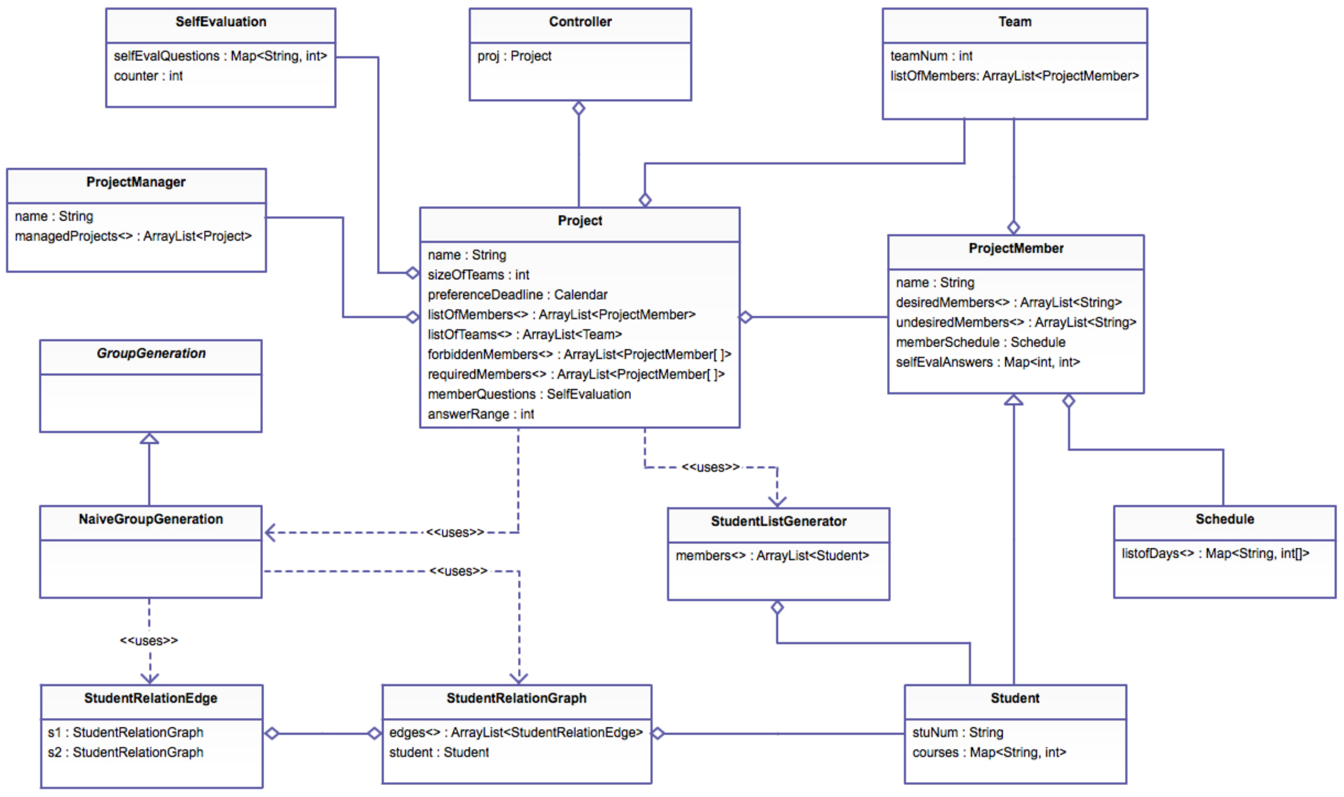
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
UC 1	⤴	⤴								⤴	
UC 2				⤴		⤴					
UC 3					⤴				⤴		
UC 4							⤴				
UC 5			⤴					⤴			
UC 6				⤴							⤴

Updated Domain Model

The updated domain model for our architecture no longer includes a “Main” class, as this was unnecessary and provided no functions. Further, its naming convention was confused with the “main” method of our software. While no new changes have been made to our core classes, we have added a new set of classes for the construction and management of teams, and the matching strategies used to build those teams. While the specific strategy can be specified by the Project Manager, it is irrelevant what the details of the strategy are from the perspective of the program. This will allow different strategies based on what is desired for the project

Group Project: UML Diagram

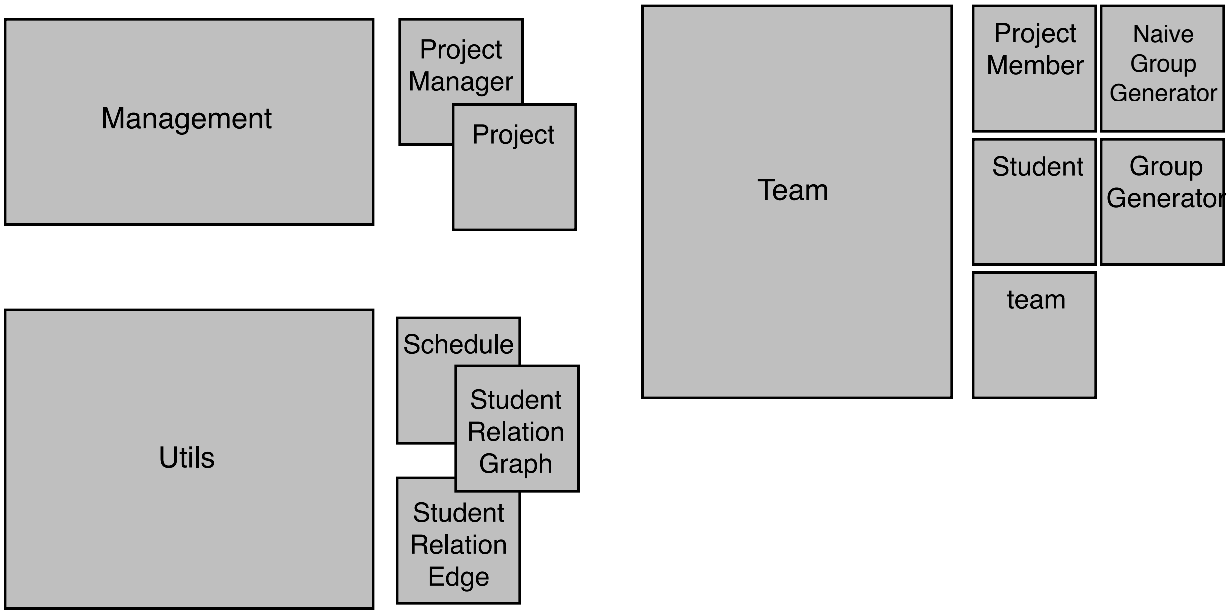
Group C: Andrew DeRoche, Christine Russell, Caroline Strickland, Scott Young



SelfEvaluation
selfEvalQuestions : Map<String, int>
counter : int
Controller
proj : Project
Team
teamNum : int
listOfMembers: ArrayList<ProjectMember>
ProjectManager
name : String
managedProjects<> : ArrayList<Project>
Project
name : String
sizeOfTeams : int
preferenceDeadline : Calendar
listOfMembers<> : ArrayList<ProjectMember>
listOfTeams<> : ArrayList<Team>
forbiddenMembers<> : ArrayList<ProjectMember[ ]>
requiredMembers<> : ArrayList<ProjectMember[ ]>
memberQuestions : SelfEvaluation
answerRange : int
ProjectMember
name : String
desiredMembers<> : ArrayList<String>
undesiredMembers<> : ArrayList<String>
memberSchedule : Schedule
selfEvalAnswers : Map<int, int>
GroupGeneration
NaiveGroupGeneration
<<uses>>
<<uses>>
<<uses>>
<<uses>>
StudentListGenerator
members<> : ArrayList<Student>
Schedule
listofDays<> : Map<String, int[]>
StudentRelationEdge
s1 : StudentRelationGraph
s2 : StudentRelationGraph
StudentRelationGraph
edges<> : ArrayList<StudentRelationEdge>
student : Student
Student
stuNum : String
courses : Map<String, int>

Functional Modules

Identified functional modules are as follows



The Management package will hold the essentials for each project, and the project manager class to manipulate the details of each project. The team package will maintain teams themselves, individual members profiles, and general group generation strategies. The Utils package will maintain individual's schedules, and the graphs that will be constructed by the group generator strategies, which are used to determine the best team composition based on the input of the strategy.